ML Training Research

There are different questions to be asked proceeding regarding our system's ML training in Semester 2:

1. What algorithm/libraries do we use to train the model?
2. What data is the model being trained on?
3. How do we determine the accuracy of the trained model?
4. What information can a PCAP file provide and where are its limits?
5. What relevant data is eventually displayed to the end-user?

1. What algorithm/libraries do we use to train the model?

The simplest and fastest way to train our model is to use scikit-learn, a Python library that uses ML algorithms so that they don't have to be rewritten from scratch by us.

The most fitting algorithm for our system is IsolationForest, as it benefits from random future selection, trains on unknown data and flags abnormal ones.

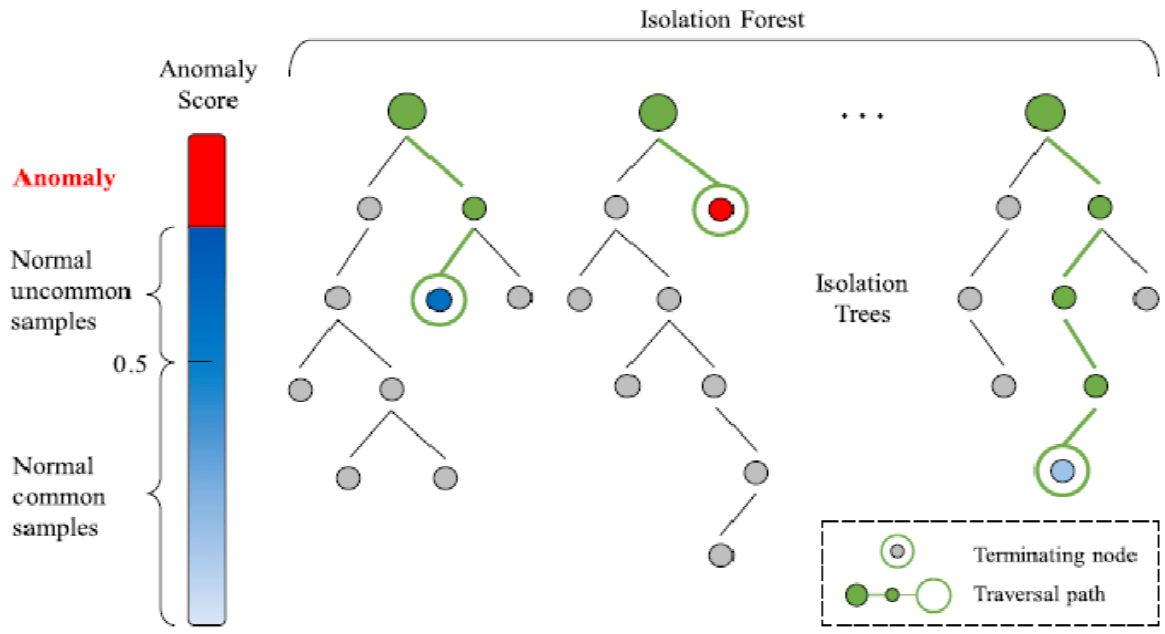

Image source: https://www.researchgate.net/figure/Anomaly-Detection-using-Isolation-Forest-18_fig3_350551253

How the algorithm works

1. Build many random trees

- at each node:
    - randomly choose a feature
    - randomly choose a split value
- split data
- repeat until:
    - only one point left, or
    - max depth reached

2. Count how many splits until isolation (path length)

    a. anomalities -> short paths
    b. normal points -> long paths

3. Compute mean path length across all trees

    a. normalize it
    b. convert to anomaly score
    c. short average path -> more anomalous

2. What data is the model being trained on?

Recording .pcap data is as simple as running a command (tcpdump) in your terminal, hence we could simply record around 20 minutes of normal browsing activity on one of our personal devices which our model could use to learn normal traffic patterns from.

3. How do we determine the accuracy of the trained model?

Since we get to record .pcap files on our own, we can also record how anomalous files would look like.

For example, this is the GitHub repository of a DoS attack simulator, which provides an in-built .pcap file generator as well: https://denizhalil.com/2023/11/29/networksherlock-port-scanning/.

If we record such a .pcap file and put it in our system, our model should recognise it as anomalous. Further attacks can be simulated, such as brute-force, beaconing, DNS tunneling and protocol misuse.

4. What information can a PCAP file provide?

- who is talking to whom -> unexpected hosts/destinations

- how much communication is being transmitted -> e.g. recognise beaconing & DoS detection
- protocol identification
- application & service fingerprints -> e.g. youtube vs netflix/vpn vs non-vpn traffic
- echos of user behavior through traffic shape, e.g. watching a video vs scrolling
- devices & OS e.g. desktop vs mobile/ios vs android

Future developers could combine our system by introducing supervised machine learning, allowing the model to classify between user behaviour through traffic shape, e.g. watching a video vs scrolling, and if desired, gain further insights, such as whether the connection is established via VPN, application

5. What relevant data is eventually displayed to the end-user?

The information taken from the PCAP files, as discussed in Section 4, can be transformed and displayed via diagrams into relevant and useful summaries, including, in addition to our already existing features in dashboard.py:

- z-score (how far a value deviates from the norm), calculated by `z = (value − mean_baseline) / std_baseline`, can be further used to explain why a flow was marked anomalous, e.g. high byte rate/long duration
- histograms/KDE curves that illustrate the separation between normal and anomalous traffic, where e.g. x: anomaly score, y: frequency
- timeline of suspicious activity, which allow to detect patterns, e.g. isolated spikes -> scanning activity, sustained elevation -> DoS/beaconing

Further improvisations

- record false positive rate on known normal PCAPs by `falsePositiveRate = flagged_normal_flows / total_normal_flows`
- same with detection rate on attack PCAPs by `detectionRate = detected_attack_flows / total_attack_flows`
- include rule flags (mark suspicious flows before even going through the model)
- combining the anomaly score by calculating the scores for IF, LOF, z and rule_flags
- allow sorting the most anomalous to least anomalous flows, helping to analyse data effectively