LSTM Autoencoder Anomaly Detection Model Documentation

1. Introduction

This Model puts into action an anomaly detection algorithm to spot unusual patterns in stock price data as it comes in. It uses LSTM (Long Short-Term Memory) autoencoders to do this. The goal is to find odd price movements that might point to big market events or chances to trade.

The Model brings together machine learning methods with real-time data handling and display. This creates a useful tool for people who analyze finances and those who use computers to trade.

Here's why LSTM is particularly useful for anomaly detection in real-time stream data:

- Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data.
- LSTMs are well-suited for time series data like stock prices because they can remember relevant information over long periods and forget irrelevant details.
- The autoencoder architecture (encoding followed by decoding) allows the model to learn a compressed representation of normal patterns in the data.
- By comparing the reconstructed output to the input, the model can identify anomalies as data points that are difficult to reconstruct accurately.

LSTM's efficiency for anomaly detection in real-time stream data:

1. Handling concept drift: LSTMs can adapt to gradual changes in the underlying data distribution (concept drift) by continuously updating their internal state. This makes them more robust to evolving patterns in stock prices.
2. Capturing complex temporal dependencies: Unlike simpler models, LSTMs can capture intricate temporal patterns and relationships in the data, which is crucial for accurately modeling stock price behavior.
3. Online learning capability: LSTMs can be updated incrementally with new data, making them suitable for real-time applications where the model needs to evolve with incoming data.
4. Noise resilience: LSTMs are good at distinguishing between normal variations (noise) and true anomalies in the data, reducing false positives in anomaly detection.

2. System Components

The Model has several main parts:

1. Data Handling: Functions for generating synthetic data and fetching real stock data.

2. LSTM Autoencoder Model: A neural network architecture for learning normal patterns in stock price movements.
3. Anomaly Detection: Algorithms for identifying deviations from learned normal patterns.
4. Real-time Visualization: A dynamic plotting system to display stock prices, predictions, and detected anomalies.

3. Installation and Dependencies

To use this Model, ensure you have the following dependencies installed:

```
numpy
matplotlib
tensorflow
yfinance
```

You can install these dependencies using pip:

```
pip install numpy matplotlib tensorflow yfinance
```

4. Usage

To use the Model, follow these steps:

1. Import the necessary modules and functions.
2. Fetch or generate stock price data.
3. Prepare the data for training.
4. Build and train the LSTM autoencoder model.
5. Apply the trained model to real-time data for anomaly detection.

Example usage:

```
# Fetch real stock data
real_stock_prices = fetch_real_stock_data(ticker='NVDA', period='10y',
interval='1d')
stream = (price for price in real_stock_prices)

# Prepare training data
X_train = collect_normal_data(stream, num_samples=3000, time_steps=10)
```

```
# Build and train the model
lstm_autoencoder = build_lstm_autoencoder((10, 1))
lstm_autoencoder.fit(X_train, X_train, epochs=150, batch_size=64,
validation_split=0.2, shuffle=True)

# Run real-time analysis
real_time_plot_with_dynamic_threshold(stream, lstm_autoencoder)
```

5. Key Functions

synthetic_stock_price_stream(mean=100, seasonal_variation=5, noise_level=1.0, trend=0.1, volatility=1.0)

Generates a synthetic stream of stock prices with configurable parameters. It simulates:
- Seasonal variations
- Long-term trends
- Random noise
- Occasional price jumps

This function is useful for testing the model with controlled data before using real stock prices.

fetch_real_stock_data(ticker='VOW3', period='1y', interval='1d')

Fetches real stock price data using the yfinance library.
- It can fetch data for any stock ticker
- Allows specification of the time period and interval
- Returns closing prices

This function provides real-world data for training and testing the model.

build_lstm_autoencoder(input_shape)

Constructs the LSTM autoencoder model architecture.

collect_normal_data(data_stream, num_samples=1000, time_steps=10)

- Collects a specified number of samples from the data stream
- Organizes the data into overlapping windows of a fixed time step
- Prepares the data in the format required by the LSTM model

This step is crucial for creating the training dataset for the model.

calculate_dynamic_threshold(model, window, multiplier=3.0)

