# **Anomaly Detection**

#### using Isolation Forest, One-class SVM, and Local Outlier Factor

### **Introduction -**

Anomaly detection, also known as outlier detection, is an important data analysis technique used to identify patterns or observations that deviate significantly from the expected norm within a dataset. These deviations from the expected norms are termed anomalies or outliers. They represent rare events or unusual data points that differ from the majority of the data. Anomalies indicate a wide range of issues which can be some technical faults and fraud or innovative and essential phenomena.

In statistical terms, an anomaly is a data point that does not fit the expected distribution or pattern of the dataset. It is a subset of data mining that deals with identifying these irregularities and understanding their implications. Anomalies are often rare and may occur occasionally or periodically in some cases, making their detection challenging but crucial.

Now let's understand why the detection of anomalies is crucial.

### **Importance of Anomaly Detection**

### **1. Ensuring Data Integrity:**

Anomaly detection is essential for upholding the relevance of the information at hand. By eliminating or correcting the outliers, data analysts can be confident that the data which their studies will rely on is correct. This is very important for predictive modelling since the presence of outliers may complicate results and lead to false deductions.

#### **2. Early Detection of Issues:**

Anomaly detection is quite advantageous as it also allows problems to be identified well ahead of time. For example, if there is a considerable drop in sales at a particular point in time, or if there is a noticeable increase in website traffic or even strange behaviour from sensors, these instances need to be corrected as early as possible so as to avert extreme cases in the future.

#### **3. Improving Decision-Making:**

The decision-making process benefits from the inclusion of anomaly detection in the processes of data analysis. It is through the understanding of anomalies that an organisation can improve its working, discover new areas, or mitigate threats that may not be visible through conventional analytical approaches. As a result, decision making becomes better.

#### **4. Enhancing Security Measures:**

Anomaly detection finds its security applications in cybersecurity. By detection on behaviour patterns that are statistically abnormal such as unauthorised access or transfers of unusually large volumes of data, it is possible for organisations to counteract possible security threats in due time, before they are able to inflict great damage. This zealous attitude to security is highly relevant in the area of sensitive information repository and sustaining the trust of customers and stakeholders.

#### **5. Optimising Operational Efficiency:**

In fields such as manufacturing or logistics, where there are clear operational imperatives, available anomaly detection tools optimise work processes by highlighting operational inefficiencies and potential failures. In this instance, monitoring these elements allows the company to keep within the normal scope and avert the risks of a catastrophic breakdown, hence maintenance is conducted only when it is due and required rather than forced.

#### **6. Supporting Innovation:**

Anomaly detection can also drive innovation by uncovering new patterns and trends that might have otherwise gone unnoticed. By exploring these anomalies, organisations can develop new products, services, or strategies that address emerging needs or capitalise on new market opportunities.

#### **7. Facilitating Compliance and Risk Management:**

In regulated industries like finance and healthcare, detecting anomalies is crucial to compliance with laws and regulations. It helps organisations identify and investigate violations that could indicate non-compliance or risk, thus avoiding fines and maintaining good relations with regulators.

#### **8. Improving Customer Experience:**

Fault detection can improve customer satisfaction by identifying and resolving issues that impact customers. For example, e-commerce platforms can use vulnerabilities to detect and fix issues such as payment errors or unnecessary delays in order fulfilment, which can enable competition and be good for customers.

By spotting anomalies, problems can be mitigated before they can grow into much larger inconveniences. Seeing as how there exist a variety of anomalies and ways in which these anomalies occur, it is important to select an appropriate method for detection. So far, a number of methods have been provided to detect these outliers according to the various types of data and requirements. Out of these, Isolation Forest, One-Class SVM and Local Outlier Factor are notable techniques that use different means of detecting the anomalies.

Let's understand each one of them in detail with examples.

### **Isolation Forest Method**

#### **1. Introduction**

Detecting anomalies is an important assignment in a number of spheres since it allows one to identify the values that have notably differed from the expected ones. Out of the most efficient algorithms created for this purpose, the Isolation Forest (iForest) is worth noting. Its distinctive feature is that it concentrates on the resolution of the problem of isolation of the observation rather than the problem of profiling the normal behaviour. This approach is especially useful in the applications involving large and high dimensional data sets hence is widely used in solving real world problems.

#### **2. How Isolation Forest Works.**

The Isolation Forest algorithm works on the principle that unusual observations are rare and distinct, making them easier to isolate than normal observations. This algorithm separates observations by randomly selecting a feature and randomly selecting the separation of the maximum and minimum values ​​of the selected feature. This process is repeated to create a tree structure where each analysis is isolated on a branch.

* **Isolation via Random Splitting**: The key idea is that the more random splits required to isolate an observation, the more likely it is to be normal. Anomalies, being few and distinct, typically require fewer splits to be isolated.
* **Tree Construction**: The algorithm builds multiple trees, forming a forest. Each tree is constructed by randomly selecting features and split values, ensuring diversity in the isolation paths.
* **Scoring Anomalies**: The anomaly score is based on the path length—the number of splits required to isolate an observation. Shorter paths indicate anomalies, while longer paths suggest normal observations.

