Anomaly & Outlier Detection

Agenda

1. Introduction to Anomalies and Outliers
    - Anomalies
    - Outliers
    - Distinction Between Outliers and Anomalies
2. Challenges in Anomaly Detection
3. Types of Anomalies
    - Point Anomalies
    - Contextual Anomalies
4. Categories of Anomaly detection techniques
    - Supervised Anomaly Detection
        - Z-Score
        - Interquartile Range
        - Support Vector Machine
        - K-Nearest Neighbors
    - Unsupervised Anomaly Detection
        - One-Class Support Vector Machine
    - Semi-Supervised Detection
        - isolation Forest Algorithm
        - Local Outlier Factor
        - DBSCAN
5. Evaluation Metrics for Anomaly Detection

Introduction to Anomalies and Outliers

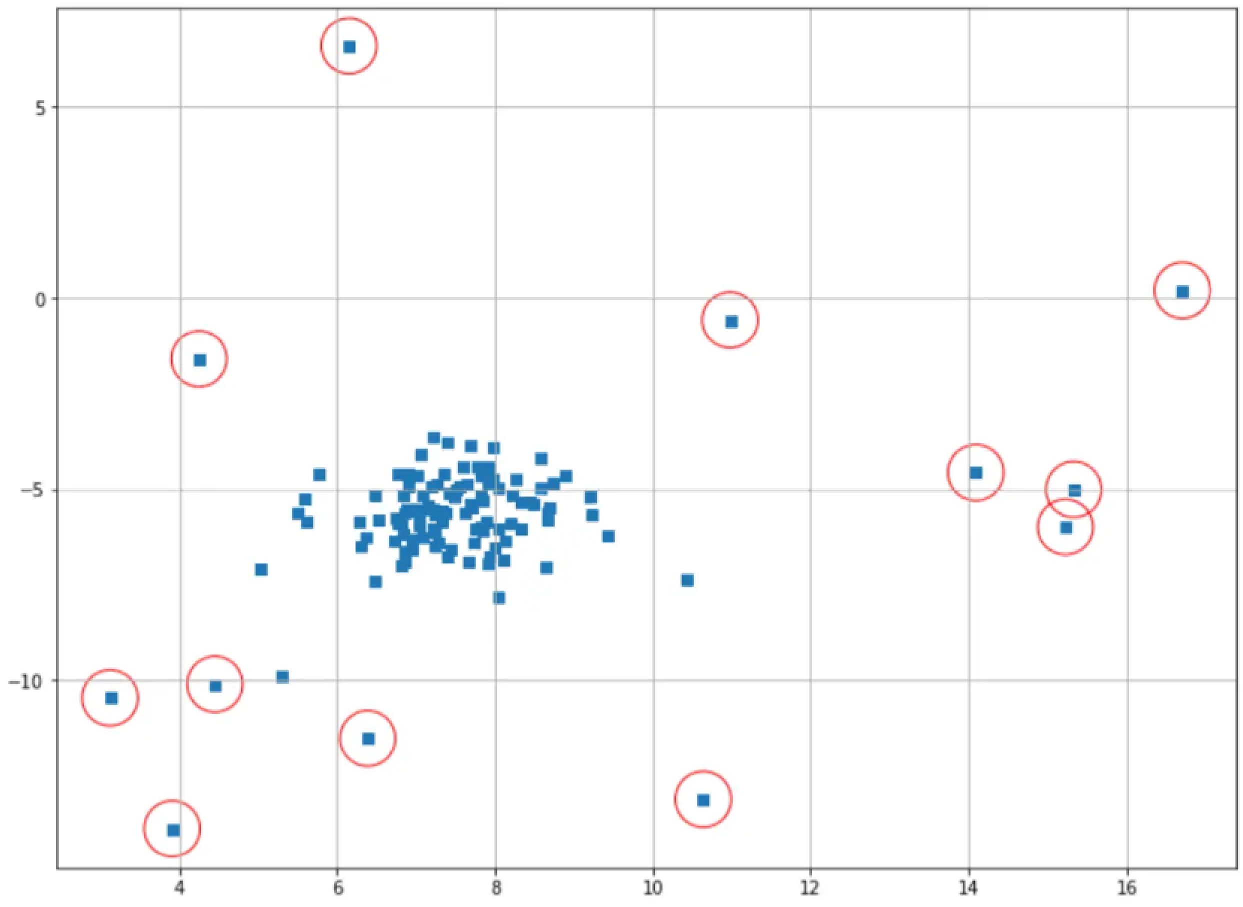
Anomalies

- Anomalies, also known as outliers or exceptions, are data points that significantly deviate from the majority of the data within a given dataset.
- These points are often identified as unusual observations that do not conform to the expected patterns or behaviors established by the bulk of the data.

- Anomalies can arise from various factors, including variability in measurement, errors in data collection, or genuine changes in the underlying processes that generate the data.

Causes of Anomalies:

- Measurement Errors: Sometimes anomalies occur due to inaccuracies in data collection methods. For example, a faulty sensor might record an implausibly high temperature reading.
- Natural Variability: In some cases, anomalies can reflect natural variations or rare events in the underlying data-generating process. For example, a sudden spike in sales during a holiday season might appear anomalous when viewed in the context of historical sales data.
- Changes in Data-Generating Processes: Anomalies may signal changes in the system being studied. For example, a significant drop in stock prices may indicate a market shift or an underlying issue with the company, rather than just a typical fluctuation.



Importance of Anomaly Detection

- Detecting anomalies is a critical task in various fields, including finance, healthcare, manufacturing, and cybersecurity, as it can lead to:
- Risk Management: In finance, identifying unusual transactions can help detect fraud, while in manufacturing, anomalies may indicate equipment malfunctions or quality control issues.

- Data Integrity: Recognizing measurement errors helps maintain the quality and reliability of data for analysis and decision-making.
- Predictive Insights: Understanding the reasons behind anomalies can provide insights into shifts in trends or patterns, enabling organizations to adapt strategies or respond proactively to changing conditions.

Example of Anomalies

- To illustrate the concept of anomalies, consider a dataset containing daily temperature readings for a specific region over a winter season. If the recorded temperatures generally range from -5°C to 10°C, a temperature reading of 40°C during this period would stand out as an anomaly.
- In this scenario:
    - Contextual Significance: The 40°C reading does not align with the expected winter temperatures, indicating it is far outside the normal range for that time of year.
    - Implications: Such an anomaly could suggest several possibilities, such as a malfunctioning thermometer, data entry error, or, in rare cases, an actual unusual weather event (like an unseasonably warm day in a typically cold winter).
    - Action: Identifying this anomaly would prompt further investigation to determine the cause, which could involve checking the data source for errors, consulting meteorological records, or analyzing the data collection methods employed.

Outliers

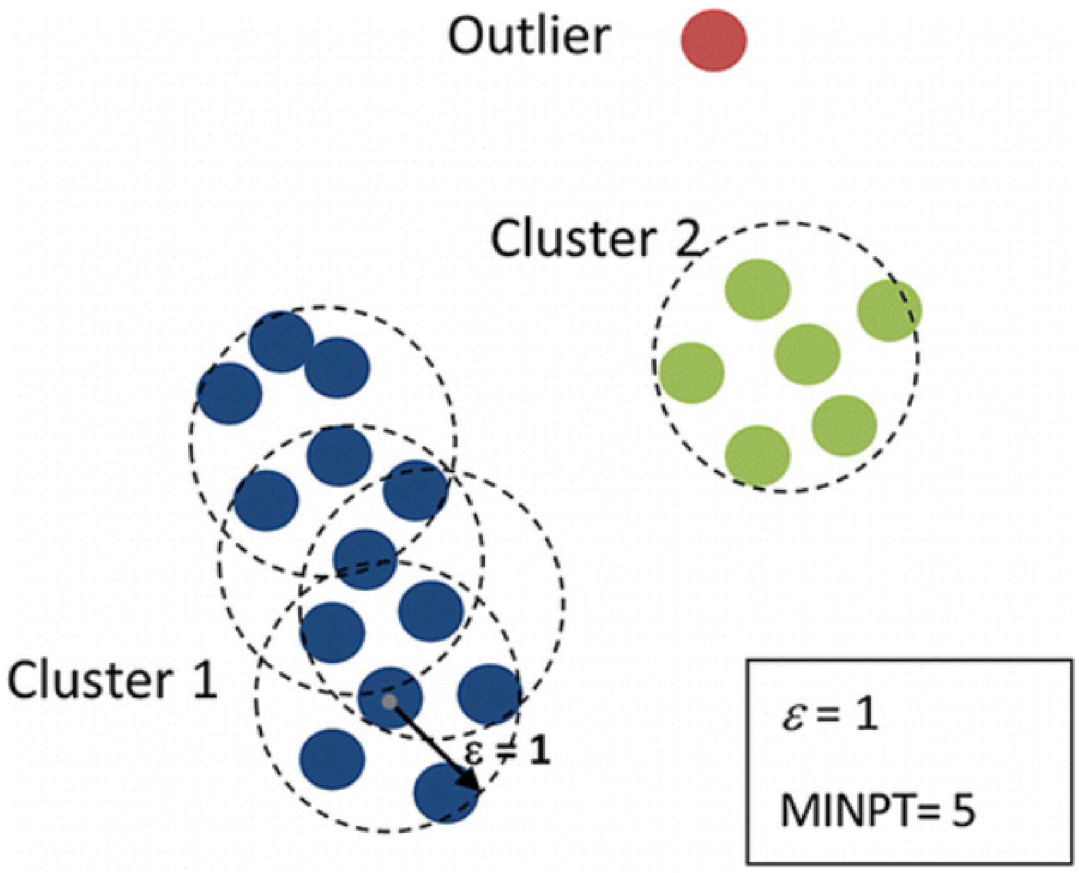
- Outliers are extreme values that significantly differ from the rest of the data in a dataset.
- These values lie outside the expected range or pattern of observations and can result from various factors, including natural variations, measurement errors, or data entry mistakes.
- While all anomalies can be classified as outliers due to their deviation from the norm, not every outlier qualifies as an anomaly.
- This distinction is crucial in data analysis, as some outliers may represent legitimate variations rather than errors or unusual occurrences.

Importance of Identifying Outliers

- Identifying outliers is essential for several reasons:
    - Data Quality and Integrity: Outliers may indicate errors or issues with data collection methods. Identifying and addressing these outliers helps ensure the integrity of the dataset.
    - Statistical Accuracy: In statistical analyses, outliers can heavily influence measures of central tendency (mean, median) and variability (standard deviation).

Understanding and addressing outliers can lead to more accurate statistical interpretations.