Calculates a dynamic threshold for anomaly detection.
- Computes the reconstruction error for a given window of data
- Sets the threshold based on the mean error plus a multiple of the standard deviation
- Allows for adaptive thresholding that adjusts to changing data patterns

real_time_plot_with_dynamic_threshold(data_stream, model, time_steps=10, plot_interval=10, smoothing_window=60)

- Processes the data stream in real-time
- Applies the LSTM model to detect anomalies
- Visualizes the results dynamically, including:
  - Actual stock prices
  - Smoothed predicted prices
  - Dynamic threshold
  - Highlighted anomaly ranges and points

This function provides a comprehensive visual representation of the model's performance in real-time, allowing for immediate identification of anomalies and assessment of the model's accuracy.

6. Model Architecture

The LSTM autoencoder model consists of:

- An encoder LSTM layer (64 units)
- A RepeatVector layer
- A decoder LSTM layer (64 units)
- A TimeDistributed Dense layer

This architecture allows the model to learn a compressed representation of normal stock price patterns and reconstruct them, facilitating anomaly detection.

7. Anomaly Detection Process

1. The model is trained on normal stock price data.
2. For each new data point:
    - It is passed through the trained model for reconstruction.
    - The reconstruction error is calculated.
    - If the error exceeds a dynamic threshold, the point is flagged as an anomaly.
3. The dynamic threshold is continuously updated based on recent error patterns.

8. Visualization

The Model provides real-time visualization of:

- Actual stock prices
- Smoothed predicted prices
- Dynamic threshold
- Detected anomalies (highlighted as red points and ranges)

This visualization updates at specified intervals, providing a clear and immediate representation of the stock's behavior and detected anomalies.

9. Customization and Extension

The Model can be customized in several ways:

- Adjusting model hyperparameters (e.g., number of LSTM units, time steps)
- Modifying the dynamic threshold calculation
- Changing the visualization style or update frequency
- Incorporating additional data sources or technical indicators

To extend the Model, consider:

- Implementing additional anomaly detection algorithms for comparison
- Adding support for multiple stock tickers simultaneously
- Integrating with trading systems for automated decision-making

For any modifications or extensions, ensure to thoroughly test the Model with both synthetic and real data to validate its performance and reliability.

