Project 3: Supervised vs Unsupervised Anomaly Detection

With shallow and deep learning methodologies

1 Objective

The goal of this laboratory is to apply **Shallow and Deep Learning anomaly detection** techniques in the framework of cybersecurity. Specifically, the lab evaluates how to create **Intrusion Detection Systems** (IDS).

To this end, students will analyze a dataset consisting of network traffic data. The traffic is labeled as either normal or one of several types of attacks (DoS, Probe, R2L). Each row in the dataset describes a connection and its associated features. We define a connection as the sequence of network packets between a source and a destination over a period of time. Each connection is characterized by several features, such as duration, protocol, number of bytes, number of failed login attempts, etc..

The goal of this lab is to apply both shallow and deep learning methods for anomaly detection and data representation to address the Intrusion Detection Systems (IDS) task. Throughout the lab, we **will not use the labels to directly train a supervised model** (i.e., we will not perform classification). Instead, the objective is to evaluate whether **unsupervised algorithms can automatically detect anomalous patterns** and to analyze how their performance evolves as our knowledge of the data increases (for example, knowing which are the normal and anomalous points).
Throughout the lab, students will:

- Learn possible strategies to **analyze datasets** composed of normal and anomalous traffic.
- Understand the **impact of different assumptions in the anomaly detection process**, e.g., knowing the class label or not.
- Experiment and compare **different anomaly detection methods**.
- **Use linear and non-linear data representation techniques** to i) visualize cybersecurity anomalies, ii) reduce the number of available features and iii) evaluate changes in anomaly detection performance.

2 Lab Rules

Each student must join a group. Groups are self-organized and consist of 3 students.
Each group must submit **four zip files**, one for each lab activity. The zip must contain:

- A report (maximum 10 pages) describing the approach, experiments, and results. The report can include tables and plots to support the analysis. Use the following naming convention: `groupID_labX.pdf` (replace `ID` with your group ID; `X` with the Lab number).
- The Jupyter notebook(s) and code (e.g., libraries with classes and functions written by you) to solve the tasks.

- **Best practice**: Add comments and headers (Markdown) sections to understand what you are doing. They will i) help you tomorrow to understand what you did today and ii) help us to interpret your solution correctly.
- **The notebook needs to be executed**: code and results must be visible so that we can interpret what you have done and what the results look like.
- **Must run**: the code must work if we need to run it again.
- **Submission format**: Include the notebook file (`.ipynb`) and an HTML export for easier review.

Use the following naming convention for the zip files: `groupID_labX.zip`. Also, notice that:

- You may submit the four reports up to 10 days before any exam session. The report submission date and the written exam date can differ – for example, you may take the written exam in January and submit the reports in February, and viceversa.

3 Dataset: Intrusion Detection System (IDS)

In this laboratory, you will use **connection-level logs from a network-based Intrusion Detection System (IDS)**.

Intrusion Detection Systems monitor and analyze network traffic to detect unauthorized or malicious behavior. They log metadata on each network connection, which is then used for offline training and evaluation of anomaly detection and classification algorithms. These connection records summarize communication sessions between two endpoints and include both statistical and behavioral features extracted from raw traffic.

You are provided with two datasets:

- `train.csv`: a curated set of labeled network connections used for training. This subset includes a balanced number of benign and malicious samples across several attack categories: Denial of Service (`dos`), Probing (`probe`), Remote-to-Local (`r2l`). In addition to the specific attack **label**, each label is labeled with a **binary label**. The binary label is `0` for `normal` connections and `1` for all types of attacks, i.e., anomalies.

- `test.csv`: a more heterogeneous set of network connections collected in varied conditions. You will use this dataset to assess whether the model you trained *generalize* to different settings.

Each connection record is described by **features**, divided into three main categories:

- **Basic features** describe attributes such as the connection duration, protocol type (e.g., TCP, UDP), service (e.g., HTTP, FTP), and status flags from the transport layer.

- **Content features** capture information from the payload content of the packets, including the number of failed login attempts, access to sensitive files, or commands executed in the session.

- **Traffic features** summarize network-level statistics such as the number of connections to the same host or service in a specific time window, helping to identify scanning and flooding behaviors.

