Exercise on Intrusion Detection Systems

The goal of this exercise is to code, using the Matlab environment[1], several anomaly-based Intrusion Detection Systems that use univariate and multivariate models.

- Import a reduced version of the `KDD99 HTTP` dataset in Matlab. The dataset is available on the exercise folder. The dataset contains 567498 measurements and each measurement has 3 features (`duration, src_bytes, dst_bytes`).
- Looking at the dataset labels, notice that only about 0.4% of the dataset are measurements that originate from an intrusion. Thus, we will choose to train an *unsupervised* IDS that only profiles the normal (non-intrusive) behavior.
- Below you can also find all the technical details about how the `KDD99` dataset was originally captured, including the full version of the dataset.

  https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

▶ In many scenarios

1 Univariate Intrusion Detection System using Normal distribution

▶ Write code that profiles the normal/regular behavior[2] by estimating the mean and standard deviation of a univariate normal distribution $\mathcal{N}(\mu, \sigma)$.

Choose the 2nd feature from the 3 available features to estimate the parameters μ and σ, i.e. the IDS that you will train is *univariate*.

▶ The dataset is quite imbalanced regarding normal (non-intrusive) behavior (99.6% of the dataset) and intrusive behavior (0.4% of the dataset). Thus a supervised IDS is not a good option and an *unsupervised* IDS must be used.

IDS training must be performed using only the normal (non-intrusive) behavior datapoints. Use 99% of the normal behavior datapoints to train the univariate IDS and keep the rest 1% for testing purposes.

For Sections 1 and 2 we assume that the `KDDCUP99` labeling is precise and that we can separate between intrusive and non-intrusive datapoints. Thus, our analysis will use an *untainted* dataset for unsupervised training, i.e. one that has not been contaminated with intrusive datapoints.

▶ After estimating the normal behavior as $\mathcal{N}(\mu, \sigma)$, set an adequate threshold th of your choice. Perform the testing using the intrusive behavior datapoints (0.4% of the dataset) and the remaining normal behavior datapoints (1% of the dataset). Compute the true positive TP, true negative TN, false positive FP and false negative FN rates for your chosen threshold th.

To train and test the IDS, you can modify the code in the `normal_ids.m`, found in the `help_code` folder to use only a single class of normal (non-intrusive) behavior datapoints.

[1] check https://datanose.nl/#byod to use the UvA Matlab license

[2] The term 'normal behavior' in this context can be confusing. 'Normal' here means 'regular' or 'ordinary' behavior and does not necessarily imply a normally distributed dataset. In this specific IDS however, we assume that the dataset describing the 'normal behavior' is also normally distributed. Naturally, this distribution assumption can be different in various IDSs.

- By varying the threshold th, construct the ROC curve for the IDS.

  To construct the ROC you can directly use the code found in `create_roc.m`, found in the `help_code` folder. Notice the sorting trick to avoid creating unnecessary ROC points (see also "An introduction to ROC analysis" by T. Fawcett, Algorithm 1).

- Use the `trapz` function (or any other integration function), to compute the Area-Under-Curve (AUC) for the constructed ROC curve. Notice that the $AUC \approx 1$, showing that in this dataset a threshold can clearly separate intrusive and non-intrusive datapoints.

- Use the ROC to pinpoint a threshold th that yields good results for the (TP, FP) pair.

2 Multivariate Intrusion Detection System using Multivariate Normal Distribution

- Write code that profiles the normal (non-intrusive) behavior of the dataset by estimating the mean vector and covariance matrix of the 3-dimensional normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

- After modeling the normal (non-intrusive) behavior, compute the anomaly scores for all datapoints in the normal and in the intrusive testsets. In particular compute 2 score metrics:

  1. The likelihood score, based on the multivariate normal probability density function, i.e. using `mvnpdf`.
  2. The simplified log-likelihood score, while assuming that the covariance matrix does not affect the outcome (homoscedasticity). I.e. use $score(\mathbf{x}) = (\mathbf{x} - \mu) * (\mathbf{x} - \mu)^\intercal$, where $\mathbf{x}$ is a datapoint and μ is the mean vector, with dimensions $dim(\mathbf{x}) = dim(\mu) = 1 \times 3$.

- For the 2 score metrics (likelihood, log-likelihood), select adequate thresholds and compute the respective TP and FP rates.

  Notice that in the case of the likelihood score, higher values imply a better match, since the metric relates directly to a distribution. However, in the case of the log-likelihood score, lower values imply a better match, since the metric relates to the distance between the datapoint $\mathbf{x}$ and the mean vector μ.

3 Intrusion Detection Libraries

- Read the Matlab documentation for using the Local Outlier Factor (LOF) library and experiment with the provided example on Outlier Detection.
  https://nl.mathworks.com/help/stats/lof.html

- Unlike Sections 1 and 2, we now assume that the `KDDCUP99` labeling is not precise and we can no longer separate intrusive and non-intrusive datapoints. Thus, our analysis will use a *tainted* (also known as *contaminated*) dataset for unsupervised training. As analysts, we only know that approximately 0.4% of the dataset amounts to intrusions.

- Use the `lof` function to perform intrusion detection on the entire dataset (i.e. without splitting in trainset and testset). Compute the anomaly scores for all datapoints.

  Make sure to set the proper `ContaminationFraction` parameter.

- Use a log-scaled histogram to visualize the anomaly scores (see again the Outlier Detection example in the Matlab documentation). Use this histogram to select a reasonable threshold.

- Find the `KDDCUP99` datapoints whose LOF scores exceed the set threshold, thus are anomalies. Compare the LOF-detected anomalies to the ground truth, i.e. to the true labels. To perform the comparison, use the `setdiff` function to count the number of true outliers that are not detected by LOF.

- Read the Matlab documentation for using the iForest library and experiment with the provided examples on Outlier Detection.
  https://nl.mathworks.com/help/stats/isolationforest.html
- Split now the full `KDDCUP99` dataset in to parts (50-50 split, each part with 283749 3-dimensional datapoints).
- Use the first part to train an isolation forest model, using the `iforest` function, producing an iforest object.
- Apply the trained model to find anomalies on the second part of the dataset, using the `isanomaly` function.