- Insight Generation: Outliers can provide valuable insights into unusual behaviors or trends within the data. In business contexts, for example, a sudden spike in sales might warrant further investigation to understand underlying causes or emerging opportunities.
- Model Performance: In predictive modeling, outliers can adversely affect model performance, leading to overfitting or misrepresentations of relationships between variables. Addressing outliers can improve model accuracy and robustness.



Example of Outliers

- To illustrate the concept of outliers, consider a dataset that represents salaries within a company. If the average salary is $50,000 with a standard deviation of $10,000, a salary of $1,000,000 would clearly stand out as an outlier.
- Contextual Analysis: In this scenario, the $1,000,000 salary might represent:
  - A senior executive's compensation package that includes bonuses or stock options, thereby reflecting legitimate variations within the salary distribution.
  - An error in data entry, where an extra zero was inadvertently added, leading to a distorted salary figure.

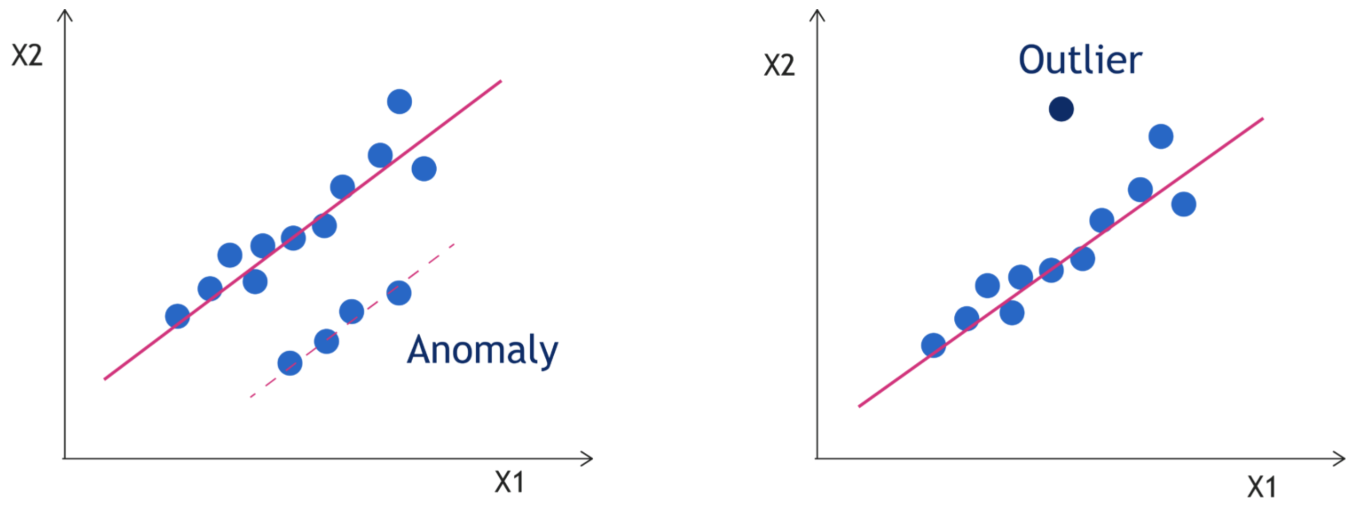
- Implications for Data Analysis: The presence of this outlier can significantly affect the average salary calculation, skewing it upward and potentially leading to incorrect conclusions about overall salary distribution within the company.

Distinction Between Outliers and Anomalies

1. Nature and Context:
   - Outliers: Outliers may be valid observations in a dataset. For instance, in a dataset of house prices, a mansion that costs significantly more than other houses in the same region could be an outlier but still be legitimate data. Outliers are not always indicative of a problem; they might represent rare but valid occurrences.
   - Anomalies: Anomalies, on the other hand, suggest something unusual that requires deeper investigation. For example, in a dataset of network traffic, a sudden surge of traffic at an unexpected time may be an anomaly indicating potential malicious activity or system failure. Anomalies are often seen as "exceptions" or "red flags."
2. Impact on Data:
   - Outliers: These can distort statistical summaries, such as the mean, standard deviation, and correlation coefficients. If not handled properly, outliers can skew the results of data analysis and lead to inaccurate conclusions. However, they do not necessarily imply an error in the data; they might just reflect unusual but valid cases.
   - Anomalies: Anomalies typically indicate that something unusual or important has occurred in the system being monitored. They may signify errors, fraud, or changes in the underlying process. Anomalies usually require further analysis to determine whether they represent an issue or a critical insight.
3. Example:
   - Outlier Example: In a dataset of employee salaries, if most salaries range between 40.000 Dollar and 60.000 Dollar, but one salary is $1,000,000, that is an outlier. The high salary might belong to a top executive, so it’s a legitimate but rare occurrence.
   - Anomaly Example: In the same salary dataset, if an employee's salary suddenly drops to 10,000 Dollar from 50,000 Dollar without explanation, this might be an anomaly. This drastic change could signal a data entry error or something unusual in the company’s salary structure.
4. Actions and Handling:
   - Outliers: When dealing with outliers, the analyst must decide whether to keep them, transform them, or remove them, depending on their relevance. Sometimes, outliers are retained if they are legitimate data points that provide valuable insights into the variability of the system.

- Anomalies: Anomalies typically warrant further investigation. They may be signs of system errors, fraud, or critical system changes. The goal in anomaly detection is to identify these unusual occurrences and explore their causes and potential consequences.

5. Relationship:
    - While all anomalies are outliers because they deviate from the norm, not all outliers are anomalies. Some outliers are simply rare but legitimate data points, while anomalies often suggest something unusual that might require attention or further analysis.



Challenges in Anomaly Detection

1. Imbalanced Data:
    - Rarity of Anomalies: By nature, anomalies are rare occurrences, which leads to highly imbalanced datasets where normal data vastly outnumber anomalies. Traditional machine learning algorithms struggle with imbalanced data because they tend to focus on optimizing accuracy for the majority class (normal data), ignoring the minority class (anomalies).
    - Lack of Anomaly Labels: In many cases, labeled anomalies are rare or unavailable. This makes supervised learning approaches challenging to implement. Without sufficient labeled data, the task of detecting anomalies relies heavily on unsupervised or semi-supervised methods, which may not perform as well as supervised approaches.
2. High Dimensionality:
    - Curse of Dimensionality: In high-dimensional datasets, the distance between data points tends to become similar, making it difficult to distinguish normal data from anomalies. This is known as the "curse of dimensionality." As the number of

dimensions increases, anomaly detection algorithms struggle to find meaningful patterns.
- Feature Selection: Selecting the right features for anomaly detection is challenging in high-dimensional data. Some features may carry useful information for identifying anomalies, while others may introduce noise and degrade the performance of detection algorithms. Dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) are often required, but these techniques can sometimes obscure important details about anomalies.

3. Data Noise and Variability:
    - Noise vs. Anomalies: Real-world data is often noisy due to sensor errors, transmission issues, or inconsistencies in data collection methods. Distinguishing between legitimate anomalies and noise is challenging. Noise can be mistaken for anomalies, leading to false positives, while actual anomalies might be overlooked because they are hidden within noisy data.
    - Variability in Normal Behavior: In some systems, normal behavior can vary widely over time, making it difficult to distinguish between normal variability and true anomalies. For example, network traffic or financial transactions may exhibit periodic bursts of activity, which complicates the task of detecting real anomalies.
4. Evolving Anomalies:
    - Concept Drift: In dynamic systems, the characteristics of normal and anomalous data can change over time due to a phenomenon known as "concept drift." This drift makes it challenging to use static models for anomaly detection because models trained on historical data may become outdated and fail to detect new types of anomalies. Adapting anomaly detection models to handle such evolving behavior is an ongoing research challenge.
    - Contextual Changes: The behavior of a system can change depending on the context. For example, seasonal variations, sudden market shifts, or changes in system configuration may cause data patterns to shift, making it difficult to detect anomalies using models based on static assumptions.
5. Algorithm Selection:
    - Choosing the Right Model: No single anomaly detection algorithm works best for all types of data and all contexts. Different algorithms have different strengths and weaknesses, depending on factors like data distribution, noise levels, and dimensionality. Selecting the most appropriate algorithm for a specific application often involves experimentation and tuning of parameters.
    - Algorithm Complexity: Some anomaly detection algorithms, such as deep learning models, require substantial computational resources and expertise to implement, making them unsuitable for smaller datasets or real-time applications. Balancing

algorithm complexity with available resources and the nature of the task is a challenge.

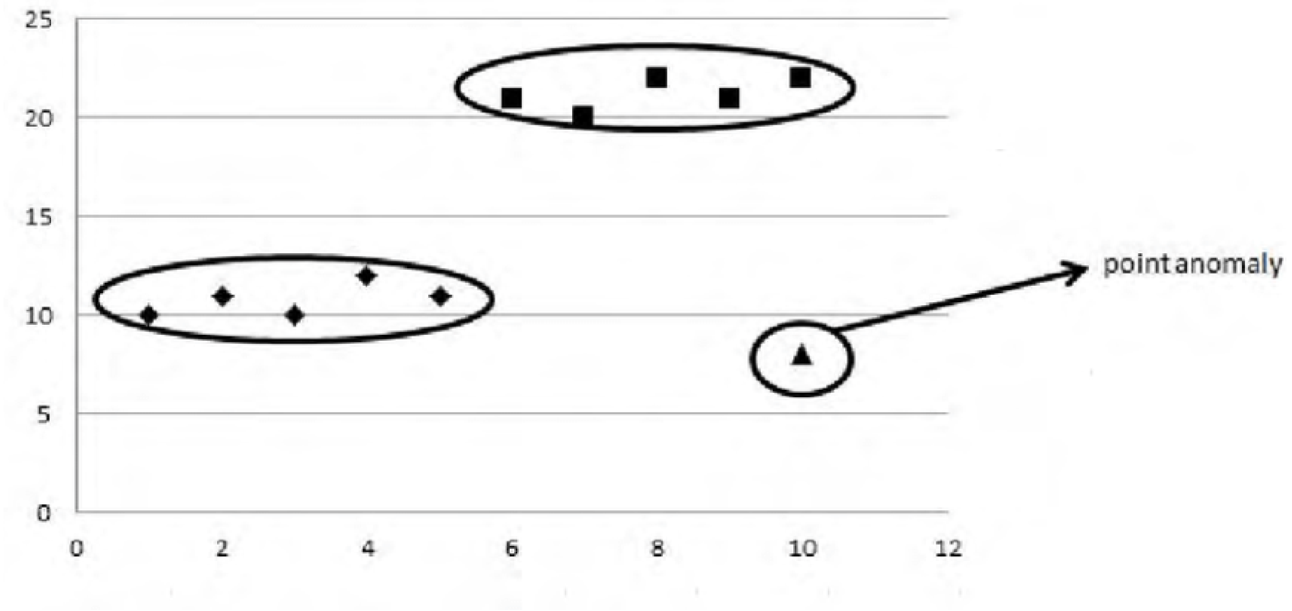
Types of Anomalies

- Anomalies, also known as outliers, are data points that deviate significantly from the expected behavior or distribution of data. - Anomaly detection is crucial in fields like fraud detection, network security, industrial monitoring, and healthcare, where unusual events can have significant consequences.
- Understanding the types of anomalies is essential for selecting the appropriate methods for detecting them.
- There are three primary types of anomalies: Point anomalies, Contextual anomalies, and Collective anomalies. Below is a detailed exploration of these types.

