Didrik Frimann Barroso Koren

Improving security visibility with event correlation

Master's thesis in Communication Technology and Digital Security
Supervisor: Yuming Jiang
Co-supervisor: Geir Solskinnsbakk
March 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Didrik Frimann Barroso Koren

Improving security visibility with event correlation

Master's thesis in Communication Technology and Digital Security
Supervisor: Yuming Jiang
Co-supervisor: Geir Solskinnsbakk
March 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title: Improving security visibility with event correlation

Student: Didrik Frimann Barroso Koren

Problem description:

When handling large amounts of security data, some are directly actionable, while others need further processing or contextualization before becoming actionable. Today, the data that is not directly actionable is commonly used by security analysts to clarify incidents arising from the actionable data. Any automation of this processing would likely reduce the workload for the security analysts, by delivering a feed of data that they could likely act upon.

This project will investigate data science methods for producing a feed of actionable data from unclassified data in Sikt's NIDS. Auxiliary data sources used in real analysis of alerts will first be identified. They will be studied to provide additional features for the unclassified NIDS data, which in turn will be engineered and normalized. This feed will then be used to build classifiers, exploiting data analytic methods like machine learning, deep learning, and statistical inference, for the unlabeled alerts. Finally, the classifiers will be evaluated according to their ability to recall actionable data without introducing a high degree of false positives.

Date approved: 2021-10-05

Supervisor: Yuming Jiang, IIK

Cosupervisor: Geir Solskinnsbakk, Sikt

Abstract

Cyber analysts are the keystone of modern cybersecurity, analyzing events and alerts, securing networks, computers, and servers, and responding to incidents. Nevertheless, analysts are not an infinite source, and as with most others, their time is limited. In addition, the number of alerts likely increases over time. Automating one of the most critical parts of their job, the alert analysis, would free up time for work on additional security. This thesis presents a method for sorting out most of the alerts, leaving the outliers for manual evaluation, using Principal Component Analysis (PCA), Autoencoding Neural Network, K-means clustering, and time series. This might reduce the strain of alert analysis on the analysts, counteracting fatigue from loads of alerts, and freeing time for additional and better security. Our method reduced the alert dataset to one-fifth of its original size. Comparison with a small set of manually classified alerts, evaluated by analysts at Sikt, shows promising results in keeping the interesting alerts in the reduced dataset.

Sammendrag

Sikkerhetsanalytikere er hjørnesteinen i moderne informasjonssikkerhet hvor de analyserer logger og alarmer, sikrer nettverk, datamaskiner og tjenere, og håndterer informasjonssikkerhetshendelser som oppstår. Analytikere er ikke en utømmelig kilde, og som andre ressurser så har de begrenset med tid. I tillegg øker sannsynligvis antall alarmer over tid. Å automatisere en av de mest, om ikke den mest kritiske, delen av jobben deres, å analysere alarmer, kan frigjøre tid for å lage enda sikrere systemer. Denne masteroppgaven presenterer en fremgangsmåte for å sortere ut de fleste alarmene, slik at bare de avvikende alarmene blir igjen til analytikerne. Fremgangsmåten baserer seg på PCA, selvkodende neurale nett, klustering med K-means, og tidsserier. Dette vil kanskje redusere lasten som analyse av alarmer kan være for analytikerne, og at dette frigjør tid så analytikerne kan jobbe mot enda mer og bedre informasjonssikkerhet. Fremgangsmåten vår reduserte datasettet med alarmer til en femtedel av opprinnelig størrelse. Da vi sammenlignet dette datasettet med et lite datasett som analytikere hos Sikt hadde analysert for oss, viste dette lovende resultater. De fleste alarmene som de hadde markert som interresante forble i det reduserte datasettet.

Preface

This thesis might be my product, but it builds directly and indirectly on contributions from many people, whom I, in varying degrees, are indebted. My professor Yuming Jiang (IIK, NTNU), and supervisor, Geir Solskinnsbakk (Sikt), have been very patient with me while still pushing me to finish this thesis. Their insights have been invaluable to this thesis, and I am very thankful for the collaboration. All my colleagues at Sikt have been very understanding and helpful with this thesis, and I am very grateful for it. I still need to thank some of them in particular. Thanks to Emil Henry Flakk (Sikt) for great discussions and insights throughout the thesis, and Christoffer Evjen Ottesen (Prev: Sikt) and Rune Sydskjør (Sikt) for help with manually classifying alerts.