10. Code

```
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, RepeatVector, TimeDistributed,
Dense
from tensorflow.keras.models import load_model
import yfinance as yf

def synthetic_stock_price_stream(mean=100, seasonal_variation=5,
noise_level=1.0, trend=0.1, volatility=1.0):
    t = 0
    price = mean
    while True:
        seasonal_effect = seasonal_variation * np.sin(2 * np.pi * t / 50)
        trend_effect = trend * t
        noise = np.random.normal(0, noise_level)
        jump = np.random.normal(0, volatility) if np.random.rand() < 0.1
else 0
        price += noise + seasonal_effect + trend_effect + jump
        yield price
        t += 1

def fetch_real_stock_data(ticker='VOW3', period='1y', interval='1d'):
    stock_data = yf.download(ticker, period=period, interval=interval)
    return stock_data['Close'].values

def build_lstm_autoencoder(input_shape):
    model = Sequential()
    model.add(LSTM(64, activation='relu', input_shape=input_shape,
return_sequences=False))
    model.add(RepeatVector(input_shape[0]))
    model.add(LSTM(64, activation='relu', return_sequences=True))
    model.add(TimeDistributed(Dense(input_shape[1])))
    model.compile(optimizer='adam', loss='mse')
    return model

def collect_normal_data(data_stream, num_samples=1000, time_steps=10):
    X_train = []
    window = []
```

```
    for i, data_point in enumerate(data_stream):
        if data_point is not None:
            window.append([data_point])
            if len(window) == time_steps:
                X_train.append(window)
                window = []
            if len(X_train) == num_samples:
                break
    return np.array(X_train)

def calculate_dynamic_threshold(model, window, multiplier=3.0):
    window_array = np.array(window).reshape(1, len(window), 1)
    reconstructed = model.predict(window_array)
    error = np.mean(np.abs(window_array - reconstructed))
    return error + multiplier * np.std(error)

def real_time_plot_with_dynamic_threshold(data_stream, model,
time_steps=10, plot_interval=10, smoothing_window=60):
    window = deque(maxlen=time_steps)
    error_window = deque(maxlen=smoothing_window)
    fig, ax = plt.subplots(figsize=(12, 6))
    plt.ion()

    x_data, y_data = [], []
    anomaly_x, anomaly_y = [], []
    anomaly_ranges = []
    prediction_y = []
    smoothed_prediction_y = []
    thresholds = []

    anomaly_start = None

    for idx, data_point in enumerate(data_stream):
        if data_point is not None:
            window.append([data_point])

            if len(window) == time_steps:
                window_array = np.array(window).reshape(1, time_steps, 1)
                reconstructed = model.predict(window_array)
                error = np.mean(np.abs(window_array - reconstructed))
                error_window.append(error)
```

```
                # Calculate dynamic threshold
                dynamic_threshold = np.mean(error_window) + 3 *
np.std(error_window)
                thresholds.append(dynamic_threshold)

                x_data.append(idx)
                y_data.append(data_point)
                prediction_y.append(reconstructed[0, -1, 0])

                # Detect anomaly and update anomaly ranges
                if error > dynamic_threshold:
                    anomaly_x.append(idx)
                    anomaly_y.append(data_point)
                    if anomaly_start is None:
                        anomaly_start = idx
                else:
                    if anomaly_start is not None:
                        anomaly_ranges.append((anomaly_start, idx))
                        anomaly_start = None

                # Calculate smoothed prediction
                if len(prediction_y) >= smoothing_window:
                    smoothed_value =
np.mean(prediction_y[-smoothing_window:])
                else:
                    smoothed_value = np.mean(prediction_y)
                smoothed_prediction_y.append(smoothed_value)

                if idx % plot_interval == 0:
                    ax.clear()
                    ax.plot(x_data, y_data, color='blue', label='Actual
Stock Prices')
                    ax.plot(x_data, smoothed_prediction_y, color='green',
label='Smoothed Predicted Prices', alpha=0.7)
                    ax.plot(x_data, thresholds, color='orange',
linestyle='--', label='Dynamic Threshold')

                    # Plot vertical lines for anomaly ranges
                    for start, end in anomaly_ranges:
                        ax.axvspan(start, end, color='red', alpha=0.3)

                    # Plot red points for individual anomalies
                    ax.scatter(anomaly_x, anomaly_y, color='red',
```

```
marker='o', label='Anomalies')

                    ax.set_xlabel('Time Steps')
                    ax.set_ylabel('Stock Price')
                    ax.set_title('Real-Time Stock Prices with Dynamic
Threshold and Anomaly Detection')
                    ax.legend()
                    plt.pause(0.01)

    # Handle case where anomaly continues to the end
    if anomaly_start is not None:
        anomaly_ranges.append((anomaly_start, len(x_data) - 1))

    plt.savefig()
    plt.ioff()
    plt.show()

if __name__ == '__main__':
    # Fetch real stock prices
    real_stock_prices = fetch_real_stock_data(ticker='NVDA', period='10y',
interval='1d')
    stream = (price for price in real_stock_prices)

    input_shape = (10, 1)
    X_train = collect_normal_data(stream, num_samples=3000, time_steps=10)

    # Build and train the LSTM Autoencoder Model
    lstm_autoencoder = build_lstm_autoencoder(input_shape)
    split = int(0.8 * len(X_train))
    X_train_data = X_train[:split]
    X_val_data = X_train[split:]

    lstm_autoencoder.fit(
        X_train_data, X_train_data,
        epochs=150,
        batch_size=64,
        validation_data=(X_val_data, X_val_data),
        shuffle=True
    )

    # Save and load the model
    lstm_autoencoder.save("lstm_autoencoder_model.keras")
    lstm_autoencoder = load_model("lstm_autoencoder_model.keras")
```

```
    # Reset the stream for real-time detection
    real_stock_prices = fetch_real_stock_data(ticker='NVDA', period='10y',
interval='1d')
    stream = (price for price in real_stock_prices)

    # Run real-time analysis with dynamic threshold
    real_time_plot_with_dynamic_threshold(stream, lstm_autoencoder,
time_steps=10, smoothing_window=60)
```

Output: