Introduction to Anomaly Detection

Dr. Charlie Abela
Department of Artificial Intelligence
University of Malta

Objectives

Provide a comprehensive overview of anomaly detection, its significance, and its applications.

- DEFINITION AND IMPORTANCE
- ANOMALY DETECTION TECHNIQUES
- TYPES OF ANOMALIES
- CHALLENGES

What is Anomaly Detection?

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

- Anomaly detection is the process of identifying **unusual data points** that do not fit the general pattern.

Situation	Normal data	Anomaly
Credit card usage	Small, local transactions	Sudden $10,000 transaction in another country
Health monitoring	Regular heartbeat patterns	Irregular heartbeats
Manufacturing	Uniform product surface	Defect (scratch, hole) in a product image

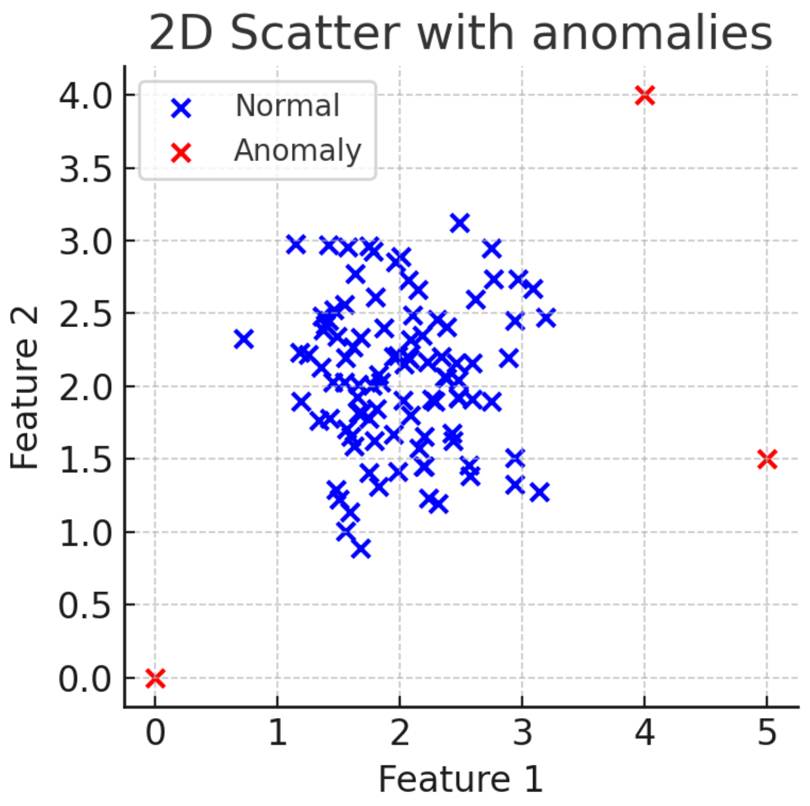
Types of Anomalies

- **Point Anomalies:** An individual data point that is **extreme or unexpected** compared to the rest. It doesn’t fit the normal pattern.
- **Contextual Anomalies:** A data point that is anomalous in a specific context (e.g., time or location) but could be normal in another context.
- **Collective Anomalies:** A collection of data points that as a group exhibit an anomalous pattern, even if each point individually appears normal.
- We need to understand these types to choose the right detection approach (some methods are better for point outliers vs. sequence anomalies, etc.).

Point Anomalies (Outliers)

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

- A **point anomaly** is a single instance that lies outside the expected range or distribution of the data. It's an outlier.
- **Example:** In credit card spending, if typical purchases are $5–$100 in home city, then a $1,000 purchase overseas is a point anomaly.
- Such a lone outlier often indicates an unusual event: e.g., a sensor glitch, fraudulent transaction, or data entry error. Detecting point anomalies is the classic "find the needle in a haystack."



Collective Anomalies

- A **collective anomaly** is a set of data points whose **joint behavior** is anomalous, even if each point looks normal in isolation.
- It's the pattern as a whole that's unexpected. **Example:** A series of heartbeats may individually be normal, but taken together they form an irregular rhythm.
- In security, multiple login attempts across different locations might each seem fine alone, but collectively they form a suspicious pattern (potential distributed attack).
- These require looking at sequences or groups of points. Anomaly detection methods need to capture temporal or spatial relationships to catch collective anomalies.