Point Anomalies (Global Anomalies)

- A point anomaly occurs when a single data point significantly deviates from the rest of the dataset.
- These anomalies are detected based on their deviation from the general distribution or expected range of data points.
- Point anomalies are the most common type and can be found in various datasets, including time-series, transactional, and spatial data.
- Characteristics:
  - Independence: The anomaly is isolated and does not depend on other data points or context.
  - Statistical Deviation: Often identified by applying statistical measures (e.g., mean, variance) to detect data points that are significantly far from the majority of the data.
- Applications:
  - Fraud Detection: Unusually large transactions compared to a user's typical transaction size.
  - Industrial Monitoring: Sensor readings that are drastically different from the normal range.
- Detection Techniques:
  - Z-score: Measures how far a point deviates from the mean in terms of standard deviations.
  - Isolation Forests: Identifies points that are easier to isolate as outliers in a data distribution.

- Density-based Methods: Methods like DBSCAN (Density-Based Spatial Clustering of Applications with Noise) help detect anomalies by identifying regions of low data density.



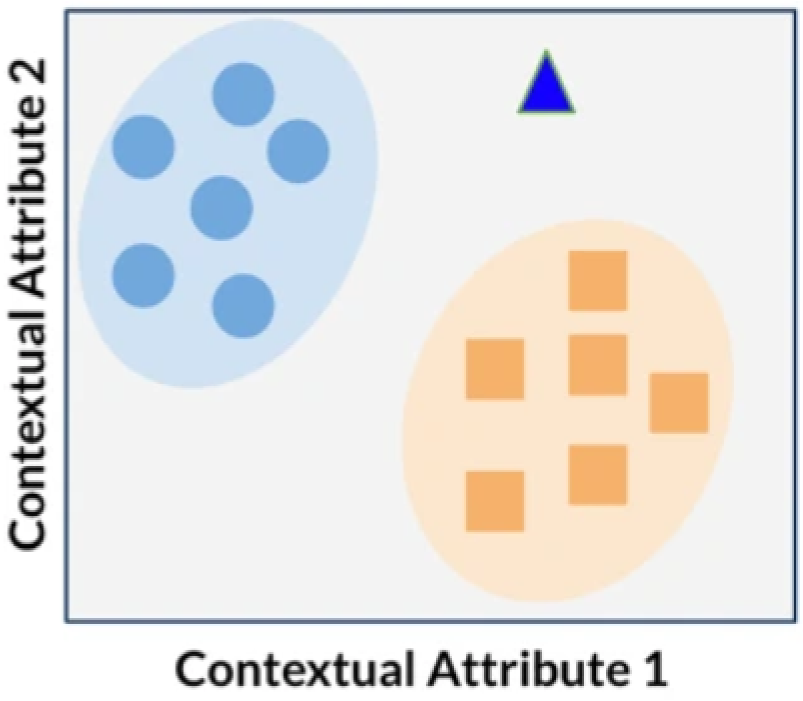
- Example:
  - In a dataset containing temperatures of a city over a month, if the temperatures typically range between 15°C and 25°C and one day records a temperature of 40°C, this would be considered a point anomaly.

Contextual Anomalies (Conditional Anomalies)

- A contextual anomaly occurs when a data point is anomalous in a specific context but might appear normal in another.
- The anomaly's detection depends not only on the data point itself but also on the surrounding context.
- This type of anomaly is prevalent in time-series data and spatiotemporal datasets, where the temporal or spatial context significantly influences what is considered normal.
- Characteristics:
  - Contextual Dependency: The data point is anomalous only when viewed within its context, such as time, location, or other features.
  - Relative Abnormality: The point may be normal under different conditions but is considered an anomaly in a specific context.
- Applications:
  - Time-Series Data: Detecting unusual spikes or drops in network traffic or stock prices over time.

- Geospatial Data: Identifying unusual events in certain locations (e.g., sudden changes in air quality in specific regions).

- Detection Techniques:
  - Seasonal Decomposition of Time Series (STL): Decomposes time series into trend, seasonal, and residual components to detect contextual anomalies.
  - Recurrent Neural Networks (RNNs): In the context of time-series data, RNNs are used to model sequential dependencies and detect contextual anomalies based on temporal patterns.
  - Sliding Windows: A technique used to detect anomalies by observing a rolling window of data points over time, comparing the current data point to its context.

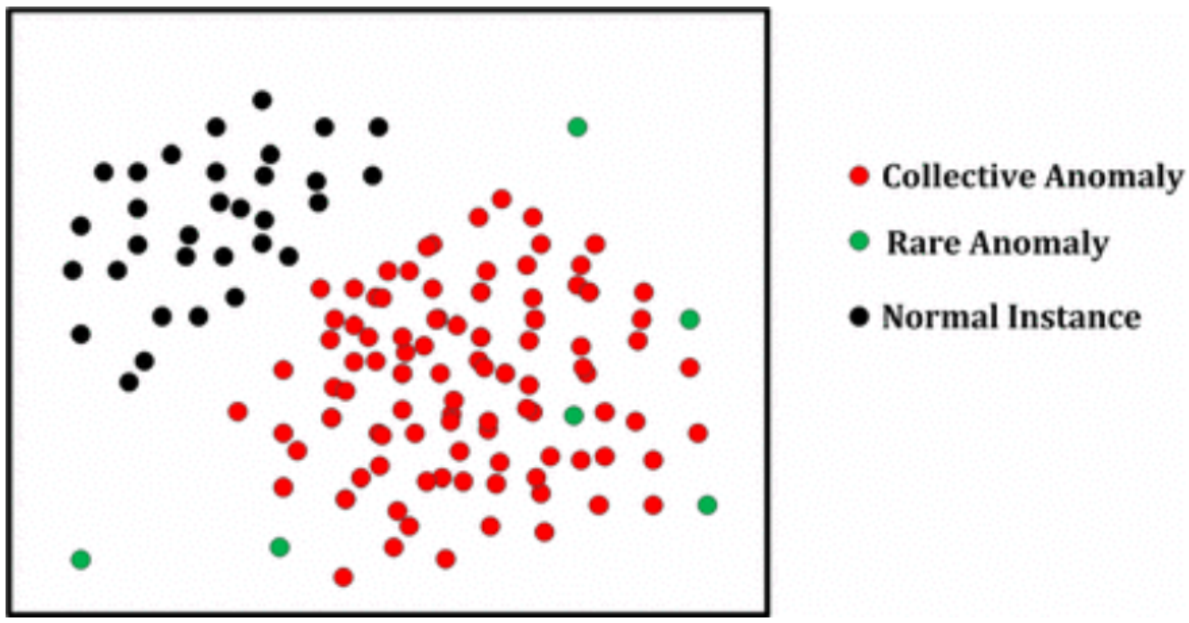


- Example:
  - A temperature reading of 30°C might be considered normal in the summer but anomalous in the winter. Similarly, a high sales figure might be normal during a holiday season but abnormal on a regular day.

Collective Anomalies

- A collective anomaly refers to a situation where a group of related data points together represents an anomaly, even though individual points in the group might not be anomalous in isolation.
- This type of anomaly typically occurs when patterns, sequences, or clusters of data points form unusual behavior.

- Collective anomalies are more complex and often more difficult to detect than point or contextual anomalies.
- Characteristics:
  - Group Behavior: The anomaly is detected only when a group of data points is considered together.
  - Pattern-Based Anomalies: The anomalous behavior is related to the overall structure, pattern, or sequence of the data points.
- Applications:
  - Network Security: Detecting patterns of low-level but suspicious network activity that, when grouped, indicate an attack.
  - Healthcare: Monitoring sequences of symptoms or test results in a patient over time that may indicate an abnormal health condition.
  - Time-Series Analysis: Identifying patterns in financial data where multiple small transactions, collectively, signal fraudulent activity.
- Detection Techniques:
  - Clustering Algorithms: Algorithms like DBSCAN or k-Means can help identify collective anomalies by detecting unusual clusters or groups of points.
  - Markov Models: Can be used to detect anomalies in sequential data where the likelihood of a particular sequence deviating from the expected model is low.
  - Sequence-to-Sequence (Seq2Seq) Models: These models, commonly used in time-series data, can help predict future sequences and flag sequences that significantly deviate from normal behavior.

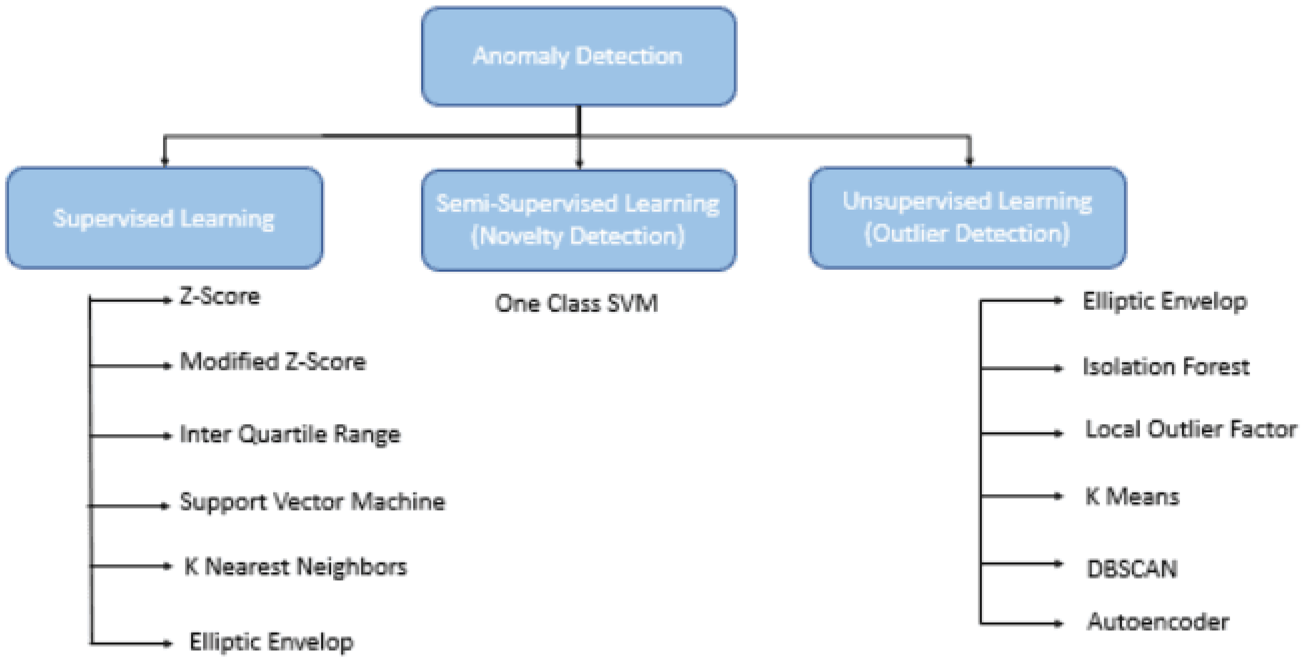


- Example:

- In a web traffic dataset, a series of small but steadily increasing visits to a website might not be anomalous individually. However, if viewed as a group, this could indicate a potential Distributed Denial of Service (DDoS) attack, representing a collective anomaly.