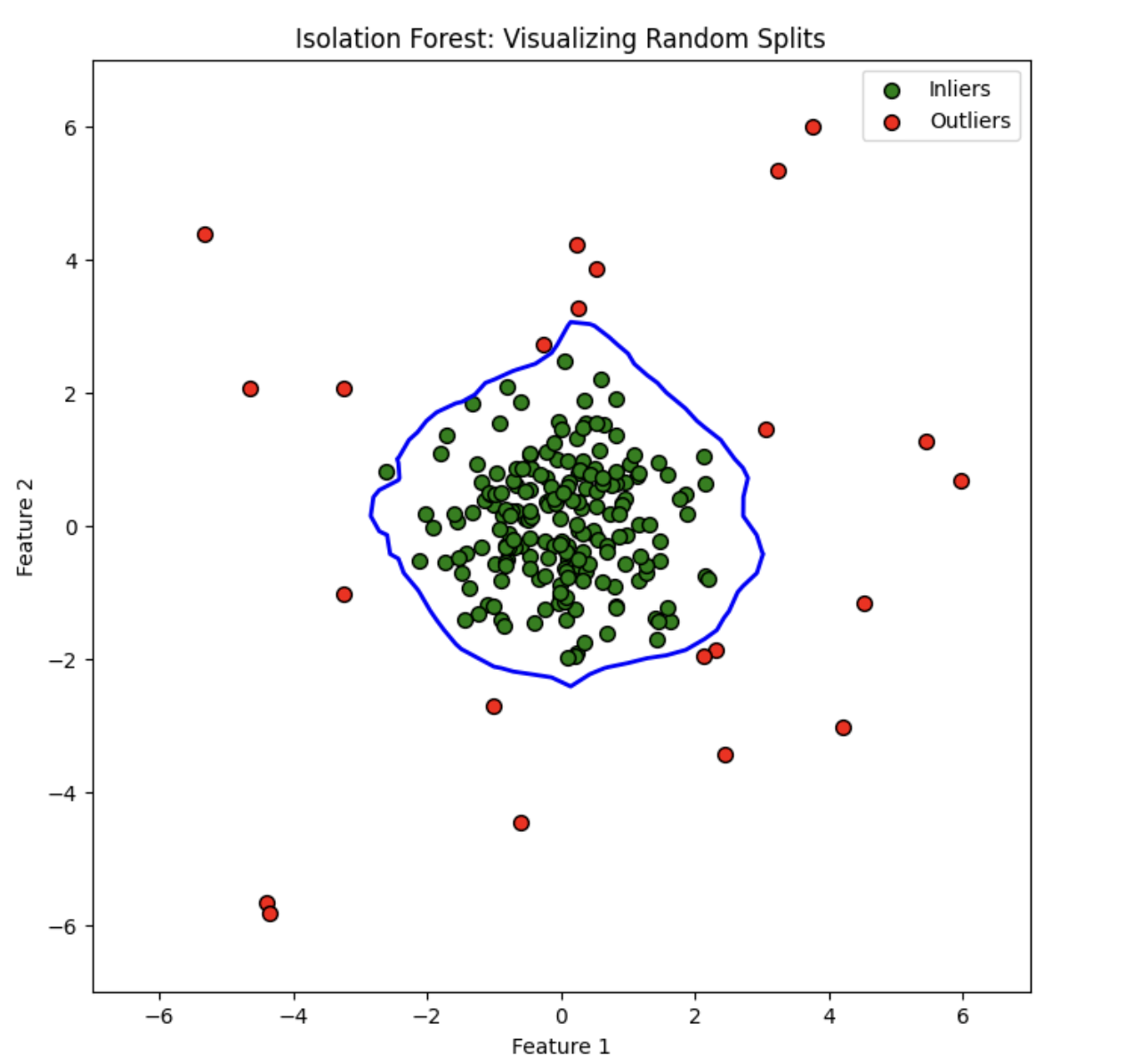
#### **3. Steps in iForest Method :**

1. **Random Subsampling**: A subset of the data is selected to build each tree, ensuring that the trees are not too large and the model remains efficient.
2. **Random Feature Selection**: For each node in the tree, a feature is chosen at random.
3. **Random Split**: A random split value is selected within the range of the chosen feature.
4. **Recursive Partitioning**: The data is split recursively until each observation is isolated or the tree reaches a specified maximum height.
5. **Anomaly Scoring**: The length of the path taken by an observation in the tree is recorded. The anomaly score is the average path length across all trees in the forest. Shorter paths imply higher anomaly scores.

The efficiency of Isolation Forest stems from its ability to perform well without any assumptions about the distribution of the data. The randomness in feature selection and split values ensures that the model captures a wide variety of possible anomaly patterns.

#### **4. Visualizing Isolation Forest**

To better understand how Isolation Forest isolates anomalies, consider the following diagram :



The diagram shows a 2-dimensional plane where the samples are distributed as per their features. Normal or inlier points are shown as green dots clumped together. Outlier or anomalous points appear as random red splotches strewn across space. The blue contour defines the parameters of the Isolation Forest algorithm and illustrates the areas where the points are likely to be isolated by the algorithm. Outliers are inside such a boundary, indicating that the point concerns less number of splits to be isolated from the rest.

#### **5. Coding Example Using Scikit-Learn**

Take an example of a tech company that provides cloud services to businesses around the world. Recently, they've noticed unusual spikes in their server traffic during odd hours, which might indicate a security breach or bot attack. To protect their infrastructure and ensure smooth operations, the company decides to use the Isolation Forest algorithm to detect anomalies in their server traffic data.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest

# Simulate server traffic data (normal and anomalies)

np.random.seed(42)

# Normal traffic: 1000 samples of normal traffic, with some variations

normal\_traffic = np.random.normal(loc=100, scale=10, size=(1000, 2))

# Anomalies: 20 samples representing unusual traffic spikes

anomalous\_traffic = np.random.normal(loc=200, scale=50, size=(20, 2))

# Combine the normal and anomalous traffic

server\_traffic\_data = np.concatenate([normal\_traffic, anomalous\_traffic], axis=0)

# Apply Isolation Forest to detect anomalies

iso\_forest = IsolationForest(contamination=0.02, random\_state=42)

y\_pred = iso\_forest.fit\_predict(server\_traffic\_data)

# Separate inliers and outliers based on predictions

inliers = server\_traffic\_data[y\_pred == 1]

outliers = server\_traffic\_data[y\_pred == -1]

# Visualise the server traffic and detected anomalies

plt.figure(figsize=(8, 8))

plt.scatter(inliers[:, 0], inliers[:, 1], c='green', s=50, edgecolor='k', label='Normal Traffic')

plt.scatter(outliers[:, 0], outliers[:, 1], c='red', s=50, edgecolor='k', label='Anomalous Traffic')

plt.title("XYZzy Inc.: Anomaly Detection in Server Traffic")

plt.xlabel("Traffic Feature 1")