Contextual Anomalies

- A **contextual anomaly** (or conditional anomaly) is an outlier only under certain context or conditions. The same value could be normal in a different context.
- **Example:** 100°F temperature is normal in a desert afternoon, but would be an anomaly in Antarctica. Or high network traffic at 3AM is abnormal compared to typical nightly patterns.
- Context is key (e.g., time, location, season). An observation is flagged because it deviates from expected behavior given its context, not from the entire dataset overall.
- Detecting contextual anomalies often requires domain knowledge or contextual features (e.g., time of day) and methods like time-series analysis or conditional models.

Statistical Methods: Z-Score (Univariate)

- **Z-Score Method:** For a numeric feature, the z-score of a value x is $z = (x - \mu) / \sigma$ (number of standard deviations from the mean). Points with $|z|$ above a threshold are considered anomalies.
- Assumes data is approximately normal. A common rule: $|z| > 3$ (beyond 3σ) is an outlier (only ~0.3% of normal data exceeds 3σ by chance).
- **Example:** In a dataset of body temperatures ($\mu \approx 98.6$°F, $\sigma \approx 0.7$°F), a reading of 102°F has $z \approx +4.86$, which is far beyond normal variation – a likely anomaly (fever).
- **Pros:** Simple to implement, gives an interpretable “outlier score”. **Cons:** Not reliable if distribution is not normal or data has heavy tails; also only looks at one feature at a time.

Mahalanobis Distance (Multivariate)

- Measures distance of a point **from the center of a multivariate distribution**, taking into account feature correlations. Formula for point $\boldsymbol{x}$:

$$MD(x) = \sqrt{(x-\mu)^T\Sigma^{-1}(x-\mu)}$$ (where Σ is covariance matrix)

- Unlike Euclidean distance, MD is **scale-invariant and correlation-aware**: it “whitens” data by covariance, so correlated features don’t inflate distance.
- **Outlier detection:** For roughly normal data, MD^2 follows a χ^2 distribution. We flag x as an outlier if $MD(x)$ is very large (p-value below threshold, e.g. 0.01). Points beyond the 99th percentile of MD are anomalies.
- **Example:** Height vs. weight: Euclidean might label a very tall, heavy person as far from average. Mahalanobis would recognize height and weight are correlated – that person might not be an anomaly if their weight is proportional to height.

Z-Score & Mahalanobis: Code Example

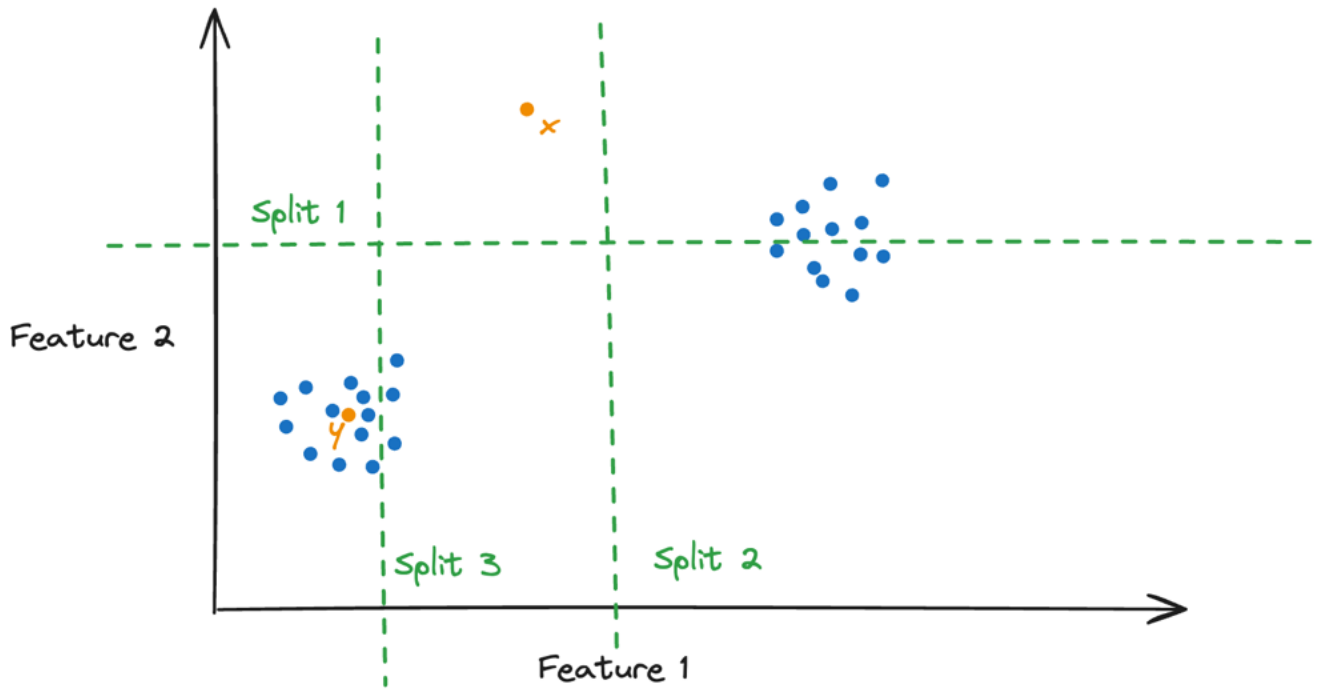
L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

```python
import numpy as np
from scipy import stats

# 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(row):
    diff = row - mu
    return np.sqrt(diff.T.dot(inv_cov).dot(diff))
md = np.apply_along_axis(mahalanobis, 1, X)
outliers_md = np.where(md > np.percentile(md, 99))  # flag top 1% distances
```