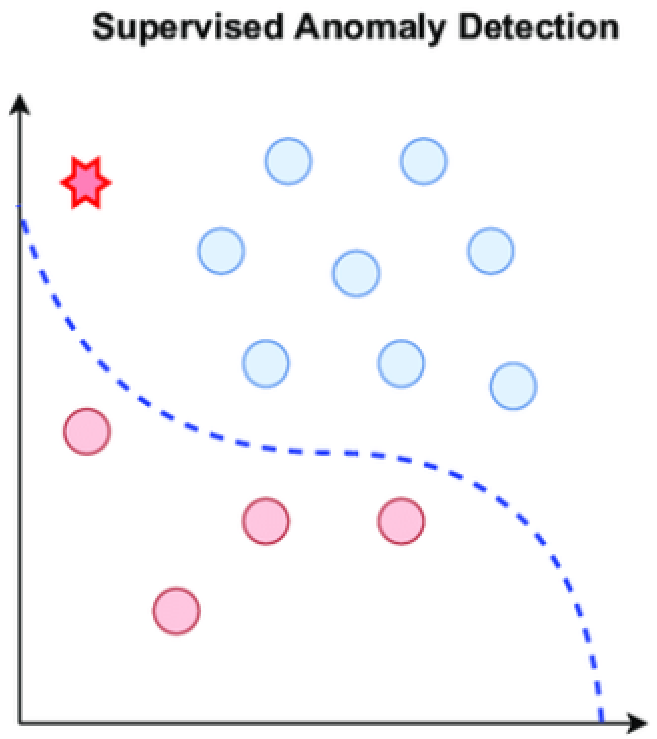
Anomaly Detection Techniques

- Anomaly detection is the task of identifying data points that significantly deviate from the normal pattern.
- These anomalies, also known as outliers, can indicate critical events, rare occurrences, or even malicious activities.
- Based on the availability of labeled data, anomaly detection techniques are divided into three categories: Supervised Anomaly Detection, Unsupervised Anomaly Detection, and Semi-Supervised Anomaly Detection.



Supervised Anomaly Detection

- Supervised anomaly detection relies on training data that is labeled as either normal or anomalous.
- Similar to traditional classification problems, a model is trained on this labeled dataset to learn the distinguishing features between normal and anomalous instances.
- Once trained, the model can be used to classify new data as normal or anomalous.



Key Concepts:

- Labeled Data: In supervised learning, both normal and anomalous instances are clearly marked in the training data.
- Classification: The model essentially performs a classification task where the goal is to predict if a data point belongs to the normal class or the anomaly class.
- Accuracy & Generalization: If the labeled dataset is comprehensive, supervised techniques can provide highly accurate and reliable results.
- Characteristics:
  - Dependence on labeled data: Supervised methods are effective only when a well-labeled dataset is available, which contains enough anomalous examples.
  - Class Imbalance: Anomalies are rare in most real-world datasets, leading to a severe class imbalance problem, which makes training the model challenging.
  - High performance on known anomalies: These models can efficiently detect known types of anomalies if adequately trained.
- Challenges:
  - Limited Anomaly Examples: Anomalies are rare, and acquiring enough labeled examples for training can be difficult.
  - Costly Labeling: Labeling data, especially anomalies, is often expensive and time-consuming, particularly in domains like cybersecurity or healthcare.
  - Overfitting: If the labeled dataset contains too few anomalous instances, the model may fail to generalize and overfit the data, leading to poor performance on unseen data.
- Example:

    - Fraud Detection in Banking: A supervised model is trained on past credit card transaction data, where each transaction is labeled as either “fraud” or “non-fraud.” The model learns to predict whether future transactions are fraudulent.

Z-Score

- The Z-score (or standard score) is a statistical measure that quantifies how many standard deviations a data point is from the mean of the dataset. It is widely used in anomaly detection to identify data points that are significantly different from the majority of the data.
- In the context of supervised anomaly detection, the Z-score can be used to transform features and detect outliers or anomalous data points based on their distance from the mean. Anomalies often have extreme Z-scores, meaning they are far from the typical range of values.
- Formula for Z-Score: The Z-score for a data point x_i is calculated as:

$Z_i = (x_i - \mu) / \sigma$

Where:

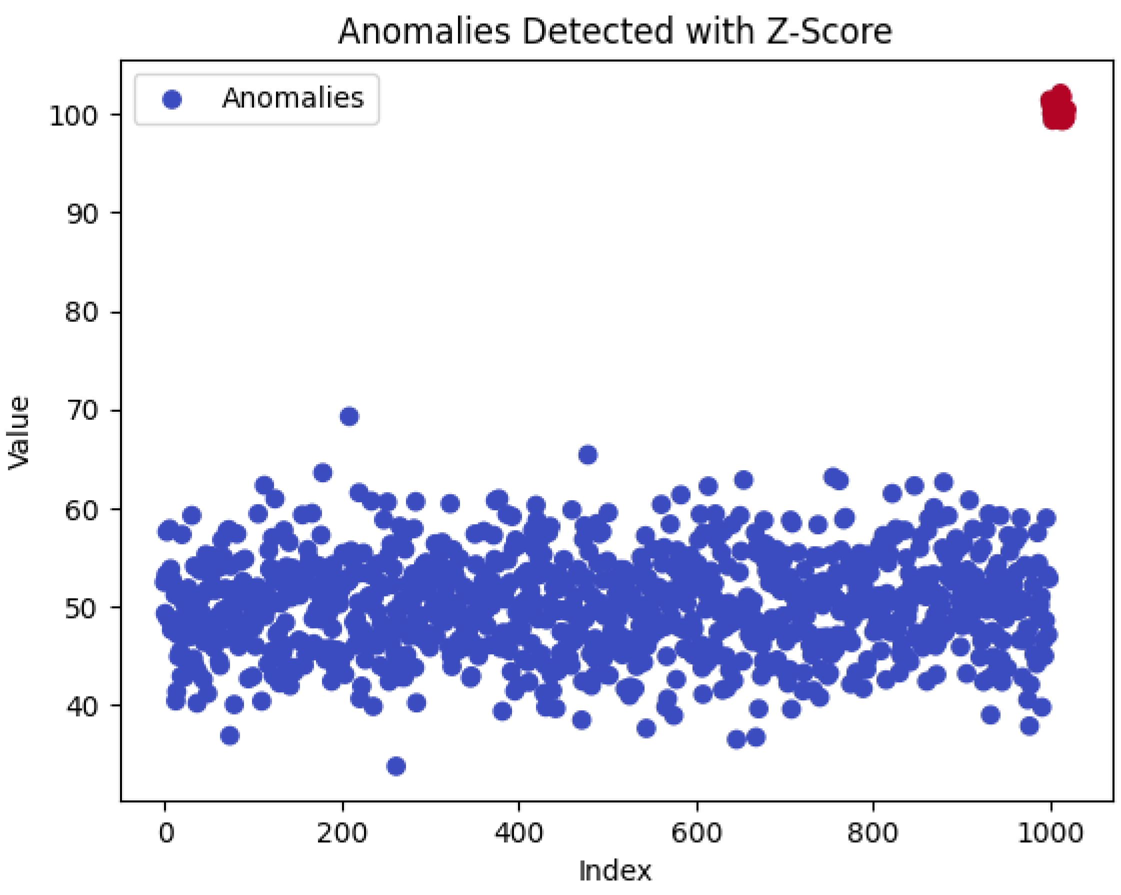
- x_i is the value of the data point.
- μ is the mean of the dataset.
- σ is the standard deviation of the dataset.
- Z-Score and Anomalies:
    - Normal Data: Data points with Z-scores close to 0 are considered normal because they fall near the mean.
    - Anomalous Data: Data points with Z-scores that are much higher or lower than the average (typically beyond ±3) are considered anomalies. These extreme values are much farther from the mean, indicating they differ significantly from the rest of the data.
- Using Z-Score in Supervised Anomaly Detection:
    - Feature Transformation: Z-scores can be used to standardize the features of the dataset. This helps in transforming the data to a common scale, ensuring that features with different units or ranges are normalized before applying a supervised anomaly detection model.
    - Threshold-Based Detection: Once the Z-scores are computed for each data point, a threshold can be set (e.g., ±3) to label data points as normal or anomalous based on their Z-scores. Points beyond the threshold are flagged as anomalies, while points within the threshold are labeled as normal.
    - Integration with Supervised Models:

- Data Preprocessing: In supervised anomaly detection, Z-score normalization is often part of the preprocessing stage, where it helps in preparing the data for machine learning algorithms. The normalized data ensures that features contribute equally during model training.
- Labeling Anomalies: Z-score can be used to generate labels for training datasets. If a data point’s Z-score exceeds a predefined threshold, it can be labeled as an anomaly for use in supervised learning models (e.g., classification models like SVM, random forest, etc.).

Demonstrating how to use Z-score for anomaly detection with a supervised approach. We'll use scikit-learn and numpy to preprocess the data and detect anomalies using Z-scores.

```
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Step 1: Simulate a dataset
np.random.seed(42)

# Create normal data points (mean=50, std=5) and some anomalies (outliers)
normal_data = np.random.normal(50, 5, 1000)   # 1000 normal points
anomalies = np.random.normal(100, 1, 20)      # 20 anomalous points

# Combine the datasets
data = np.concatenate([normal_data, anomalies])

# Create a dataframe
df = pd.DataFrame(data, columns=["value"])

# Step 2: Calculate Z-scores
df['z_score'] = stats.zscore(df['value'])

# Step 3: Detect anomalies using Z-score (set threshold at |Z| > 3)
df['anomaly'] = df['z_score'].apply(lambda x: 1 if np.abs(x) > 3 else 0)

# Split the data into normal and anomalous
X = df[['value']]  # Features
y = df['anomaly']  # Labels (0: normal, 1: anomaly)

# Step 4: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

# Step 5: Train a simple logistic regression model to classify anomalies
model = LogisticRegression()
model.fit(X_train, y_train)

# Step 6: Predict on the test data
y_pred = model.predict(X_test)

# Step 7: Evaluation - Confusion matrix and classification report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Visualize the anomalies
import matplotlib.pyplot as plt
plt.scatter(df.index, df['value'], c=df['anomaly'], cmap='coolwarm', label='Anoma
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Anomalies Detected with Z-Score')
plt.legend()
plt.show()
```

```
[[300   0]
 [  0   6]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       300
           1       1.00      1.00      1.00         6

    accuracy                           1.00       306
   macro avg       1.00      1.00      1.00       306
weighted avg       1.00      1.00      1.00       306
```



Interquartile Range (IQR)

- The Interquartile Range (IQR) is a simple yet effective method for detecting anomalies or outliers in data.
- It is based on dividing the dataset into quartiles and identifying the range within which the central 50% of the data lies.
- This method is particularly useful for datasets that do not follow a normal distribution and can handle skewed data.
- It is also resistant to the influence of extreme values, making it robust for many real-world applications.

What is IQR?

- The Interquartile Range (IQR) is the difference between the third quartile (Q3) and the first quartile (Q1) of a dataset:

```
IQR=Q3−Q1
```

- Q1 (First Quartile): The 25th percentile, meaning 25% of the data is below this value.
- Q3 (Third Quartile): The 75th percentile, meaning 75% of the data is below this value.
- IQR: The middle 50% of the data, where most of the values lie.

Anomaly Detection Using IQR

- In anomaly detection, we can define a data point as an outlier if it lies significantly outside the interquartile range. To identify outliers, we calculate thresholds called fences:
  - Lower Fence:

    ```
    Lower Fence = Q1 − 1.5 × IQR
    ```

    Any data point below this value is considered an outlier.
  - Upper Fence:

    ```
    Upper Fence = Q3 + 1.5 × IQR
    ```

    Any data point above this value is considered an outlier.

The factor of 1.5 is a rule of thumb that balances between flagging outliers and avoiding too many false positives. However, in some extreme cases, you can increase this factor (e.g., 3) to detect more extreme anomalies.

Code Example for IQR-Based Anomaly Detection:

```
import numpy as np
import pandas as pd

# Step 1: Create a dataset
data = np.array([10, 12, 14, 15, 16, 18, 19, 21, 22, 24, 30, 45])

# Step 2: Convert to a Pandas DataFrame
df = pd.DataFrame(data, columns=['value'])

# Step 3: Calculate Q1 and Q3
Q1 = df['value'].quantile(0.25)
Q3 = df['value'].quantile(0.75)

# Step 4: Compute the IQR
IQR = Q3 - Q1

# Step 5: Calculate lower and upper fences
lower_fence = Q1 - 1.5 * IQR
upper_fence = Q3 + 1.5 * IQR

print(f"Q1: {Q1}")
print(f"Q3: {Q3}")
print(f"IQR: {IQR}")
print(f"Lower Fence: {lower_fence}")
print(f"Upper Fence: {upper_fence}")

# Step 6: Detect outliers
df['outlier'] = ((df['value'] < lower_fence) | (df['value'] > upper_fence))

# Display results
print("Outliers detected:")
print(df[df['outlier']])
```

```
Q1: 14.75
Q3: 22.5
IQR: 7.75
Lower Fence: 3.125
Upper Fence: 34.125
Outliers detected:
    value  outlier
11     45     True
```

Support Vector Machine

- Support Vector Machine (SVM), a powerful supervised machine learning algorithm, can be adapted for anomaly detection using a specialized technique known as One-Class SVM (OC-SVM).
- In this context, SVM is employed to identify instances that deviate from the majority of the data points, i.e., anomalies.

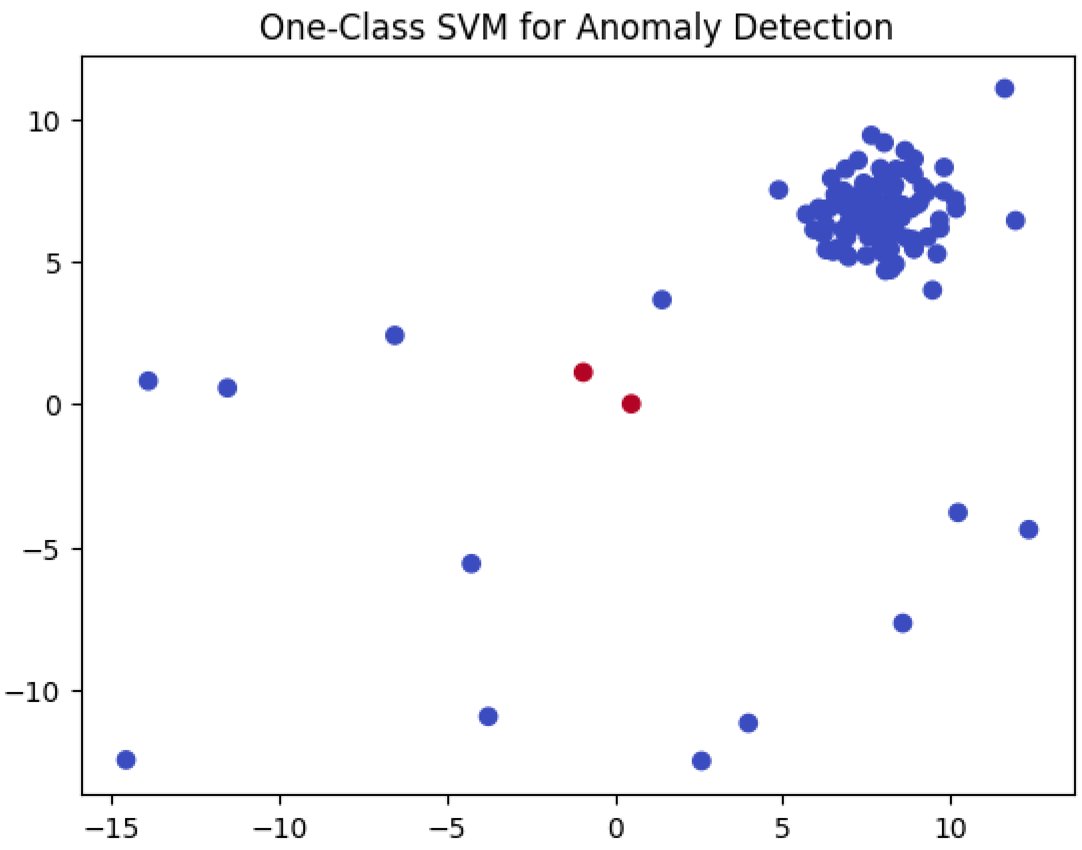
Key Concepts of SVM for Anomaly Detection:

- SVM Basics:
    - SVM is fundamentally a margin-based classifier that works by finding a hyperplane that best separates data into distinct classes. The key idea is to maximize the margin (distance) between the separating hyperplane and the closest data points.
- One-Class SVM:
    - In anomaly detection, One-Class SVM is used to identify outliers. Instead of separating two classes, OC-SVM aims to define a boundary around the "normal" class, separating it from outliers.
    - OC-SVM is trained only on the "normal" instances in the data. It learns the boundary that encloses the majority of data points, and any data point that falls outside this boundary is considered an anomaly.
- Kernel Trick:
    - SVMs use the kernel trick to project data into higher dimensions, making it easier to separate complex data patterns using a hyperplane. In OC-SVM, the kernel function (e.g., linear, radial basis function (RBF)) is crucial for handling non-linearly separable data in anomaly detection.
- Support Vectors in Anomaly Detection:
    - In anomaly detection using One-Class SVM, the support vectors represent the boundary points of the "normal" data distribution. These support vectors determine the shape of the boundary, and anomalies are the points that fall outside this region.
    - While traditional SVMs are used for classification or regression tasks, One-Class SVM focuses on distinguishing the "normal" data points from the "abnormal" ones.

Example Code: One-Class SVM in Python (Using Scikit-Learn)

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.preprocessing import StandardScaler

# Step 1: Create a synthetic dataset
np.random.seed(42)
# Generate normal data points (e.g., 100 2D points centered around (0, 0))
X_train = 0.3 * np.random.randn(100, 2)
# Generate 20 anomalous points far from the normal cluster
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))

# Combine the normal and outlier points into a single dataset for prediction
X_test = np.r_[X_train + 2, X_outliers]

# Step 2: Standardize the data (important for SVMs)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 3: Train One-Class SVM
# Create a One-Class SVM with RBF kernel
oc_svm = svm.OneClassSVM(kernel='rbf', gamma=0.1, nu=0.1)
oc_svm.fit(X_train)

# Step 4: Predict anomalies in the test data
y_pred_train = oc_svm.predict(X_train)
y_pred_test = oc_svm.predict(X_test)

# Anomalies are labeled as -1, and normal points are labeled as 1
n_error_train = y_pred_train[y_pred_train == -1].size
n_error_test = y_pred_test[y_pred_test == -1].size

print(f'Number of anomalies in training data: {n_error_train}')
print(f'Number of anomalies in test data: {n_error_test}')

# Step 5: Plot the results
plt.title("One-Class SVM for Anomaly Detection")
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred_test, cmap='coolwarm')
plt.show()
```

```
Number of anomalies in training data: 11
Number of anomalies in test data: 118
```



K-Nearest Neighbors (KNN)

- K-Nearest Neighbors (KNN) is a simple yet powerful machine learning algorithm that can be effectively utilized for anomaly detection.
- The KNN algorithm is based on the idea of identifying the k nearest points in the feature space to determine whether a given point is an anomaly based on its distance from its neighbors.

Key Concepts of KNN for Anomaly Detection:

- Distance Metric:
    - KNN relies on a distance metric (e.g., Euclidean, Manhattan, Minkowski) to measure how far apart data points are in the feature space. This metric plays a crucial role in determining the neighbors and, consequently, in classifying points as normal or anomalous.
- Local Density:
    - Anomalies are often identified based on local density. If a point is surrounded by many neighbors in a local region, it is considered normal; if it is far from its neighbors, it is flagged as an anomaly. The k nearest neighbors provide a local context for each data point.

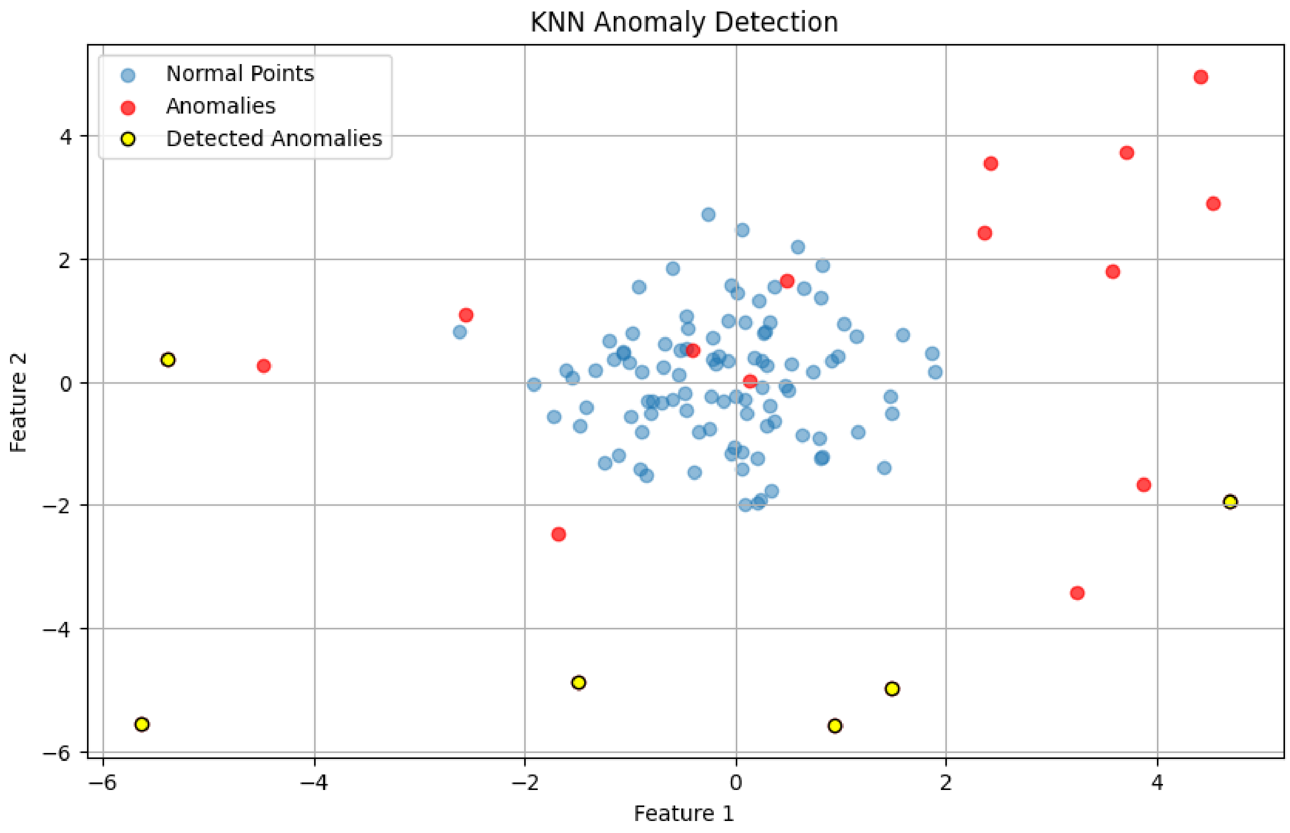
- Parameter k:
    - The choice of k (the number of nearest neighbors to consider) is important. A small k can make the model sensitive to noise, leading to false positives (incorrectly classifying normal points as anomalies), while a large k can smooth out the distinction between normal and anomalous points.

Steps in Using KNN for Anomaly Detection:

- Data Preparation:
    - Start with a dataset containing both normal and anomalous data points. It's preferable to have a majority of normal instances to train the KNN model effectively.
- Distance Calculation:
    - For each data point, calculate the distance to all other points in the dataset.
- Identifying Neighbors:
    - For each data point, identify the k nearest neighbors based on the calculated distances.
- Anomaly Scoring:
    - Anomaly scores can be calculated based on the distance to the k nearest neighbors. A common method is to average the distances to the nearest neighbors. Points with higher average distances are more likely to be anomalies.
- Thresholding:
    - Set a threshold on the anomaly scores to classify data points as normal or anomalous. Points with scores above the threshold are considered anomalies.

Example: KNN for Anomaly Detection in Python (Using Scikit-Learn)

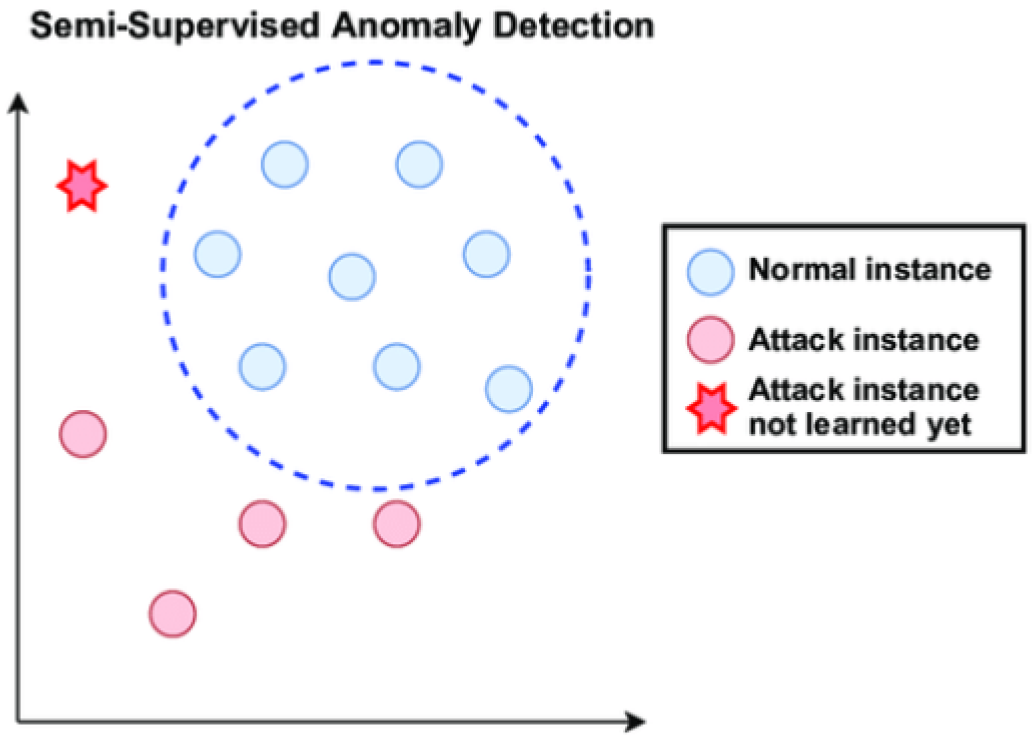
```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

# Step 1: Create a synthetic dataset
np.random.seed(42)
# Generate normal data points (100 2D points centered around (0, 0))
X_normal = np.random.randn(100, 2)
# Generate anomalous points (20 points far from the normal cluster)
X_anomalies = np.random.uniform(low=-6, high=6, size=(20, 2))

# Combine the normal and anomaly points
X = np.vstack((X_normal, X_anomalies))

# Step 2: Fit the KNN model
k = 5  # Number of neighbors
knn = NearestNeighbors(n_neighbors=k)
knn.fit(X)

# Step 3: Calculate distances to the k-nearest neighbors
distances, indices = knn.kneighbors(X)
average_distances = np.mean(distances, axis=1)

# Step 4: Define a threshold for anomaly detection
threshold = np.percentile(average_distances, 95)  # Top 5% as anomalies

# Step 5: Identify anomalies
anomalies = X[average_distances > threshold]

# Step 6: Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(X_normal[:, 0], X_normal[:, 1], label='Normal Points', alpha=0.5)
plt.scatter(X_anomalies[:, 0], X_anomalies[:, 1], label='Anomalies', color='red',
plt.scatter(anomalies[:, 0], anomalies[:, 1], label='Detected Anomalies', color='
plt.title("KNN Anomaly Detection")
plt.legend()
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid()
plt.show()