The last two columns of each record are the attack **label** and the **binary label**. This structure enables the development of supervised, self-supervised and unsupervised learning models for intrusion detection.

Note: These datasets simulate real-world network activity and present realistic challenges such as class imbalance, redundant patterns, and evolving attack strategies. The goal is to design robust models.

4 Tasks

Students will go through a multi-step machine learning pipeline for anomaly detection:

- **Dataset characterization**: Examine the dataset to understand the number of categorical and numerical features. Check how the attack **labels** and **binary label** is distributed.
- **Shallow anomaly detection**: Use One-Class SVM in a supervised and unsupervised setting.
- **Deep anomaly detection and representation**: Use Autoencoder for anomaly detection and compare the ability to create a meaningful data representation comparing it with PCA.
- **Unsupervised detection and interpretation**: Use clustering results and visualization techniques to tackle a situation where anomaly detection does not have the label to learn the patterns.

Task 1: Dataset Characterization and Preprocessing

- **Explore the dataset**: Before preprocessing the data, explore the data to understand the available features.
  Q: What are the dataset characteristics? How many categorical and numerical attributes do you have?[1] How are your **attack labels** and **binary label** distributed?
- **Preprocessing**: Preprocess your features before performing any AI/ML algorithms.
  Q: How do you preprocess categorical and numerical data?
- **Study your data from a domain expert perspective**: When dealing with unsupervised learning, domain experts must frequently analyze data by hand. One way of doing this is to rely on heatmaps that describe the *statical characteristics* of each feature for each attack **label**.
  Plot and report the following 3 heatmaps:
  - **Mean heatmap**: `groupby` the data points for each attack label and extract the *mean* of each feature. Then plot and report the result as a heatmap.
  - **Standard Deviation heatmap**: group the data points for each attack **label** and extract the *standard deviation* of each feature. Then plot and report the result as a heatmap.
  - **Median Heatmap**: group the data points for each attack **label** and extract the **median** of each feature. Then plot and report the result as a heatmap.

[1]**Notice**: we consider 0/1 features as numerical.

Q: Looking at the different heatmaps, do you find any main characteristics that are strongly correlated with a specific attack?

Task 2: Shallow Anomaly Detection - Supervised vs Unsupervised

Start leveraging the OC-SVM in a *Supervised vs Unsupervised* for anomaly detection.

- **One-Class SVM with Normal data only**: First, train a One-Class Support Vector Machine (OC-SVM) with benign (normal) traffic only using an `rbf` kernel. Then, evaluate the performance using all training data (normal + anomalies).
  Q: Considering that you are currently training only on normal data, which is a good estimate for the parameter `nu`[2]? What is the impact on training performance? Try both your estimate and the default value of `nu`.

- **One-Class SVM with All data**: Now train the OC-SVM with both normal and anomalous data. Estimate `nu` as the ratio of anomalous data across the entire collection. Then, evaluate the performance using all training data (normal + anomalies).
  Q: Which model performs better? Why do you think that?

- **One-Class SVM with normal traffic and some anomalies**: Evaluate the impact of the percentage of anomalies while training the OC-SVM. Train several OC-SVMs with an increasing subsample of anomalous classes ([0%, 10%, 20%, 50%, 100%] of anomalies). Estimate the `nu` parameter for each scenario. Then, evaluate each model using all training data (normal + anomalies).
  Q: Plot the `f1-macro` score for each scenario. How does the increasing ratio of anomalies affect the results?

- **Robustness of the One-Class SVM model**: Finally, use the test set to assess the robustness of your model. Use models trained with only normal data (point 1); all data (point 2); and 10% of anomalous data (point 3).
  Q: Is the best-performing model in the training set also the best here? Does it confuse normal data with anomalies? Which attack is the most confused?

Task 3: Deep Anomaly Detection and Data Representation

Now, switch to Deep Learning algorithms. As you know, Deep Learning techniques can be directly used for Anomaly Detection or Data Representation only.

- **Training and Validating Autoencoder with Normal data only**: Create an Auto-Encoder architecture. Autonomously choose the implementation details: the only requirement is that the architecture must have a shrinking encoder and an expansion decoder, with a bottleneck in between. Use normal data only; split the normal data into training and validation sets, and use the validation set to pick the best number of epochs and learning rate.