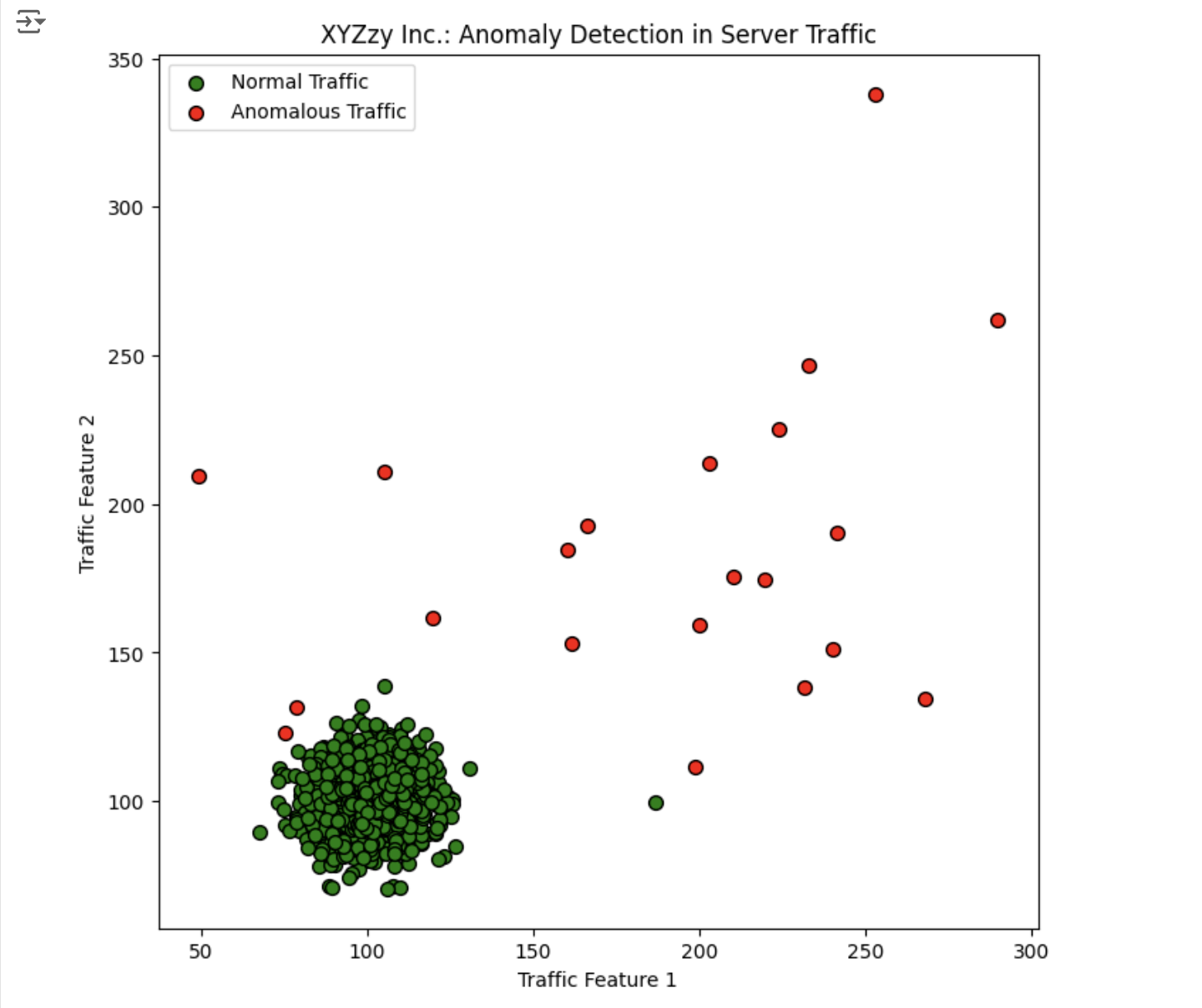
plt.ylabel("Traffic Feature 2")

plt.legend()

plt.show()

### **Explanation:**

* **Normal Traffic**: Simulated as 1,000 samples with typical server activity, centred around 100 units with slight variations.
* **Anomalous Traffic**: Represented by 20 samples showing unusual spikes in traffic, centred around 200 units.
* **Isolation Forest**: The model is trained on this data to identify anomalies. The contamination parameter is set to 0.02, assuming that about 2% of the data might be anomalous.
* **Visualisation**: The plot shows normal traffic in green and detected anomalies in red, helping the organisation to quickly identify and investigate unusual server activities.



This approach helps the company to protect its servers by efficiently detecting and addressing potential security threats in real time.

#### **6. Advantages**

Isolation Forest offers several advantages:

* **Scalability**: It handles large datasets effectively due to its linear time complexity.
* **No Assumptions**: It doesn't require any assumptions about the underlying data distribution.
* **Versatility**: It works well with both univariate and multivariate data.

**7. Applications**:

* **Fraud Detection**: Identifying unusual transactions in financial systems.
* **Network Security**: Detecting unusual patterns in network traffic.
* **Industrial Monitoring**: Spotting equipment failures or unusual operational behaviour.

Hence, classification forests are powerful tools for detecting anomalies that use the concept of classification data to distinguish normal and abnormal observations. Its efficiency and ability to work in multiple domains make it a valuable data scientist tool. Whether you are working to detect fraud, monitor business systems, or analyse network connections, Isolation Forests offers powerful solutions for rare but important detections.

