Real-Time Adaptive Anomaly Detection

Overview

This project implements a **Efficient Data Stream Anomaly Detection** that monitors a simulated data stream and flags unusual values. The system visualizes the data stream and marks detected anomalies on a live plot. Using a moving average approach, this system is lightweight and effective for streaming data with periodic patterns and noise.

Structure

- **generate_data_stream**: Generates a stream of data points with seasonality and random noise, simulating real-world data behavior.
- **adaptive_anomaly_detection**: Analyzes data to detect outliers in real time based on a rolling mean and standard deviation.
- **real_time_monitor**: Visualizes the live data stream, marking detected anomalies in real-time.

Installation and Requirements

This project requires the following Python libraries:

pip install numpy matplotlib

Code Structure and Explanation

1. `generate_data_stream(num_points=1)`

- **Purpose**: Simulates a data stream with periodic patterns and noise, reflecting fluctuations often seen in real-world streaming data.
- **Parameters**:
    - `num_points`: The number of data points to generate in the stream. Default is 1.
- **Return**: Returns a signal with both seasonal components and random noise.

- **Explanation**: Uses a sinusoidal function to create a regular pattern, adding Gaussian noise to introduce randomness.

Code:

```
def generate_data_stream(num_points=1):
    """Simulates a data stream with regular patterns, seasonality, and random noise."""
    time = np.arange(num_points)
    signal = np.sin(0.1 * time) + 0.5 * np.random.normal(size=num_points)
    return signal
```

2. `adaptive_anomaly_detection(data, window_size=20, threshold=3)`

- **Purpose**: Detects anomalies in the latest data point based on its deviation from a rolling mean and standard deviation.
- **Parameters**:
  - `data`: The deque containing recent data points.
  - `window_size`: The number of data points to consider for calculating the rolling statistics.
  - `threshold`: The Z-score threshold beyond which a point is considered anomalous.
- **Return**: Returns a list of detected anomalies.
- **Explanation**: Computes the rolling mean and standard deviation of the last `window_size` points. If the latest point exceeds the specified Z-score threshold, it is flagged as an anomaly.

Code:

```
def adaptive_anomaly_detection(data, window_size=20, threshold=3):
    """Detects anomalies in data based on deviation from rolling mean and std deviation."""
    anomalies = []
    if len(data) >= window_size:
        window_data = list(data)[-window_size:]
        mean = np.mean(window_data)
        std_dev = np.std(window_data)

        if abs((data[-1] - mean) / std_dev) > threshold:
            anomalies.append((len(data) - 1, data[-1]))
    return anomalies
```

3. `real_time_monitor(window_size=20)`

- **Purpose**: Continuously monitors the data stream, detecting and plotting anomalies in real time.
- **Parameters**:
  - `window_size`: Specifies the number of points for calculating the rolling statistics.
- **Explanation**:
  - Initializes a deque to store recent data and sets up a live plot for real-time visualization.
  - For each new data point:
    - Updates the stream and checks for anomalies.
    - Redraws the plot, marking anomalies with red circles.
  - Uses `plt.pause(0.1)` to create real-time updates and finally closes the interactive mode.

Code:

```
def real_time_monitor(window_size=20):
    """Monitors data stream in real-time, marking anomalies on a live plot."""
    stream_data = deque(maxlen=window_size)
    anomalies = []
    plt.ion()
    fig, ax = plt.subplots()
    line, = ax.plot([], [], label='Data Stream')
    anomaly_points, = ax.plot([], [], 'ro', label='Anomalies')
    try:
        for _ in range(100):
            new_data = generate_data_stream(1)[0]
            stream_data.append(new_data)
            new_anomalies = adaptive_anomaly_detection(stream_data,
window_size=window_size)
            anomalies.extend(new_anomalies)
```

```
            ax.clear()
            ax.plot(range(len(stream_data)), stream_data, label='Data Stream')
            anomaly_indices = [idx for idx, _ in anomalies]
            anomaly_values = [stream_data[idx - len(stream_data)] for idx in anomaly_indices]
            ax.plot(anomaly_indices, anomaly_values, 'ro', label='Anomalies')
            ax.legend()
            plt.pause(0.1)
    except Exception as e:
        print("Error in real_time_monitor:", e)
    finally:
        plt.ioff()
        plt.show()
```

Algorithm Explanation

Adaptive Anomaly Detection Algorithm

This system uses a moving average with a rolling window for anomaly detection:

- **Rolling Mean and Standard Deviation**: Calculates statistics for the latest `window_size` points in the data stream.
- **Z-Score Thresholding**: Flags a point as anomalous if its Z-score, calculated as the deviation from the mean in standard deviations, exceeds the set `threshold`.

Why It's Effective:

- **Adaptive**: The rolling window allows the system to adjust to new patterns in the data stream dynamically.
- **Real-time Suitability**: Fast calculations make it ideal for high-frequency data.
- **Simplicity**: With only basic statistical methods, it achieves quick and reliable results without complex model training.

Execution

To run the program, execute the following:

python main.py

This will start the real-time data monitoring with a live plot displaying the data stream and anomalies as they are detected.

Conclusion

This real-time adaptive anomaly detection system provides a streamlined solution for monitoring and visualizing data streams. By using an adaptable, rolling average method, it ensures robust anomaly detection with minimal computational overhead.