Efficient Data Stream Anomaly Detection

Submitted by:

Moataz Eldeeb

moatazdx1@gmail.com

01113339422

For:

Graduate Software Engineer role at Cobblestone Energy

Algorithm Selection:

The anomaly detection approach here is based on **trend, seasonality, and residuals decomposition**. The time series data is decomposed into three components:

1. Trend: Represents the long-term movement or pattern in the data.
2. Seasonality: Captures the repeating cyclical behavior within the data over a fixed period.
3. Residuals: Represents the remaining noise or random fluctuations after removing the trend and seasonality.

Anomaly Detection Process:

1. Decomposition: The time series is decomposed into its trend, seasonality, and residuals using either a moving average or similar decomposition technique.
2. Residuals as Anomalies: After removing the trend and seasonality, the residuals represent the unpredictable part of the data. Values that significantly deviate from zero in the residuals are flagged as anomalies, since they do not conform to the normal trend or seasonality.
3. Threshold Setting: A threshold (e.g., based on standard deviation or percentile) is set to define which residuals are considered outliers.

Effectiveness:

1. Seasonality and Trend Handling: This method is effective because it accounts for the repeating patterns and long-term trends in the data, reducing false positives caused by seasonal variations.
2. Flexibility: By focusing on the residuals, it isolates true anomalies (random, unexpected deviations) that aren't explained by normal trend or seasonal patterns.
3. Practicality: It's highly effective in time series data where trends and seasonality are strong, such as sales data, stock prices, or weather patterns.

Realtime plotting and running the algorithm

```python
def animate(i):
    """
        Function to be called each time the interval of [FuncAnimation]
        finishes.
    """
    # Read the csv
    data = pd.read_csv('data.csv')

    # Get x-axis values and y-axis values
    x= data['x_value']
    y = data['y_value']


    outlier_indexes = annomaly_detection(data)
    print("Anomalies:")
    print(outlier_indexes)

    # Clear output of plt figure
    plt.cla()

    # Plot the data points using blue color
    plt.plot(x,y, label='Values', color='blue', marker='o')
    # Highlight the outliers using red color
    plt.plot(outlier_indexes, data.loc[outlier_indexes, 'y_value'], 'ro', label='Anomalies')

    plt.title(f'Data Points with seasonal period = {SEASONAL_PERIOD}')

    # Add legend
    plt.legend()
    plt.tight_layout()
```

Data Stream Simulation

```python
def generate_data_stream(n_points=1000, trend=0.01, seasonality_amplitude=10, noise_std=1, seasonality_period=50, stream_speed=1):
    """
    Simulates a data stream with a trend, seasonality, and noise.

    Parameters:
    - n_points: Number of data points to generate. if None it will run indefinitely
    - trend: The linear trend component (e.g., 0.01 for a gradual upward trend).
    - seasonality_amplitude: Amplitude of the seasonal fluctuations.
    - noise_std: Standard deviation of the random noise.
    - seasonality_period: The period of the seasonality (e.g., 50 for periodic fluctuations).
    - stream_speed: Time delay between each data point (in seconds).

    Yields:
    - Data points one at a time.
    """
    if n_points is None:
        t=0
        while True:
            # Trend component: grows over time (linear trend)
            trend_component = trend * t
            t+=1

            # Seasonality component: a sine wave to simulate periodic behavior
            seasonality_component = seasonality_amplitude * np.sin(2 * np.pi * t / seasonality_period)

            # Random noise component: normally distributed noise
            noise_component = np.random.normal(0, noise_std)

            # Combine all components to form the final data point
            value = trend_component + seasonality_component + noise_component

            # Wait to simulate a real-time data stream
            time.sleep(stream_speed)

            yield value
    else:
        for t in range(n_points):
            # Trend component: grows over time (linear trend)
            trend_component = trend * t

            # Seasonality component: a sine wave to simulate periodic behavior
            seasonality_component = seasonality_amplitude * np.sin(2 * np.pi * t / seasonality_period)

            # Random noise component: normally distributed noise
            noise_component = np.random.normal(0, noise_std)

            # Combine all components to form the final data point
            value = trend_component + seasonality_component + noise_component

            # Wait to simulate a real-time data stream
            time.sleep(stream_speed)

            yield value
```

This function simulates a data stream that includes a linear trend, seasonal patterns, and random noise. It is useful for testing time series algorithms, real-time processing applications, or anomaly detection systems. The function yields data points in a real-time fashion, one by one, mimicking the behavior of a data stream.

Function Overview:

The simulated data stream has three components:

1. **Trend Component**: A linear trend that increases or decreases over time.
2. **Seasonality Component**: A repeating cyclical pattern (e.g., a sine wave).

3. **Noise Component**: Random fluctuations (noise) added to the data to make it less predictable.

The function runs for n_points steps and yields each data point after a short delay, simulating real-time data generation

Output:

As we can see below some anomalies where detected, the output shows that the algorithm is capable of adapting to concept drift and seasonal variations.