print(f'Number of detected anomalies: {len(anomalies)}')
```



```
Number of detected anomalies: 6
```

Semi-supervised Anomaly Detection

- Semi-supervised anomaly detection is a machine learning approach used to identify unusual patterns or outliers in a dataset that contains both labeled and unlabeled data.
- This method leverages the labeled examples (often a small subset) to learn the characteristics of normal data while using the larger set of unlabeled data to improve the detection of anomalies.
- It is particularly useful in domains where acquiring labeled data is expensive or time-consuming, such as fraud detection, network security, and healthcare.


Semi-Supervised Anomaly Detection
Normal instance
Attack instance
Attack instance not learned yet

Methodologies for Semi-Supervised Anomaly Detection

1. Data Preprocessing
   - Data Cleaning: Handle missing values, remove duplicates, and standardize formats.
   - Feature Engineering: Create new features that may enhance the model's ability to detect anomalies.
   - Normalization: Scale features to a standard range to ensure uniformity.
2. Model Selection
   - Common algorithms used for semi-supervised anomaly detection include One-Class SVM.
   - A variant of Support Vector Machines that learns the boundary of normal data points. It is effective when labeled data is limited and can distinguish between normal and anomalous classes.
3. Training Process
   - Labeled Data: Train the model on available labeled data to learn the characteristics of normal data.
   - Unlabeled Data: Use unlabeled data to enhance the model's understanding of the feature space, improving the decision boundary for detecting anomalies.
4. Anomaly Scoring
   - Once the model is trained, each data point is assigned an anomaly score based on its distance from the learned representation of normal data.
   - Anomalies can be identified by setting a threshold on the anomaly score, with points exceeding this threshold classified as anomalous.

One-Class Support Vector Machine (SVM)

- One-Class Support Vector Machine (One-Class SVM) is a specialized version of the Support Vector Machine algorithm designed specifically for anomaly detection.
- It is particularly useful in scenarios where the data is heavily imbalanced, meaning there are significantly fewer instances of anomalies compared to normal data points.

Key Concepts of One-Class SVM:

- Boundary Definition:
    - One-Class SVM attempts to learn a decision boundary that encompasses the majority of the normal data points while excluding the anomalies.
    - The idea is to find a hyperplane that separates the normal instances from the origin in the feature space.
- Kernel Trick:
    - One-Class SVM utilizes the kernel trick to transform the data into a higher-dimensional space.
    - This transformation allows for the construction of more complex decision boundaries, enabling the model to capture non-linear relationships between features.
- Soft Margin:
    - Like traditional SVMs, One-Class SVM employs a soft margin approach, allowing some normal data points to fall outside the decision boundary.
    - This is controlled by a hyperparameter (nu) that defines the trade-off between the number of support vectors and the acceptable fraction of anomalies in the dataset.

Steps in Using One-Class SVM for Anomaly Detection:

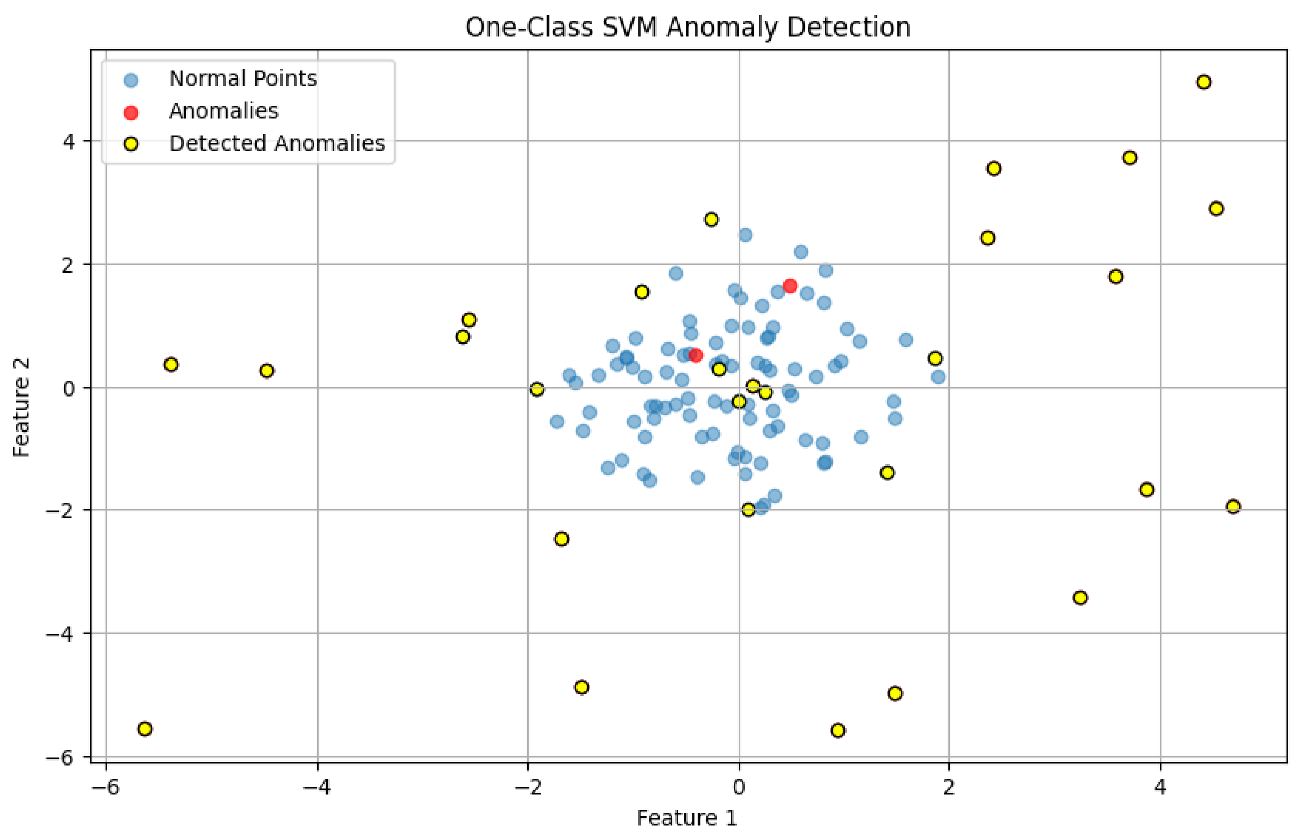
- Data Preparation:
    - Start with a dataset that contains predominantly normal instances. It’s preferable to have a clear representation of the normal class, as One-Class SVM learns exclusively from this class.
- Model Training:
    - Fit the One-Class SVM model on the normal data points using an appropriate kernel (e.g., linear, RBF, polynomial).
- Anomaly Scoring:
    - For each point in the dataset, compute the distance from the decision boundary. Points falling outside this boundary are classified as anomalies.
- Thresholding:

- Establish a threshold to determine which points are considered anomalies based on their distance from the boundary.

- Evaluation:
  - Evaluate the performance of the One-Class SVM using metrics such as precision, recall, and F1-score, if labeled data is available.

Example: One-Class SVM for Anomaly Detection in Python (Using Scikit-Learn)

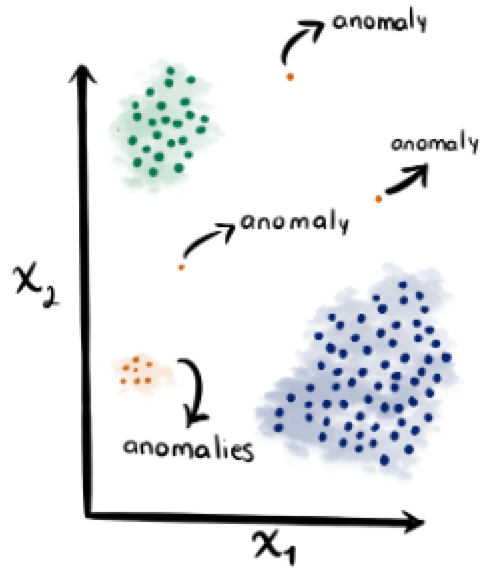
```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import OneClassSVM

# Step 1: Create a synthetic dataset
np.random.seed(42)
# Generate normal data points (100 2D points centered around (0, 0))
X_normal = np.random.randn(100, 2)
# Generate anomalous points (20 points far from the normal cluster)
X_anomalies = np.random.uniform(low=-6, high=6, size=(20, 2))

# Combine the normal and anomaly points
X = np.vstack((X_normal, X_anomalies))

# Step 2: Fit the One-Class SVM model
ocsvm = OneClassSVM(kernel='rbf', gamma='scale', nu=0.1)  # nu is the upper bound
ocsvm.fit(X_normal)

# Step 3: Predict anomalies
y_pred = ocsvm.predict(X)
# -1 indicates anomaly, 1 indicates normal
anomalies = X[y_pred == -1]

# Step 4: Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(X_normal[:, 0], X_normal[:, 1], label='Normal Points', alpha=0.5)
plt.scatter(X_anomalies[:, 0], X_anomalies[:, 1], label='Anomalies', color='red',
plt.scatter(anomalies[:, 0], anomalies[:, 1], label='Detected Anomalies', color='
plt.title("One-Class SVM Anomaly Detection")
plt.legend()
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid()
plt.show()

print(f'Number of detected anomalies: {len(anomalies)}')
```



```
Number of detected anomalies: 28
```

Unsupervised Anomaly Detection

- Unsupervised anomaly detection is a technique used to identify unusual or abnormal observations in datasets without the need for labeled examples.
- Unlike supervised learning methods that require annotated data with known classes, unsupervised methods operate solely on the data itself, seeking to identify patterns, clusters, and deviations from these patterns.



Key Concepts

- Lack of Labels:
    - Unsupervised anomaly detection works with datasets where normal and anomalous instances are not labeled. The algorithm must identify the underlying structure of the data to distinguish between regular and outlier observations.
- Data Representation:
    - The approach often involves transforming the data into a suitable representation that highlights its essential features, making it easier to identify anomalies.
- Pattern Recognition:
    - Unsupervised methods rely on recognizing patterns in the data. Anomalies are detected as points that do not conform to these established patterns.

Limitations of Unsupervised Anomaly Detection

- Higher False Positive Rate: Unsupervised methods may produce a higher rate of false positives compared to supervised methods, as they lack context on what constitutes a true anomaly.
- Assumption of Distribution: Some methods (like Z-score) assume a specific distribution of the data, which may not hold true for all datasets.
- Sensitivity to Parameters: Many unsupervised methods are sensitive to the choice of parameters (e.g., the number of clusters in K-means, distance metrics in KNN), which can significantly affect the results.

Isolation Forest Algorithm

- Isolation Forest (iForest) is a machine learning algorithm specifically designed for anomaly detection.

- It is based on the principle of isolating anomalies instead of profiling normal points, making it one of the most effective and efficient techniques for detecting outliers in large datasets.

Key Idea Behind Isolation Forest

- The key insight of the Isolation Forest algorithm is that anomalies are:
    - Few in number: They constitute a small fraction of the data.
    - Different: They tend to be far from the majority of data points.
- Isolation Forest works by recursively splitting data points in a way that isolates anomalies faster. The basic idea is that anomalies are easier to isolate because they have unique attributes that differ significantly from normal points. The fewer splits it takes to isolate a point, the more likely it is to be an anomaly.

How Isolation Forest Works

- Random Partitioning:
    - Isolation Forest constructs multiple decision trees (called "isolation trees" or "iTrees") by randomly selecting a feature and a random split value between the minimum and maximum values of that feature.
    - Each tree isolates data points by making splits. A point is isolated when it is separated from other data points in the feature space.
- Shorter Paths for Anomalies:
    - Anomalies are isolated faster (i.e., with fewer splits) since they are located far from other points and their values differ more drastically.
    - Normal points take more splits to isolate, as they are more densely packed with other similar points.
- Tree Construction:
    - The algorithm builds a forest of iTrees, with each tree representing a recursive partition of the data.
    - Each tree isolates points, and the number of splits required to isolate each point is recorded.
- Anomaly Score:
    - After the forest of trees is built, each point is assigned an anomaly score based on how easy it was to isolate.
    - Points that are isolated with fewer splits (shorter paths in the tree) are more likely to be anomalies. The anomaly score is a measure of the "average path length" across all trees for each point.
    - The anomaly score ranges from 0 to 1:
        - A score close to 1 indicates the point is an anomaly.

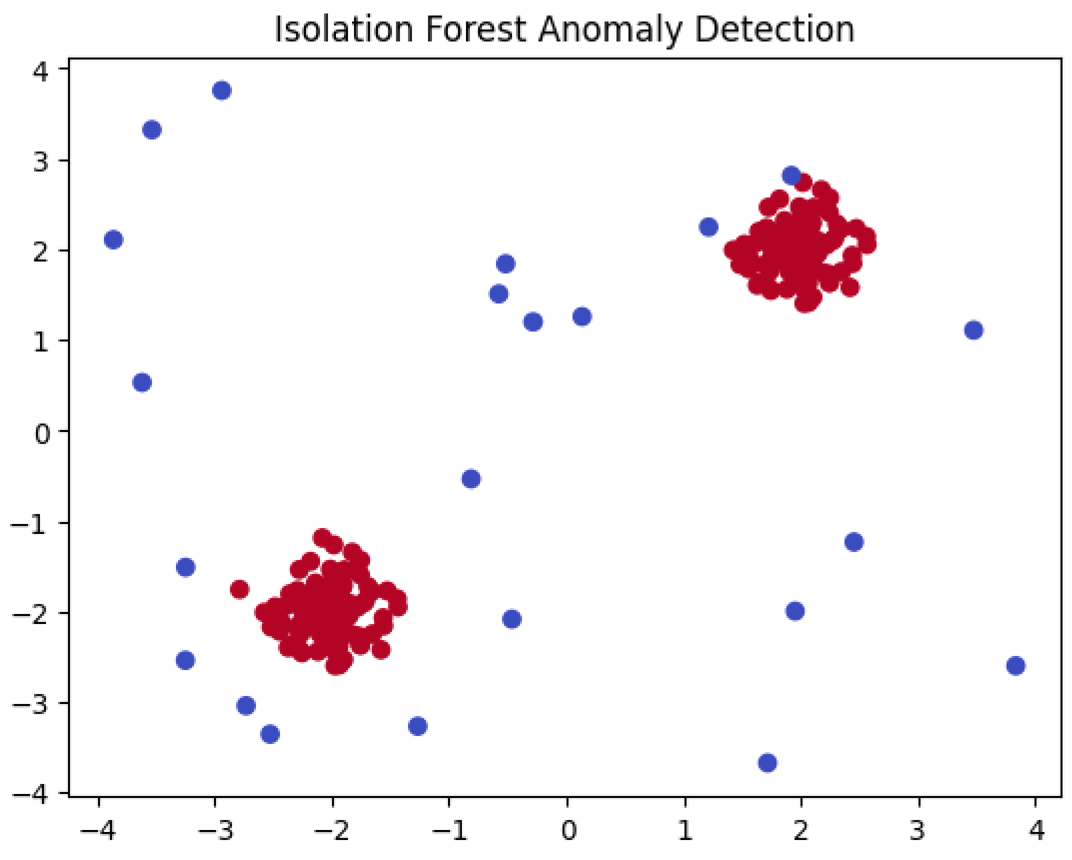
        - A score close to 0 indicates the point is likely normal.

Steps in Isolation Forest Algorithm

- Build an Isolation Tree:
    - Select a feature randomly.
    - Select a random split value within the feature range.
    - Split the data using the random feature and split value.
    - Recursively apply this process until each point is isolated or a certain depth is reached.
- Construct a Forest:
    - Repeat the tree-building process multiple times to create a forest of isolation trees.
    - Each tree is constructed using a random subset of the data.
- Compute Anomaly Scores:
    - For each point, calculate the average path length across all trees.
    - Convert the path length into an anomaly score. The shorter the path, the more likely the point is an anomaly.
- Set a Threshold:
    - Set a threshold on the anomaly score to classify points as anomalies or normal points.

Here’s a simple example of how to use Isolation Forest for anomaly detection in Python using scikit-learn:

```
# Import libraries
from sklearn.ensemble import IsolationForest
import numpy as np
import matplotlib.pyplot as plt

# Generate sample data (normal data)
rng = np.random.RandomState(42)
X = 0.3 * rng.randn(100, 2)
X = np.r_[X + 2, X - 2]

# Add some anomalies
X_anomalies = np.random.uniform(low=-4, high=4, size=(20, 2))
X = np.r_[X, X_anomalies]

# Fit the Isolation Forest model
clf = IsolationForest(contamination=0.1, random_state=rng)
clf.fit(X)

# Predict anomalies
y_pred = clf.predict(X)

# Plot the data and highlight anomalies
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='coolwarm')
plt.title("Isolation Forest Anomaly Detection")
plt.show()
```



Local Outlier Factor

- Local Outlier Factor (LOF) is a popular density-based algorithm used for unsupervised anomaly detection.
- The key idea behind LOF is to compare the local density of a data point to the densities of its neighbors.
- If a data point’s density is significantly lower than its neighbors', it is considered an outlier.
- Unlike global anomaly detection methods, which look for anomalies based on the overall distribution of the dataset, LOF detects anomalies that are local, meaning that an object is considered an anomaly only in relation to its surrounding data points.

Steps of the LOF Algorithm

- Choose k (the number of neighbors):
  - Select a value of k, which determines the number of neighbors considered when calculating the local density.
- Calculate k-distance:
  - For each point, compute the distance to its k-th nearest neighbor. This is used to determine the reachability distance between points.
- Compute reachability distance:
  - For each point, compute the reachability distance to its k-nearest neighbors.
- Calculate local reachability density (LRD):
  - For each point, calculate the local reachability density based on the reachability distances to its k-nearest neighbors.
- Compute Local Outlier Factor (LOF):
  - For each point, compute the LOF as the ratio of its local reachability density to the local reachability density of its neighbors.
- Identify anomalies:
  - Points with LOF values significantly larger than 1 are considered outliers. The higher the LOF value, the more anomalous the point.\

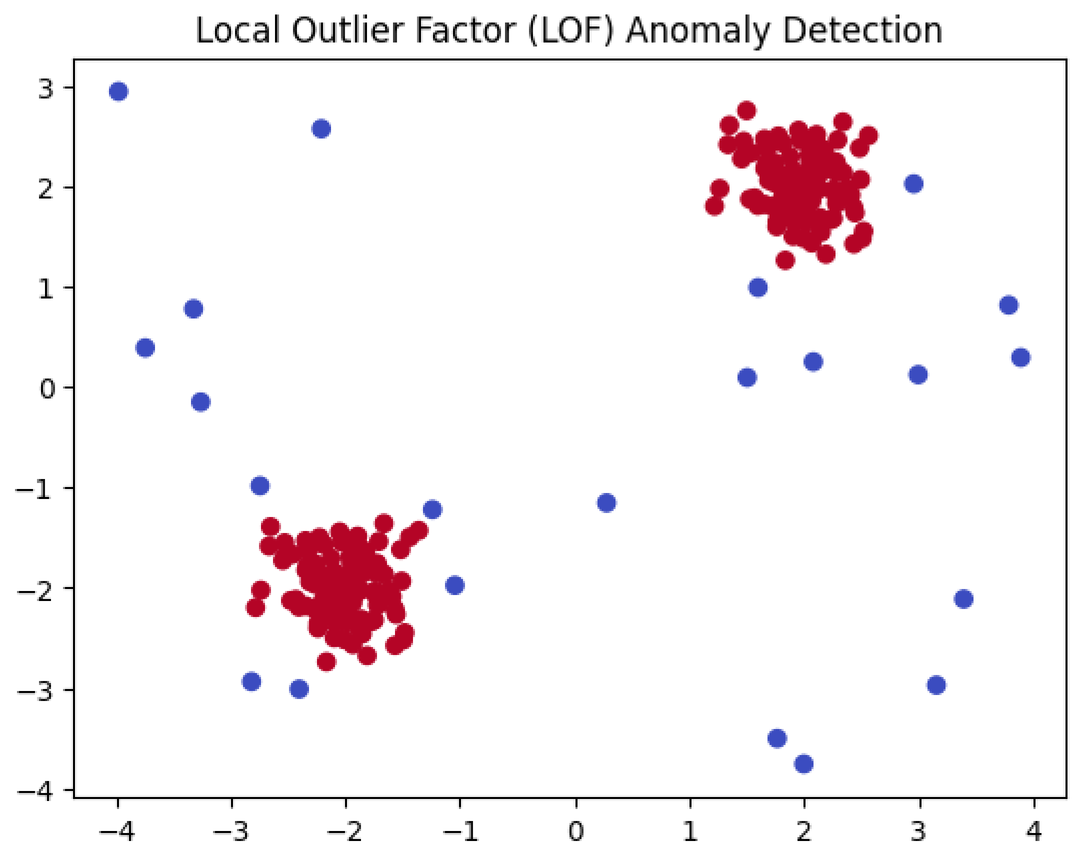
Advantages of LOF

- Detects Local Outliers:
  - Unlike global anomaly detection methods, LOF focuses on identifying local outliers that may not be apparent when considering the entire dataset.
  - This is particularly useful in datasets where different regions have different densities, making it effective for real-world datasets with complex patterns.
- No Assumptions about Data Distribution:

  - LOF is non-parametric, meaning it does not assume any particular distribution of the data (e.g., Gaussian).
- Handles Multidimensional Data:
  - LOF can handle high-dimensional data because it uses the concept of nearest neighbors, making it suitable for datasets with many features.

Here's an example of how to implement LOF using scikit-learn:

```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor

# Generate sample data
X_inliers = 0.3 * np.random.randn(100, 2)
X_inliers = np.r_[X_inliers + 2, X_inliers - 2]

# Add some outliers
X_outliers = np.random.uniform(low=-4, high=4, size=(20, 2))

# Combine inliers and outliers
X = np.r_[X_inliers, X_outliers]

# Fit the LOF model
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
y_pred = lof.fit_predict(X)

# Plot the results
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='coolwarm')
plt.title("Local Outlier Factor (LOF) Anomaly Detection")
plt.show()
```



DBSCAN

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that can be used for both clustering and anomaly detection.
- It is particularly effective in detecting outliers or anomalies by identifying points that lie in low-density regions of the data.
- DBSCAN is based on the idea that clusters are areas of high density separated by areas of low density.
- Points that do not belong to any cluster are considered outliers (or noise), making DBSCAN useful for unsupervised anomaly detection.

How DBSCAN Detects Anomalies

- DBSCAN is capable of detecting anomalies by identifying points that do not belong to any cluster. Specifically, DBSCAN labels points as noise when:
    - They do not have enough neighbors to form a dense region (i.e., fewer than minPts neighbors within a distance of eps).
    - They cannot be reached by other core points through their neighbors.
- These points are considered outliers or anomalies, as they do not conform to the overall structure of the data.

DBSCAN Steps for Anomaly Detection

- Choose eps and minPts:

  - Select the parameters eps (radius) and minPts (minimum number of points required to form a cluster). The selection of these parameters is critical for detecting clusters and anomalies.
- Find Core Points:
  - For each point in the dataset, calculate the number of points within its eps neighborhood. If the number of points is greater than or equal to minPts, the point is marked as a core point.
- Form Clusters:
  - For each core point, recursively find all density-reachable points (those reachable from the core point within eps), and assign them to the same cluster.
  - Border points are added to clusters if they are within the neighborhood of a core point.
- Identify Noise:
  - Any point that is not a core point or a border point is labeled as noise, indicating that it is an anomaly or outlier.

Example of DBSCAN Anomaly Detection

```
# Import libraries
import numpy as np
```