This code computes z-scores and Mahalanobis distances, then flags outliers (|z|>3 or MD beyond 99th percentile).

Isolation Forest (IF) – Intuition

- Random splits isolate an outlier (orange **x**) quickly in just 2 splits, whereas a normal point (orange **y** in cluster) requires more splits. Anomalies end up with shorter tree paths on average.



Isolation Forest (IF)

- **Isolation Forest:** An ensemble of random binary trees that **explicitly isolates observations**. The idea: anomalies are easier to isolate because they differ markedly from the rest.
- **Mechanism:** Randomly pick a feature and a random split value; this partitioning forms a tree. Continue splitting subsets until each point is isolated. Repeat to build many trees (a forest).
- An anomaly (outlier) tends to be isolated in just a few splits (short path length), whereas normal points require more splits to isolate (longer path).
- **Anomaly Score:** In IF, score is based on the **average path length** for a point across all trees, normalized for tree size. Shorter average path = more anomalous. Scores are typically scaled to [0,1], where 1 means very likely anomaly.

Isolation Forest (IF): Algorithm & Features

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

Training Isolation Trees:

1. For each tree, take a random subsample of data.
2. While not isolated: choose a random feature and a random split value between min and max of that feature. Split the node into two branches (<= split, > split).
3. Recursively partition until each observation is isolated in its own leaf, or tree reaches max height.

Isolation Forest (IF): Algorithm & Features

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

- **Ensemble:** Combine T isolation trees. Compute path length $h(x)$ for point x in each tree (number of splits until isolation). The expected path length $E[h(x)]$ is low for anomalies.
- **Scoring:** Compute $s(x) = 2^{-\frac{E[h(x)]}{c(n)}}$ where $c(n)$ is average path length of unsuccessful search in a Binary Search Tree. This formula yields scores ~0.5 for normal points, closer to 1 for anomalies.
- **Advantages:** Unsupervised, no distribution assumptions, works well in high dimensions. Linear time complexity $O(n \log n)$.
- **Limitations:** Requires setting contamination (expected fraction of outliers) for thresholding. Many false positives if data is highly imbalanced.

Isolation Forest (IF): Usage Example

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

# Load data (e.g., credit card transactions dataset)
data = pd.read_csv("creditcard.csv")
X = data.drop(columns=["Class"])          # features
model = IsolationForest(contamination=0.001, random_state=42)
model.fit(X)

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

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

- In this example, we train an Isolation Forest on credit card data.
- The contamination rate (0.1%) is set based on expected fraud rate.
- We then obtain anomaly scores and identify the anomalous transactions.