For the past five and a half years, I have been volunteering at Studentersamfundet i Trondhjem (the Student Society in Trondheim), and all the people I have met there have been vital in making my time studying in Trondheim great and memorable years. Finally, I have to thank my family, my parents in particular, for being there for me and helping motivate me.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

EVE Extensible Event Format.

HIDS Host Intrusion Detection System.

IDS Intrusion Detection System.

IPFIX IP Flow Information Export.

IPS Intrusion Protection System.

IQR Interquartile Range.

MAD median of all absolute deviations from the median.

NIDS Network Intrusion Detection System.

OISF Open Information Security Foundation.

PCA Principal Component Analysis.

SNMP Simple Network Management Protocol.

SVM Support vector machine.

Chapter 1
Introduction

Cyber security analysts struggle to catch up with an increasing amount of alerts and security events. When sampling a random day during Spring 2021 from the dataset studied by this thesis, more than 71 000 alerts were observed. Even with an optimistic estimate of one alert processed per minute, an analyst will only cover roughly 450 alerts. An additional challenge is that the triage process is not recorded in a structured way, leaving us without any labels to work with.

Sikt is not the only one who noticed how the alert load received could affect the receivers. Lin et al.[LCC+18] describe it as "alert fatigue" when referencing a 2014 report from the Cybersecurity firm FireEye[Fir14] that claims more than 50% of all organizations received more than 17 000 alerts a week, in 2014, with more than 51% of those alerts being false positives.

This thesis is done in collaboration with the Cybersecurity Centre at Sikt. Sikt is an agency of the Norwegian government under the Ministry of Education and Research.

1.1 Problem description

In this thesis, our goal is to produce a curated feed of alerts, primarily containing the outliers. We choose to look at the outliers because the cybersecurity analysts at Sikt have claimed that most alerts are irrelevant. An alert can be irrelevant because it is unactionable, so the analysts cannot do anything with it regardless of whether the alert is a true or false positive. While for the remaining alerts, most are false positives. We, therefore, assume that outlier alerts are more attractive than non-outliers.

The problem description also includes some sub-goals that should help with the progress of the primary goal. Finding the auxiliary data sources that the analysts typically use in their analysis could make the context available to the automation. This also applies to some normalizations and feature derivations

created based on their domain knowledge.

We look at three different families of data science methods for this thesis because they are exciting topics discussed by the media and many cybersecurity professionals. The analysts at Sikt already treat the alerts as time series because their usual analysis tool, Humio, by default, has a plot of the alerts over time. Machine learning and deep learning were technologies that potential vendors have often presented as the be-all and end-all of cybersecurity, often without details about how it is implemented to be helpful. Piqued by their often vague claims, we settled for trying those technologies.

1.2 Contributions

This thesis contributes a novel method for detecting abnormal Network Intrusion Detection System (NIDS) alerts through outlier detection across multiple algorithms. We use statistical methods on time series to detect the outliers, and PCA and Deep Learning compress the alerts before clustering them and looking for outliers in the clustered data. This method is not limited to the algorithms used in this thesis and can be expanded with other outlier detections if so desired. The compressed alerts from PCA and the autoencoder can serve as a basis for other outlier detections or classifications. The identified auxiliary datatypes can be used for more comprehensive analyses or extended context.

1.3 Outline

This thesis is divided into eight chapters, with chapter 1 being this introduction. The theory is covered in chapter 2, Related works in chapter 3, and the data and preprocessing in chapter 4. These chapters cover the preparations necessary for this thesis. Then we will cover the methods used in chapter 5. The results are presented and analyzed in chapter 6. The discussion and future works are covered in chapter 7. Finally, there is chapter 8, which contains this thesis's conclusion.