### **One-Class SVM (Support Vector Machine)**

#### **1. Introduction**

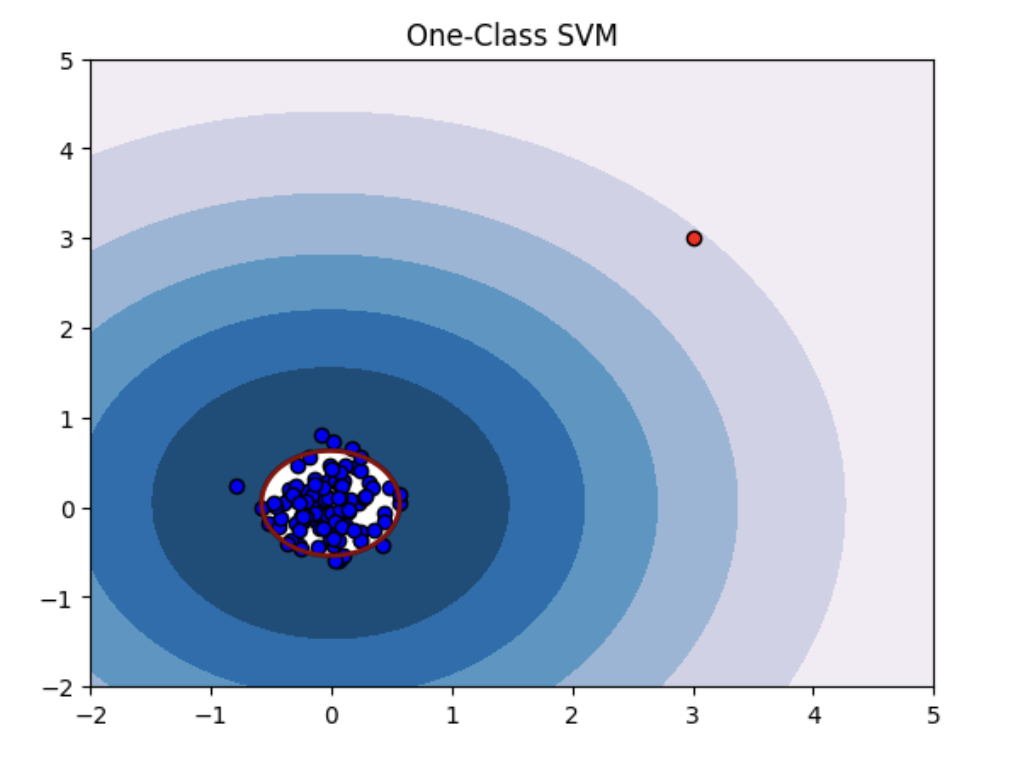
One-Class SVM is a variation of the Support Vector Machine (SVM) algorithm primarily used for anomaly detection. Unlike traditional SVMs that are used for binary classification, One-Class SVM is designed for scenarios where you only have data from one class, typically the "normal" class, and you want to identify outliers or anomalies that do not conform to this class.

#### **2. How One-Class SVM Works.**

The One-Class SVM algorithm learns a decision function for anomaly detection. It tries to capture the regions in the data space where the normal data points (the majority class) are concentrated. The goal is to find a boundary (a hyperplane in high-dimensional space) that separates the normal data points from the origin as much as possible. Points falling outside this boundary are considered anomalies.

**3. Steps in One-Class SVM:**

1. **Input Data:** The algorithm is trained on data that is mostly or entirely from one class (normal data).
2. **Transformation:** The data is transformed into a higher-dimensional space using a kernel function (often the Radial Basis Function (RBF) kernel).
3. **Hyperplane Construction:** The algorithm constructs a hyperplane in this transformed space that best separates the normal data points from the origin.
4. **Decision Function:** The model uses this hyperplane to classify new data points. Points that fall on one side of the hyperplane are considered normal, while those on the other side are labelled as anomalies.

**4. Diagram illustrating One-Class SVM :**

The diagram demonstrates One-Class SVM for anomaly detection. It shows normal data points (blue) clustered in a certain area and an outlier (red) outside this region. The model creates a decision boundary, often curved, to distinguish between normal data and anomalies. The red point, being outside the boundary, is flagged as an anomaly.

The diagram effectively demonstrates how One-Class SVM differentiates between normal data and outliers based on the training it received on normal data.

**5. Coding Example**

Consider the example of a bank, “Quantum Trust”, facing a crisis—suspicious transactions were slipping through their security. Alex, the data scientist, was tasked with finding the culprits. Using a special tool called One-Class SVM, Alex trained a model to detect unusual patterns in transaction data. This model became the Guardian Algorithm, silently protecting customers from fraud.

# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.svm import OneClassSVM

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import classification\_report, confusion\_matrix

# Step 1: Simulating Normal and Fraudulent Transactions

np.random.seed(42)

normal\_transactions = np.random.normal(loc=50, scale=10, size=(200, 2)) # 200 normal transactions

fraudulent\_transactions = np.array([[150, 10], [200, 5], [180, 20], [170, 25]]) # 4 fraudulent transactions

# Combining normal and fraudulent transactions

transactions = np.vstack([normal\_transactions, fraudulent\_transactions])

labels = np.hstack([np.ones(200), -1 \* np.ones(4)]) # 1 for normal, -1 for fraud

# Step 2: Scaling the Data

scaler = StandardScaler()

transactions\_scaled = scaler.fit\_transform(transactions)

# Step 3: Training the One-Class SVM Model

model = OneClassSVM(kernel='rbf', gamma=0.1, nu=0.05)

model.fit(transactions\_scaled[:200]) # Train on normal transactions only

# Step 4: Making Predictions

predictions = model.predict(transactions\_scaled)

# Step 5: Evaluating the Model

print("Confusion Matrix:")

print(confusion\_matrix(labels, predictions))

print("\nClassification Report:")

print(classification\_report(labels, predictions))

# Step 6: Visualizing the Results

plt.figure(figsize=(10, 6))

plt.scatter(transactions\_scaled[:200, 0], transactions\_scaled[:200, 1], c='blue', label='Normal Transactions')

plt.scatter(transactions\_scaled[200:, 0], transactions\_scaled[200:, 1], c='red', label='Fraudulent Transactions')