One-Class SVM (OC-SVM)

- **One-Class SVM:** An algorithm that learns the boundary of “normal” data in feature space, and flags points outside this boundary as anomalies. It’s essentially a support vector machine trained on one class (the normal data).
- OC-SVM finds a **maximally distant hyperplane from the origin** that separates most data points from the origin with a margin. Intuitively, it tries to include the normal instances in a “small” region around the origin, leaving outliers outside.
- Often uses the **RBF kernel** to form a nonlinear boundary (like an enclosing contour around data). The parameter ν controls the fraction of points allowed outside the frontier (the outliers).
- **Result:** A decision function $f(x)$ where $f(x) < 0$ indicates an anomaly and $f(x) > 0$ indicates an inlier (for sklearn’s OneClassSVM). The model is basically learning the support of the normal data distribution.
- **Use Cases:** Appropriate for novelty detection when you have plenty of “normal” data to train on but want to detect if a new point is unlike those. E.g., train on normal network traffic to detect new types of attacks.

One-Class SVM (OC-SVM)

- **Training:** Only “normal” data is used. The algorithm solves an optimization to find a hyperplane that **maximizes the margin** from the origin, while permitting some fraction ν of data to lie outside (outliers).
- **Kernel Trick:** By using a kernel (e.g., RBF), OC-SVM can learn a **non-linear boundary** in original space (a complex shaped envelope around normal data). This allows capturing non-convex normal regions.
- **Tuning:** ν (nu) is the proportion of outliers expected (or an upper bound) – higher ν makes the boundary looser (more points can be outliers). The kernel bandwidth (e.g., gamma in RBF) must be set to reflect data scale; if too large or small, you might get all points inside or all flagged out.
- **Output:** The sign of decision function tells anomaly vs normal. Many implementations also offer an anomaly score. In sklearn, `OneClassSVM.predict` gives +1 or -1, and `decision_function` gives the raw distance to boundary.
- **Limitations:** OC-SVM can be sensitive to high dimensionality (curse of dimensionality) and scaling. Also, training complexity is higher (requires solving quadratic programming, not as scalable as IF).

One-Class SVM (OC-SVM)

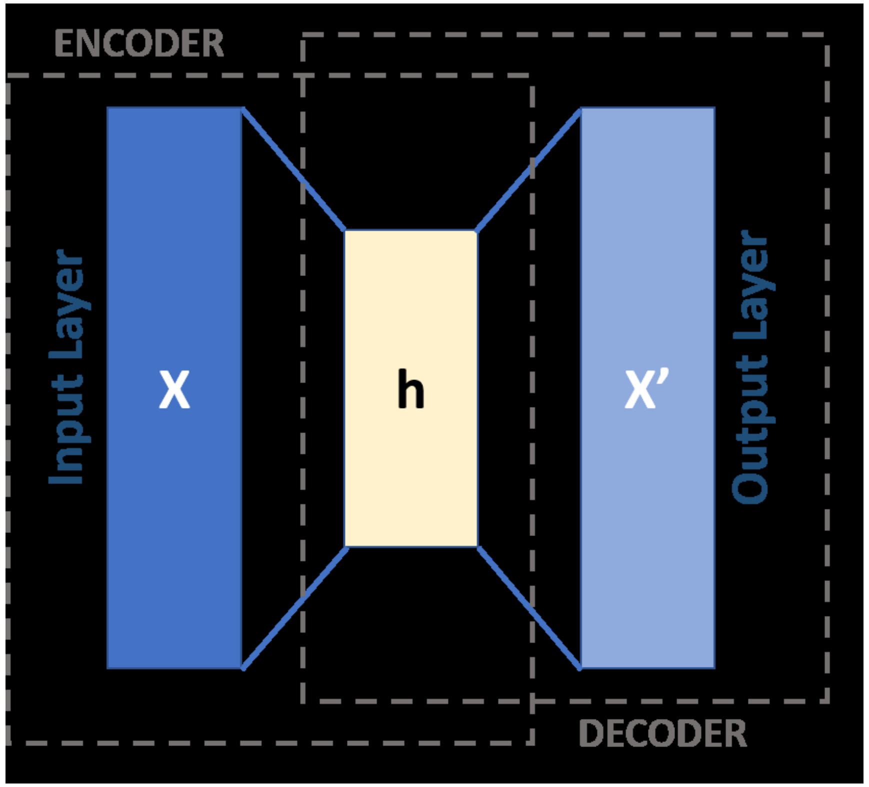
```python
from sklearn.svm import OneClassSVM

# Using a small subset of Iris dataset as normal data (e.g., only setosa class as normal)
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]
```

- This trains an OC-SVM on a normal subset (setosa Iris flowers).
- The RBF kernel with $\nu=0.05$ lets ~5% of points be outliers.
- Predictions on the full Iris dataset should flag many virginica and versicolor as anomalies (since the model only knows setosa).

Anomaly Detection: Autoencoders

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

- Autoencoder architecture:
- The **encoder** (left) compresses input X into a lower-dimensional code $\boldsymbol{h}$,
- Then the **decoder** (right) reconstructs X' from h.
- For anomaly detection, differences between X and X' (reconstruction error) are key.



Autoencoders - Concept

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

- An **Autoencoder (AE)** is a neural network that learns to reproduce its input at the output. It consists of two parts: an **encoder** f that maps input $X \rightarrow h$ (latent representation), and a **decoder** g that maps $h \rightarrow X'$ (reconstructed input).
- The network is trained to minimize the mean squared reconstruction error

$$\mathrm{L}(\mathrm{X}, X') = \frac{1}{d} \sum_{i=1}^{d} (x_i - x_i')^2$$

where d is the number of input features. This loss function penalizes deviations between the input X and its reconstruction X', making it suitable for continuous data.

The autoencoder is thus trained to learn a compressed representation of the normal patterns in the training data, similar in spirit to a non-linear form of Principal Component Analysis (PCA).

Autoencoders - Concept

- **Undercomplete AE:** We choose a small dimension for code h (bottleneck), forcing the model to learn the most important aspects of the data. After training on mostly normal data, the AE will reconstruct normal inputs well (yielding low reconstruction error) but will **fail to reconstruct anomalies** accurately (higher reconstruction error).
- **Why**: The AE cannot accurately encode or reconstruct feature patterns it has rarely or never seen during training. Therefore, we can define an **anomaly score** based on the reconstruction error:

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

- If this error $E(X)$ exceeds a predefined threshold, the input X is flagged as an anomaly.

Autoencoders for Anomaly Detection - Details

- **Training Phase:** Feed many normal examples through the autoencoder. The model learns to compress and decompress these examples, capturing the regularities of normal data. The objective is to minimize **mean squared error (MSE)** between input and output:

$$\mathrm{L}(\mathrm{X}, X') = \frac{1}{d}\sum_{i=1}^{d}(x_i - x_i')^2$$

  Typically use stochastic gradient descent/backpropagation.

- **Anomaly Scoring:** For a new instance, compute the reconstruction error

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

  (i.e., the average of squared differences per feature). Anomalies will have significantly larger E values, since the model cannot accurately reconstruct novel or rare patterns it has not seen during training.

Autoencoders for Anomaly Detection - Details

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

- **Threshold Selection:** Determine a threshold on E (e.g., based on a percentile of error on validation data or statistical significance). If $E >$ threshold, label the instance as an anomaly. Sometimes one uses the distribution of reconstruction errors on known normal data to set this threshold
- **Extensions:** Instead of raw MSE, one can use more complex AEs (e.g., denoising autoencoders, variational AEs) or other loss functions (like binary cross-entropy for binary data). For time-series anomalies, sequence autoencoders (e.g., LSTM-based) are often used

Autoencoders Example

- This code trains a basic autoencoder on normal training data (e.g., non-fraud transactions).
- We then compute reconstruction MSE for test instances.
- Those with highest 5% errors are flagged as anomalies.

```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')
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 as anomalies
```

Evaluation Metrics

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

  $$Precision = \frac{TP}{TP+FP}$$ (High precision = few false alarms)

- **Recall (Detection Rate):** Among all true anomalies, how many did we catch?

  $$\mathrm{Re}call = \frac{TP}{TP+FN}$$ (High recall = catching most anomalies, but could include more false positives).

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

  $$\mathrm{F1} = 2\frac{PR \cdot TP}{PR + RC}$$ (A balanced measure useful when we want a single metric to optimize)

- **Accuracy** is often not informative in anomaly detection because of class imbalance (e.g., 99.9% of transactions are normal, a trivial detector can be 99.9% "accurate" by labeling everything normal).
- **ROC Curve:** Plots True Positive Rate vs False Positive Rate as the anomaly score threshold varies. **ROC AUC** is threshold-independent measure of ranking quality. However, with extreme imbalance, ROC can be misleading (a high TPR and low FPR might still correspond to many false positives in absolute terms).
- **Precision-Recall (PR) Curve:** Plots Precision vs Recall for varying thresholds. **PR AUC** focuses on the positive class (anomalies) and is more informative under class imbalance. In highly imbalanced scenarios, AUPRC is often the preferred metric.

ROC & PR Curve: when to use which

- **ROC Curve:** Shows trade-off between sensitivity (TPR) and fallout (FPR). It treats all classes equally. If false positives (FP) are not extremely numerous relative to true negatives, ROC AUC is okay. But when anomalies are very rare, ROC can give an overly optimistic picture (since even a lot of FP may be a tiny FPR given huge normal count).
- **Precision-Recall Curve:** Focuses on performance on the positive (anomaly) class. When anomalies are rare, precision drops quickly as you include more detections (increase recall) and vice versa. PR curve is more sensitive to changes in FP when positives are rare.
- **Interpretation:** On a PR curve, a method that maintains high precision at a decent recall is desirable (e.g., in fraud detection, we want to catch many frauds (high recall) but without too many false alarms to investigate (precision)).
- In practice, with imbalanced data, **PR AUC is often more indicative** of algorithm performance. A low PR AUC (near 0) means either low recall or low precision or both – potentially unacceptable in critical anomaly detection tasks.
- **Example:** For credit card fraud (0.17% fraud rate), an Isolation Forest might achieve ROC AUC ~0.97 (great separation) but PR AUC only ~0.22 – indicating that, at high recall, precision is very low (many FPs).

Choosing Thresholds

L-Università ta' Malta
Faculty of Information & Communication Technology | Department of Artificial Intelligence

- An anomaly detector often produces a **continuous score** (e.g., anomaly score or probability). We must choose a threshold to classify anomalies vs normal. This is a critical decision balancing false positives and false negatives.
- **Methods to select threshold:**
    - **Manual/Domain-driven:** Determine what false alarm rate is tolerable. E.g., in medical screening, you may set a threshold to ensure 99% sensitivity (recall), accepting some FP.
    - **Statistical quantile:** Assume a certain contamination rate. E.g., flag top 1% highest anomaly scores as anomalies (often used in unsupervised settings when no labels are available).
    - **Maximizing a metric:** If you have labelled validation data, choose the threshold that maximizes F1-score, or the point on the PR curve with the best trade-off (maybe using the Youden's J statistic on ROC or similar).
    - **Cost-based optimization:** If you can assign a cost to false alarm vs missed anomaly, choose threshold minimizing total cost.
- **Example:** In our autoencoder model, we might observe reconstruction errors cluster near 0 for normals and a long tail for anomalies. We could pick a threshold at, say, $\mu\_normal + 3\sigma\_normal$ of reconstruction error. Or use the elbow of the precision-recall curve.
- It's often useful to visualize the score distribution for known normal vs anomaly in validation to pick a sensible cutoff.

Explainability

- As we deploy complex anomaly detection models (IF, OC-SVM, neural nets), understanding **why** a point was flagged is crucial. Reasons:
- **Trust and Verification:** In high-stakes areas (finance, healthcare), analysts or operators need to verify that an alert is valid. Explanations provide insight into the factors driving the anomaly detection.
- **Actionability:** Knowing *which features* are anomalous helps direct corrective action. (E.g., “Transaction flagged due to unusual high amount and foreign location” is more actionable than a cryptic alert.)
- **Model Improvement:** Explanations of false positives can reveal model blind spots or data issues, guiding improvements.
- **Regulations and Ethics:** For decisions affecting individuals (fraud blocking, medical diagnosis), interpretable results may be required by law or ethical practice.
- **Challenge:** Many anomaly detectors are black boxes (especially ensemble and deep models). But we can apply eXplainable AI (XAI) techniques to interpret them.

SHAP for Model Explainability

- **SHAP (SHapley Additive exPlanations):** A unified framework based on Shapley values from cooperative game theory for explaining model outputs. It treats each feature as a "player" in a game where the prediction is the payout. The **Shapley value** for a feature is the average contribution of that feature to the prediction, over all feature orderings.
- Key properties of SHAP: **Additivity** (feature contributions sum to the difference between the model output and a baseline expectation) and **fair credit allocation** (each feature's influence is fairly considered).
- **Baseline:** Usually the average model output on a reference dataset. For anomalies, baseline might be the expected "normality score". SHAP values explain how much each feature shifts the model output from that baseline.
- **Output:** For each prediction, you get a set of SHAP values (one per feature). A positive SHAP value means the feature pushed the model output higher (toward anomaly if we define output as anomaly score), and negative means it pushed it lower (toward normal).
- **Why SHAP for anomalies?** It provides **local explanations** – e.g., for a flagged transaction, SHAP can tell us which features (amount, merchant, time, etc.) contributed most to it being considered anomalous. It also provides **global insight** by averaging absolute SHAP values to rank overall important factors.