Chapter 2
Background Theories

The following sections cover the necessary theory for this thesis. First, we look into the general idea of Network Intrusion Detection Systems in section 2.1, then the specifics of Suricata NIDS in section 2.2. Some general theory on machine learning is covered in section 2.3, while the theory of autoencoders, within neural networks, and Principal Component Analysis is covered in section 2.4 and section 2.5, respectively. The clustering theory is in section 2.6, and the time series is in section 2.7. Finally, we look at some outlier detection for the evaluation in section 2.8.

2.1 Network Intrusion Detection System

An Intrusion Detection System (IDS) is used to identify and warn about suspicious activity in the monitored domain. For cybersecurity, two monitoring types are the most common, host-based and network-based. Host-based intrusion detection systems (HIDS), which are not a part of this thesis, are also known as endpoint detection systems and, as such, need to be installed on the device to monitor it. Network-based intrusion detection systems, or just Network Intrusion Detection System (NIDS), can be deployed at central locations in a network and thus monitor all traffic to and through that network location. For instance, the central locations can be the gateway, between network segments, or at the core router.

As covered by the taxonomy from Hindy et al. [HBB+18], there are multiple designs for an IDS, especially around the detections. Branch 4 of the taxonomy covers these divisions as signature-based and anomaly-based detection, including a possible hybrid. Anomaly-based IDSes are further divided into how the anomaly is detected through statistics, machine learning, or finite state machines.

IDSes can be designed with varying degrees of distribution, and different parts of the detection can be distributed. Hindy et al. [HBB+18] cover these in branch 2, which divides an IDS into different groups based on its Decision-Making (2.1), Infrastructure (2.2), and the Computation Location (2.3).

2.2 Suricata

Suricata is an open-source Network Intrusion Detection System (NIDS) that is developed by the Open Information Security Foundation (OISF)[1]. Suricata is a signature-based IDS, which primarily creates events when a signature is detected. All these events have the type of "Alert". Suricata servers work independently, but the resulting events can be sent to a centralized data store. The resulting alerts are deterministic when detecting the same traffic using the same rule set on different but similarly configured servers.

Suricata has the concept of rules which can detect 0 or more signatures. The rules can be one of two types, flow-based or packet-based. For flow-based rules, such as those looking at HTTP traffic, the rules can be written to detect signatures across multiple packets. The Suricata engine handles gathering, deduplicating, and normalizing the individual packets into a flow. Packet-based rules only look at a single packet at a time, such as a single UDP packet. In addition to detection based on signatures, Suricata can mark a host, a pair of hosts, or a flow, with a "bit". Hosts are marked using hostbits or Xbits, while flows are marked using flowbits. These "bits" can then be checked in place of a signature. These methods can also be combined so that a bit is sat or checked together with the signature. For instance, having one rule to detect Windows Updates, and marking those flows as Windows Update, can allow other rules that detect the download of Windows executables to ignore that traffic.

Suricata can also create other event types, such as logging all DNS traffic. The different protocols can be enabled in the configuration and work similarly to creating a rule that matches all the traffic of the enabled protocols. Enabling logging of different protocols can make Suricata act as part of a collaborative IDS, where it can send the events to a centralized location for further detection or processing, such as through Anomaly-based detection. Another configuration of Suricata can make it a Network Intrusion Protection System, where it works as a deep-packet inspecting firewall based on the rules it has.

2.2.1 Rule format

Suricata rules consist of three main parts, the action, the header, and the options. The action is what Suricata should do when the rule is matched. Only "pass" and "alert" are usable actions when running detection mode. Five other actions are available when running Suricata in inline mode as an intrusion protection system, which drops, rejects silently, and rejects with a warning to either source, destination, or both.

[1] https://oisf.net/

The header of the rule contains the protocol, source, destination IP and port, and the direction of the traffic. Suricata checks that the traffic is of the selected protocol, including when used by higher-level protocols. For instance, if Suricata looks at HTTP traffic, it can trigger an alert on a TCP rule but not a UDP rule. The source and destination's IPs and ports can, in addition to a single IP or Port, be a set of IPs or ports, or "any" to accept all legal values. The direction is either "->" or "<>", used to indicate whether the Suricata should accept both orders of the source-destination pair.

