Email Logs Anomaly Detection - Report

This report presents a Machine Learning solution, built using the TensorFlow framework, to analyze and detect anomalies in email logs.

Autoencoders are a type of Neural Network, designed to learn efficient data encoding, by compressing input data into a lower-dimensional representation, and then reconstructing it.

In the context of anomaly detection, Autoencoders are relevant because they are (only) trained to minimize reconstruction error for normal data. So, when presented with anomalous data, the reconstruction error tends to be considerably higher, making it easy to identify outliers.

The primary goal of this exercise is to develop and deploy an Autoencoder-based model to monitor email logs and flag anomalies that could indicate malicious activity.

The solution is implemented using two Python scripts: modelTraining.py and modelInference.py

Script 1: Model Training (modelTraining.py)

- The first script, modelTraining.py, is responsible for Data Preprocessing, and Model Construction, Training, and Evaluation.
- **Data Preprocessing**
    - The script begins by loading the raw email log data from a CSV file, which includes the following fields: UUID, Recipient, Sender, Subject, and Time Processed.
    - The UUID field, being a unique identifier, is dropped, as it does not contribute to anomaly detection.
    - The remaining fields undergo several transformations.
    - Categorical features (Recipient, Sender, and Subject) are encoded, using the LabelEncoder method from the scikit-learn library, to convert string data into numerical format, suitable for model/tensor input.
    - The Time Processed field, initially of datetime format, is converted into a numerical representation, representing Unix time, to capture the temporal aspect of the data.
    - The features are then scaled, using the StandardScaler method, to normalize the data, ensuring that each feature contributes equally to the model.
- **Model Construction**
    - An Autoencoder model is then built using the TensorFlow and Keras frameworks.
    - The Autoencoder architecture consists of an input layer, three encoding layers with decreasing dimensions (64, 32, and 16 neurons, respectively), three symmetric decoding layers, and finally an output layer.

    - The encoding layers capture the most significant features of the model input, while the decoding layers attempt to reconstruct the original data from the encoded representation.
- **Model Training**
    - The preprocessed data is first split into independent training and testing data sets, with the training set used to train the Autoencoder.
    - The model is trained to minimize the reconstruction error (mean squared error), meaning it learns to compress and then accurately reconstruct normal (non-anomalous) data points.
    - The model's training performance is monitored using a Validation Split, to detect overfitting. After 50 epochs elapse in the training phase, the fine-tuned model architecture is saved as 'autoencoder_model.h5' for future inference purposes.
- **Model Evaluation**
    - To assess the model's overall performance, a plot is generated, relating the training and validation losses, across epochs.
    - Additionally, reconstruction errors on the testing data set are computed, and visualized using a histogram. This allows for the identification of an appropriate threshold for anomaly detection/classification, initially set at the 95th percentile of the reconstruction error distribution.
- **Saving Scalers and Encoders**
    - To ensure consistent data transformation, and preprocessing, during the inference phase, we save and export the (fitted) scaler and label encoders, using the joblib library.

Script 2: Model Inference (modelInference.py)

- The second script, modelInference.py, is designed to inherit and apply the trained Autoencoder model to new data, and detect anomalies.
- **Data Preprocessing**
    - Similar to the training phase, the script begins by loading and transforming new email log data.
    - In this phase, however, the UUIDs are first stored in a separate entity (for retrieval later in the process), before being deleted in the data transformation operation.
    - The data is preprocessed using the previously saved label encoder and scaler, ensuring that the data fed into the model is scaled and encoded consistently with the training set.
- **Anomaly Detection**
    - The preprocessed data is passed through the Autoencoder model to compute the reconstruction error for each data point.
    - Here, it is important to note that each data point corresponds to an individual log instance.
    - Anomalies are identified by comparing these errors to the pre-established threshold (which may be fine-tuned as per sensitivity requirements). Data points with reconstruction errors exceeding this threshold are then flagged as anomalies.

- **Result Mapping and Visualization**
    - First, the UUIDs of all anomalous email logs are retrieved/extracted and reported, providing a clear identification of all potentially harmful emails being exchanged.
    - We also generate a histogram of reconstruction errors with a threshold margin, as well as a t-SNE plot for visualizing the separation between normal and anomalous data points.
    - The t-SNE plot reduces the otherwise high-dimensional feature space into two dimensions, facilitating the visual inspection of the model's effectiveness in distinguishing between normal and anomalous data.

The presented solution provides a solid foundation for anomaly detection in network email logs using an Autoencoder model.

The two scripts work in tandem to preprocess data, train the model, and then detect anomalies in new data.

While the solution is effective, there are numerous opportunities for further enhancement, particularly in the areas of Feature Engineering (ex.- incorporating additional features like email body text, or attachment metadata), Model Architecture (ex.- experimenting with deeper or more complex neural networks to capture subtle anomalies), and Threshold Optimization (ex.- dynamically adjusting the anomaly detection threshold based on real-time data distribution).