Explaining Unsupervised Models with SHAP

- Even for unsupervised detectors like IF or autoencoder, we can apply SHAP by treating the model's anomaly score as a "prediction" to explain. For Isolation Forest (a tree model), we can use TreeSHAP; for others, KernelSHAP or sampling-based approaches.
- **Isolation Forest + SHAP:** Since IF is tree-based, TreeExplainer can compute exact Shapley values. We obtain how each feature contributed to the anomaly score for a given point. For example, "feature X had a +0.5 SHAP value, meaning it pushed the anomaly score up by 0.5 (making the point more anomalous)".
- **Autoencoder + SHAP:** One approach is to treat the reconstruction error as the output and use Kernel SHAP. Alternatively, train a surrogate model (like a Random Forest) on the binary labels (normal/anomaly) and explain that. In any case, SHAP can help attribute high error to specific features — e.g., "Feature A's value was largely different from what the autoencoder expects, contributing most to the error."
- **Outcome:** We get insight into *why* an instance is anomalous: e.g., "Transaction was flagged mainly due to unusually high TransactionAmount and MerchantCountry being rare for this card" – features with largest positive SHAP values toward anomaly. This transforms a black-box alert into an explainable one.

Isolation Forest with SHAP

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

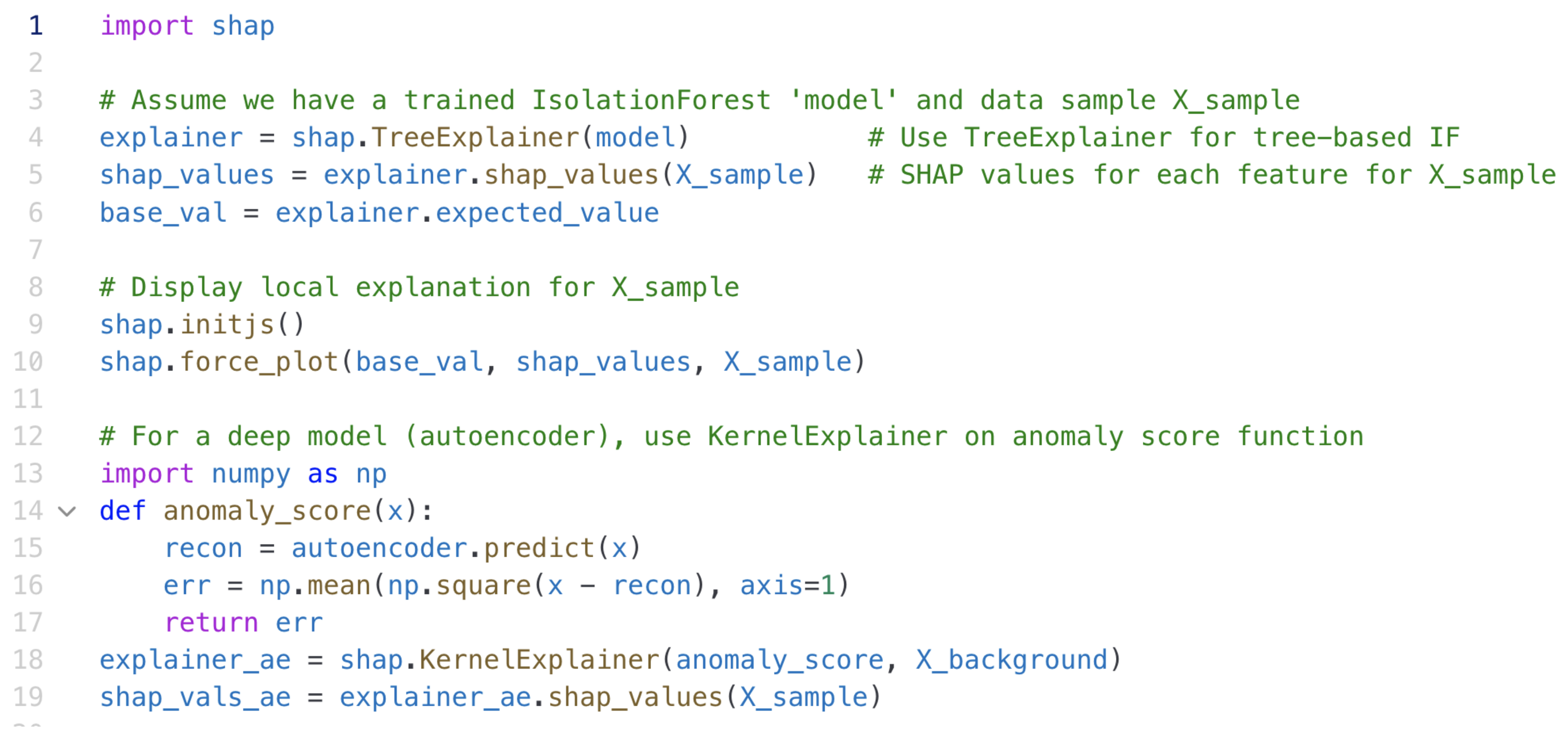
- Consider an Isolation Forest that flagged a network log entry as anomalous. Using SHAP, we explain the anomaly score:
    - **Feature contributions:** e.g., LoginTime (3 AM) has a SHAP value of +0.3 (increasing anomaly score) because it's an uncommon login hour. IPRegionMismatch contributed +0.5 (login from a new region), etc. Meanwhile, other features like LoginDevice might have a negative SHAP (–0.1) indicating it's common and actually pulls the score toward normal.
    - **Waterfall Plot:** A visualization can stack these contributions: start from the baseline (expected score). Features with **blue bars (negative SHAP)** push the score down towards normal, and **red bars (positive SHAP)** push it up towards anomalous. The final score is the baseline plus all contributions.
    - In a typical waterfall the length of each bar shows magnitude of that feature's impact. For our example, we might see IPRegionMismatch and LoginTime as the largest red bars, explaining why the model deemed this event anomalous.
- **Global Explanation:** By computing SHAP values for many points, we can rank feature importance. For instance, a SHAP summary plot might show TransactionAmount and TransactionType have the highest average |SHAP| values for fraud detection, meaning they heavily influence anomaly decisions in general.