The Rule options contain keywords and keyword-value pairs, ending with a semicolon. There are multiple sets of keywords for each of the supported protocols, from IP, TCP, UDP, and ICMP, to higher tier protocols like HTTP, DNS, SIP, and MQTT. The flow- and hostbits that were covered earlier are also considered keywords. The meta keywords describe the rule, such as the name, rule id, revision, references, and "metadata", a nested key-value list, separated by commas, used to include free form metadata about the rules. There are attempts to standardize the schema of the metadata keyword, such as "BETTER schema"[2].

An example rule is shown in Listing 2.1. This rule was made to generate an illustrative alert on a pcap created with scapy-pcap[3]. The words in light pink in the listing are keywords. The first one is the action, "alert", for this rule. This indicates how Suricata should handle traffic matching the rule. This is useful as Suricata can be used as an Intrusion Protection System (IPS), where this is the option that can block traffic.

Next is the header of the rule. Here the protocol is placed, "tcp" for this rule, which is used to tell Suricata what protocol to check with this rule, so this rule will not, for instance, be used to check a UDP packet. The other content of the header is "1.1.1.11 any ->1.1.1.14 any", which tells Suricata that this rule should only apply to traffic from 1.1.1.11 to 1.1.1.14 at any source and destination port.

The final part of the rule, the rule options, is also the largest, that is, everything within the parenthesis in the listing. The options can vary depending on the protocol used in the header, so a sample of typical options is used here. First, we have "msg", the name of the rule, used to give a quick way for humans to read what the alert is attempting to detect. Then there is "content", which looks for the value given in the payload. In this case, "Alqghv", a random string from the PCAP, is looked for in the payloads. "flow" is an option to limit noise in that "established" and "to_client" are requirements for the flows that this rule should alert on. "established" asks for a flow after the three-way handshake, and "to_client" asks for traffic only from the server. Then there are some meta keywords, "reference" is a way to show sources of the rule, "metadata" is a key-

[2] https://better-schema.readthedocs.io/en/latest/
[3] https://github.com/vnetman/scapy-pcap

value list for extra info. "classtype" is referencing a class from a classification file that Suricata uses to prioritize alerts, which is configurable by the users of Suricata. "sid" is the rule id, a unique numerical identifier. Furthermore, "rev" is the revision number used to optimize a lot of rule processing, as most tooling ignores already seen rules if the revision is the same.

```
alert tcp 1.1.1.11 any -> 1.1.1.14 any (msg: "alert"; \
  content: "Alqghv"; flow:established,to_client; \
  reference:url,example.com; metadata:author dfkoren; \
  classtype:string-detect; sid:2; rev:1;)
```

Listing 2.1: Example of a Suricata rule, created to help illustrate how the rules work and to generate an alert from a generated PCAP.

2.2.2 Alert format

Suricata output, including alerts, are available in a couple of different formats, but we will only cover the Extensible Event Format (EVE) JSON output here, as that is the one used in the source data of this thesis. As with the Suricata rules, which allow for protocol-specific keywords, the EVE JSON alerts contain different data based on the rules' protocol. The common schema of the output contains a timestamp, event_type, and any information about the related PCAPs if Suricata is generating them. Then there is the alert section which contains information about which rule triggered, including the metadata of said rule. The IP-port-pair of the communicating parties that triggered the alert is stored. The information about the flow is stored in a flow section, which covers sizes, packet counts, and start time of the flow. The output greatly depends on how the Suricata instance is configured, but a sample is shown in Listing 2.2.

```
{
    "timestamp": "2022-02-10T11:00:20.013333+0100",
    "flow_id": 1122135368007680,
    "event_type": "alert",
    "src_ip": "1.1.1.11",
    "src_port": 80,
    "dest_ip": "1.1.1.14",
    "dest_port": 1050,
    "proto": "TCP",
    "alert": {
        "action": "allowed",
        "gid": 1,
        "signature_id": 2,
        "rev": 1,
        "signature": "alert",
        "category": "A suspicious string was detected",
```

```
        "severity": 3,
        "metadata": {
            "author": [
                "dfkoren"
            ]
        }
    },
    "flow": {
        "pkts_toserver": 54,
        "pkts_toclient": 54,
        "bytes_toserver": 2916,
        "bytes_toclient": 74254,
        "start": "2022-02-10T11:00:00.000000+0100"
    }
}
```


Listing 2.2: An example alert from the rule in Listing 2.1. This alert was generated to illustrate how the Suricata alert usually looks.