- **Estimate the Reconstruction Error Threshold**: Once you trained the model, estimate a *reconstruction threshold*. The idea is that, if the model has a higher reconstruction error than the threshold, you define the point as an anomaly. To estimate this threshold,

[2]Remember: normal data **always** contain errors. 0 is **NOT** a good estimate.

calculate the reconstruction error in the validation set. Plot the *EDCF curve* of the reconstruction error on your validation data and estimate your reconstruction error threshold.
Q: How did you pick the threshold? What is its value?

- **Anomaly Detection with reconstruction error**: Now, use the trained model to compute the reconstruction errors for each point in the full training set (normal data + anomalies) and test set.
  Q: Plot and report the ECDF of the reconstruction errors for each point i) in the validation set; ii) in the full training set; iii) in the test set. Why the reconstruction errors higher on the full training set than on the validation one? And why are the reconstruction errors in the test set even higher? Use the threshold identified in the previous point to classify anomalies. How is the performance on the training, validation set and test set?

- **Auto-Encoder's bottleneck and OC-SVM**: Another way of using the auto-encoder is to leverage the encoder's bottleneck for data representation. From the model you previously trained, extract the encoder, and use it to extract the bottleneck embeddings of the normal data in the training set. Use these embeddings to train an OC-SVM. Then, extract the bottleneck embeddings of the test data and use the trained OC-SVM to predict normal data vs anomalies.
  Q: Compare the results with the best original OC-SVM and with the Autoencoder with reconstruction error. Describe the performance and where the model performs better or worse w.r.t. the original OC-SVM.

- **PCA and OC-SVM**: Another option for data representation is to use the Principal Components Analysis (PCA). Use the PCA analysis on the training set of normal data only to analyze the explained variance. Increase the number of PCA components, and find the *elbow point* with respect to the explained variance. Fit and transform the training set of normal data only using the estimated best number of components. Then, transform the test set with the same number of components. Finally, use the transformed training set to train an OC-SVM and use it on the transformed test set.
  Q: compare results with the original OC-SVM and the OC-SVM trained using the Encoder embeddings. Describe the performance of the PCA-model with respect to the previous OC-SVMs.

Task 4: Unsupervised Anomaly Detection and Interpretation

Finally, we will explore totally unsupervised clustering algorithms.

- **K-means with little domain knowledge**: As a domain expert, you may know the number of common attacks on your network, but not their actual attack label. Under this assumption, fit k-means with 4 clusters and the full training data (normal + anomalous).

- **K-means cluster interpretation**: After creating clusters that are completely unsupervised, we need to examine them to understand their quality.
  Q: How big are the clusters? How are the attack labels distributed across the clusters? Are the clusters *pure* (i.e., they consist of only one attack label)[3]?
  Q: How high is the silhouette per cluster? Is there any clusters with a lower silhouette

[3]Remember: this is something we can only measure in the lab assuming we know the labels. It is often NOT the case on the real world!

value? If it is the case, what attack labels are present in these clusters?
Q: Use the t-SNE algorithm to obtain a 2D visualization of your points. Plot and report: i) t-SNE using all training data and as label the cluster ID. To do this, try different values (max 3) of perplexity to determine the best visualization[4]. ii) Use the t-SNE with the best-looking perplexity and plot all training data with the **attack label**. Can you find a difference between the two visualizations? What are the misinterpreted points?

- **DB-Scan anomalies are anomalies?**: Finally, DB-Scan is a clustering algorithm designed to detect anomalous patterns that may represent anomalies. One way to estimate **min_points** is to evaluate the k-means result and look for the smallest cluster that consists only of normal data. This enables the definition of clusters with normal behavior. Set **min_points** according to this analysis. If you cannot find any pure cluster varying the number of cluster k, suggest and motivate your **min_points** choice. Next, use the elbow rule to determine the ϵ parameter based on the increasing distance between each point and its **min_points** neighbor.
  Q: Create the clustering results using the entire training set (normal + anomalous) using the parameters **min_points** and ϵ. Does the DB-Scan noise cluster (cluster -1) consist only of **anomalous** points (cross-reference with real attack labels)?

[4] Only from a visual point of view, See: `https://distill.pub/2016/misread-tsne/`