SHAP Visualisation Tools

- **Waterfall Plot (Local):** Shows how each feature's SHAP value takes the model output from the baseline to the final prediction. Useful for a detailed explanation of a single anomaly. We see exactly how much each feature added or subtracted from the anomaly score. (Blue indicating a feature made the instance more normal, red indicating more anomalous.)
- **Force Plot (Local):** Similar information as waterfall, but in a visual "push-pull" format on a number line. It shows the baseline score and arrows pushing towards anomalous or normal side. This can be more intuitive for non-technical stakeholders.
- **Summary Plot (Global):** Combines feature importance with feature effects. Typically, a bee-swarm plot where each point is a SHAP value for an instance, on a horizontal axis (impact on model output), and features on vertical axis sorted by overall importance. Colour often indicates feature value (high or low). This plot tells us which features are most influential and how their values relate to anomaly outcome.
- **Bar Chart of Mean |SHAP| (Global):** Simply the average absolute SHAP value per feature, to rank features by overall influence. E.g., we might see that "Amount" has the highest importance score for fraud model, followed by "Location" etc., confirming domain intuition.

Explainability in code

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence



```python
import shap

# Assume we have a trained IsolationForest 'model' and data sample X_sample
explainer = shap.TreeExplainer(model)           # Use TreeExplainer for tree-based IF
shap_values = explainer.shap_values(X_sample)   # SHAP values for each feature for X_sample
base_val = explainer.expected_value

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

# For a deep model (autoencoder), use KernelExplainer on anomaly score function
import numpy as np
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)
```

- In this snippet, we use SHAP to explain an Isolation Forest's decision on a sample (using `TreeExplainer`).
- We also illustrate using `KernelExplainer` for an autoencoder's reconstruction error.
- The `force_plot` (when run in a Jupyter notebook) would show the contribution of each feature for the sample's anomaly score.

Resources

- Check Anomaly Detection Workshop on GitHub
  - Notebooks
  - Slides
  - Detailed notes

Questions?