# Plotting the decision boundary

xx, yy = np.meshgrid(np.linspace(-3, 3, 500), np.linspace(-3, 3, 500))

Z = model.decision\_function(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), 0, 7), cmap=plt.cm.coolwarm)

plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')

plt.title("Quantum Trust: Protecting Against Fraud with One-Class SVM")

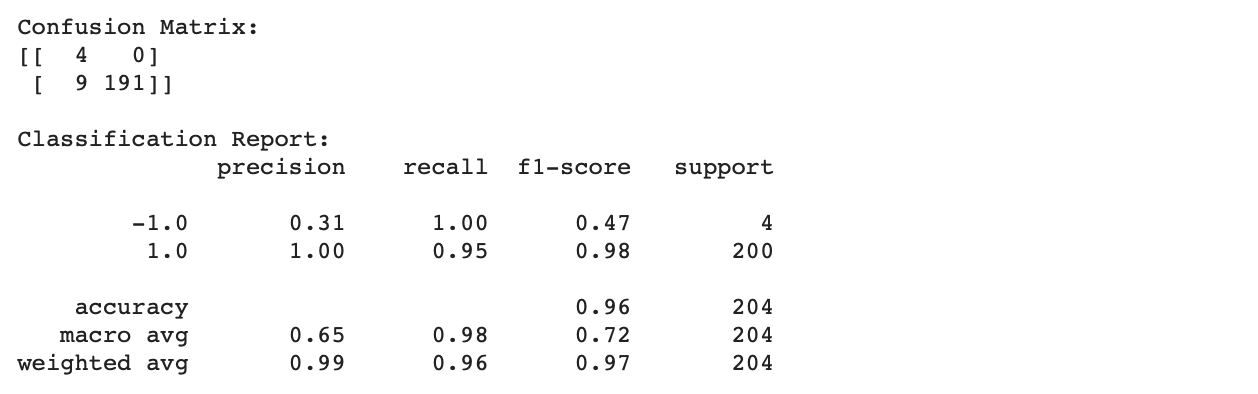
plt.xlabel("Feature 1 (e.g., Amount)")

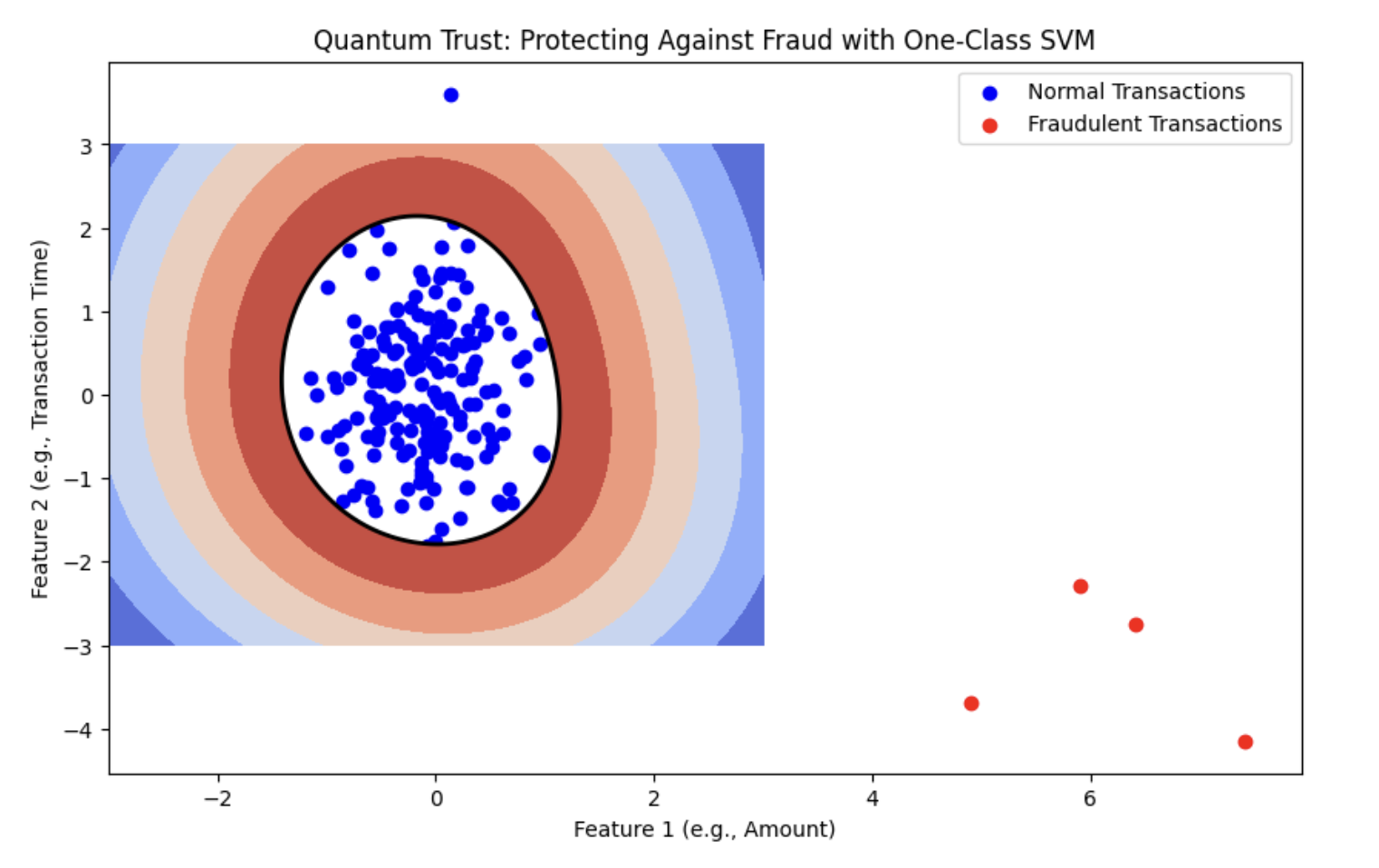
plt.ylabel("Feature 2 (e.g., Transaction Time)")

plt.legend()

plt.show()

**Output-**





**Explanation:**

The Guardian Algorithm uses a One-Class SVM to identify and prevent fraudulent transactions. It begins by creating a dataset with both normal and fraudulent transactions. The data is scaled and used to train the One-Class SVM model on typical, non-fraudulent transactions. The model then detects anomalies by flagging transactions that differ from the norm. Results are visualised to evaluate how well the model distinguishes between normal and fraudulent transactions. The model's accuracy and effectiveness are assessed using a confusion matrix and classification report, helping Quantum Trust Bank effectively combat fraud.

### **6. Advantages of One-Class SVM:**

1. **Effective in Anomaly Detection:** One-Class SVM is designed to handle cases where anomalies or rare events are of greater interest than the normal cases. It can effectively identify outliers or deviations from the norm.
2. **Works Well with Limited Anomaly Data:** It is particularly useful when there is a scarcity of data representing the anomalous class, which is often the case in fraud detection, network intrusion, or equipment failure.
3. **Versatility:** It can be applied across various domains due to its ability to model complex, high-dimensional data and identify anomalies that do not conform to the general pattern.
4. **Robustness to Imbalanced Data:** One-Class SVM does not require balanced data, which is advantageous when the number of normal samples greatly exceeds the number of anomalies.

### **7. Applications of One-Class SVM:**

1. **Fraud Detection:** Identifying fraudulent transactions by detecting deviations from typical spending patterns. For example, unusual transaction amounts or frequencies can be flagged as potential fraud.
2. **Network Security:** Detecting abnormal network traffic that may indicate a cyber attack or unauthorised access. Unusual patterns in data packets or login attempts can be detected using One-Class SVM.
3. **Industrial Maintenance:** Predicting equipment failures by identifying unusual patterns in sensor data. For instance, deviations in temperature or vibration readings could signal potential mechanical issues.
4. **Healthcare:** Detecting rare diseases or abnormal medical conditions by identifying patients whose medical data significantly deviates from normal patterns. This can help in early diagnosis of conditions that are not frequently encountered.

**Local Outlier Factor (LOF)**

**1. Introduction:**

Local Outlier Factor (LOF) is an optimal detection method that, unlike traditional methods, evaluates outliers based on local data rather than global statistics. This method is especially good for datasets with different spatial distributions because it takes into account the density of data points in the local environment.

**2. How Local Outlier Factor (LOF) Algorithm works**

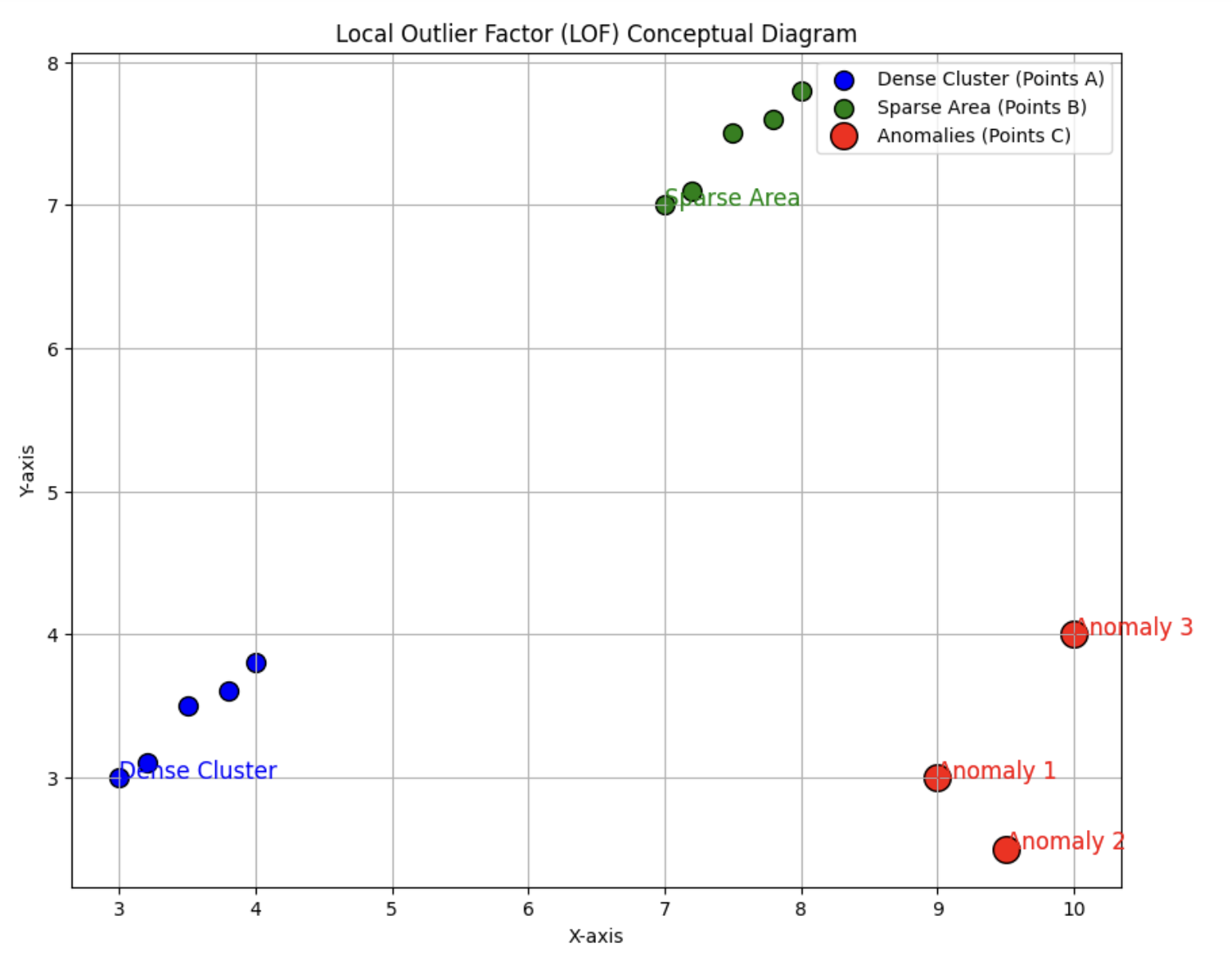
LOF operates on the principle that anomalies are data points that significantly differ from their neighbours. To understand LOF, we need to grasp the concept of local density.

* **Local Density:** It measures how densely packed the data points are in a local neighbourhood. A data point is considered an anomaly if its local density is significantly lower compared to its neighbours.

**3. LOF Algorithm Steps:**

Here’s a simplified explanation of how LOF identifies anomalies:

1. **Determine the k-nearest neighbours:** For each data point, find its k-nearest neighbours. The choice of k is crucial as it determines the size of the neighbourhood considered.
2. **Compute Local Reachability Density (LRD):**
   * For each data point pp, compute its reachability distance to its k-nearest neighbours.
   * The reachability distance is the maximum of the distance between pp and its neighbours or the distance between the neighbours themselves.
3. **Calculate Local Outlier Factor (LOF):**
   * For each data point pp, compute the average local reachability density ratio of pp and its neighbours.
   * LOF is defined as the ratio of the average local density of pp’s neighbours to pp’s own local density.

**4. Conceptual diagram illustrating LOF:**

The diagram illustrates the Local Outlier Factor (LOF) algorithm. It shows a dense cluster of blue points, indicating a high local density where points are closely packed and considered normal. In contrast, the green points in the sparse area are more spread out, representing lower local density. Red points, isolated from both the dense cluster and the sparse area, are identified as anomalies due to their significantly lower local density compared to their neighbours. The diagram highlights how outliers can appear in various regions of the data.

**4. Coding Example:**

Consider a manufacturing plant that uses sensors to monitor machinery performance. The sensors record data like temperature, vibration, and pressure. Occasionally, machinery may experience faults or abnormalities that cause unusual sensor readings. The company faces a challenge to identify these anomalies in real-time to prevent equipment failures and ensure smooth operations.

The maintenance team faces the task of monitoring machinery performance across multiple sensors. With thousands of sensor readings being collected every minute, detecting faults based on global thresholds alone becomes impractical. Instead, they use the Local Outlier Factor (LOF) algorithm to identify unusual sensor behaviour whose code is given below.

from sklearn.neighbors import LocalOutlierFactor

import numpy as np

import matplotlib.pyplot as plt

# Simulate sensor data for normal and faulty machinery

np.random.seed(0)

normal\_sensor\_data = np.random.normal(loc=50, scale=5, size=(200, 2)) # Normal operating conditions

faulty\_sensor\_data = np.random.normal(loc=100, scale=10, size=(20, 2)) # Faulty conditions

sensor\_data = np.concatenate([normal\_sensor\_data, faulty\_sensor\_data], axis=0)

# Apply LOF model

clf = LocalOutlierFactor(n\_neighbors=20)

outlier\_labels = clf.fit\_predict(sensor\_data)

# Plot results

plt.figure(figsize=(10, 6))

# Normal sensor data

plt.scatter(sensor\_data[outlier\_labels == 1][:, 0], sensor\_data[outlier\_labels == 1][:, 1], color='blue', label='Normal Sensor Data', edgecolor='black')

# Faulty sensor data

plt.scatter(sensor\_data[outlier\_labels == -1][:, 0], sensor\_data[outlier\_labels == -1][:, 1], color='red', label='Faulty Sensor Data', edgecolor='black')

plt.xlabel('Sensor Reading 1')

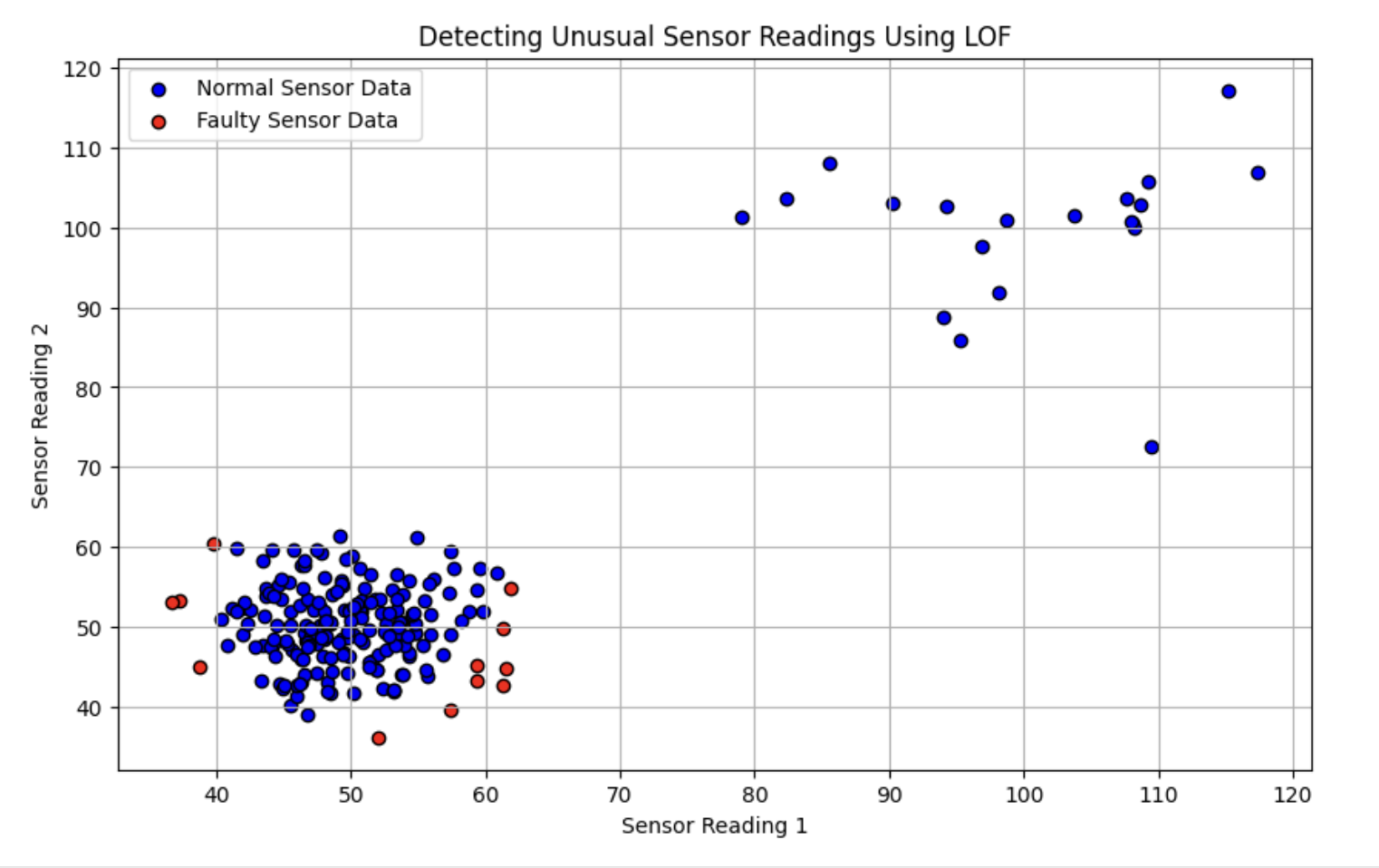
plt.ylabel('Sensor Reading 2')

plt.title('Detecting Unusual Sensor Readings Using LOF')

plt.legend()

plt.grid(True)

plt.show()



### **Explanation:**

* **Data Generation**: Simulates sensor data with normal readings and faulty conditions.
* **LOF Model**: Applies LOF to detect anomalies by analysing local density in sensor readings.
* **Visualisation**: Displays normal sensor readings in blue and faulty readings in red.

**Outcome**:

Using LOF, manufacturers can detect anomalous readings that differ from operational standards, allowing for timely maintenance and reducing the risk of equipment failure. This approach improves the ability to detect errors that would not be possible using the general threshold alone.

**Comparison of Anomaly Detection Methods**

| **Factor** | **Isolation Forest** | **One-Class SVM** | **Local Outlier Factor (LOF)** |
| --- | --- | --- | --- |
| Algorithm Complexity | Training: O(T \* N \* log N) | Training: O(N^2) to O(N^3) | Training: O(N^2) in the worst case |
| Prediction: O(T \* log N) | Prediction: O(N) | Prediction: O(N \* log N) |
| Scalability | Highly scalable for large and high-dimensional data | Less scalable with large or high-dimensional data | Can struggle with very large datasets |
| Handling High-Dimensional Data | Relatively good with high-dimensional data | Performance degrades with high-dimensional data | Difficulties in distance calculations in high dimensions |
| Parameter Tuning | Tuning: Number of trees, contamination parameter | Tuning: nu, gamma, kernel type | Tuning: Number of neighbours (n\_neighbors) |
| Use Cases | Large datasets, high-dimensional data, rare anomalies | Smaller datasets, well-defined data, single class | Datasets with varying densities, contextually different anomalies |
| Robustness to Noise | Generally robust to noise | Sensitive to noise and overlaps | Can be sensitive to noisy data |
| Interpretability | Straightforward: Tree depth and isolation | Less interpretable: Complex decision boundaries | Intuitive: Local density and distance metrics |

### **Key Points :**

**1. Isolation Forest**:

* + **Efficiency**: Highly scalable and efficient for large datasets and high-dimensional spaces due to its tree-based structure.
  + **Performance**: Works well when anomalies are sparse and isolated. Less sensitive to parameter tuning and noise.
  + **Use Cases**: Ideal for scenarios such as fraud detection, network intrusion detection, and any application dealing with large volumes of data where anomalies are rare and distinct.

2. **One-Class SVM**:

* **Scalability**: Can be less scalable due to its reliance on quadratic programming, which can be computationally intensive.
* **Performance**: Best suited for smaller, well-defined datasets where normal data is expected to be distinct from outliers. Sensitive to parameter settings and noise.
* **Use Cases**: Suitable for applications where the data is expected to belong to a single class, such as in industrial defect detection and certain quality control scenarios.

**Local Outlier Factor (LOF)**:

* **Handling Density**: Excels in datasets with varying densities,detecting anomalies based on local density deviations.
* **Performance**: Can be less effective with very large datasets or noisy data due to its reliance on distance calculations. Requires careful choice of parameters like the number of neighbors.
* **Use Cases**: Useful in scenarios with irregular patterns, such as environmental monitoring or sensor data analysis where local density variations are crucial.

### **When to Use Each Method ?**

* **Isolation Forest**: Choose this method when dealing with large-scale datasets and high-dimensional data where you expect anomalies to be rare and isolated. Its efficiency and robustness make it a go-to choice for many practical anomaly detection problems.
* **One-Class SVM**: Opt for this approach if you have a well-defined dataset with a clear distinction between normal and abnormal observations. It is particularly effective when you can afford the computational cost and need a model that focuses on separating the normal class from outliers.
* **Local Outlier Factor (LOF)**: Use LOF when your dataset exhibits varying local densities and you need to detect anomalies based on these variations. It is ideal for datasets where anomalies are contextually different from their neighbours and where local context is critical for accurate detection.

By understanding the strengths and limitations of each method, you can select the most appropriate anomaly detection technique based on your specific needs, dataset characteristics, and computational resources.

### 