2.3 Machine learning

Machine learning is the concept of algorithms that summarize, or learn from, the input data without having a developer specify how. This is also known as training the algorithm using experience [GBC16, Chapter 5.1.3]. The trained algorithm can then be used to make predictions on unseen data. This new input is often expected to have not been part of the training. For instance, the algorithm is trained on an existing data set to create the clusters, then used to cluster new data. These clusters may also uncover some previously unknown structures in the data. New but similar data (i.e., in the same format) can then be mapped into one of the clusters [LRU20].

2.3.1 Supervised learning

The class of supervised machine learning algorithms depends on a set of data that contains the desired output, such as the classification of the data. An example is the MNIST dataset [LBBH98] which contains handwritten digits with labels of the correct digit. Here the algorithm can try to classify the image it sees and then check and correct with the answer provided by the dataset.

An advantage of supervised learning is that the labels can be used to calculate performance metrics, such as precision and recall[GBC16, Chapter 11], without gathering new data or labeling new data (as it should already be labeled). Labels are also the main disadvantage, unless the data gathering includes the label

natively in some way, getting the data labeled is usually an expensive, time-consuming, and possibly biased process[BC19].

2.3.2 Unsupervised learning

It is unsupervised learning when a machine learning algorithm can be trained without labels. Unsupervised machine learning often assumes that the dataset has some random distribution from which each data point is sampled. Enough samples will then give the data's probability distribution, predicting the output based on the input.

An advantage of using an unsupervised machine learning algorithm is that the data can be used without regard for labels, allowing more datasets to be used. There are still advantages to using labeled data, as the labels can help validate the results. On the other hand, unlabeled data is more of a blind approach without the possibility to validate the results through labels[GBC16, Chapter 5.1.3].

2.4 Neural Network

When training a neural network, the data is passed from the input neurons, through the network, to the output neurons. The output is then compared to the target, and the error is passed backward through the network. This is called backpropagation. The training process is divided into epochs, where each epoch goes through the whole dataset once. The epochs can be subdivided into smaller batches, a subset of the data that gets bundled together, and then the cumulative error is passed back through the network.

The data is usually split into multiple sets, training and test sets. If, after training, the error is still high for both the training set and the test set, then the model is underfitting. This means that the network cannot represent the whole dataset, and the network would need to be redesigned, such as increasing the number of neurons in the thinnest layer to fit all the data. On the other hand, if the training set error is low, while the test set error is still high, the model is overfitting. Fixing overfitting is either done by reducing the training time or through regularization.

Depending on how the neural network is initialized, adding a regularization layer, such as a dropout layer, which randomly sets values from the previous layer to zero, reduces the overfitting of the model. This can also help remove symmetries from networks if all initial values are the same.

2.4.1 Autoencoder

Autoencoder is a method for getting a neural network to train on recreating its inputs. The algorithm can, for instance, reduce the number of dimensions used to represent the data. Autoencoders are often based on neural networks, a part of the machine learning subarea of deep learning. The autoencoder can be considered as two different parts. The first part compresses the input data down to the desired size and is trained to do this better and better. The second part trains to reconstruct the original data from the compressed representation. This has two advantages which this thesis will use, it does not require any labels for the data, and the accuracy of the size reduction can be shown through the end-to-end error.[GBC16, Chapter 14]

Figure 2.1 illustrates how an autoencoder looks when created with a neural network. The main components that should be noted are the boxes, the neurons, which are organized as layers, being fully interconnected between each layer. The input layer directly maps to the input data and only includes a score that represents the corresponding input. In the remaining layers, the values are generated from the inputs, the outputs from the previous layer, scaled by weights. The weights are changed through training the neural network.

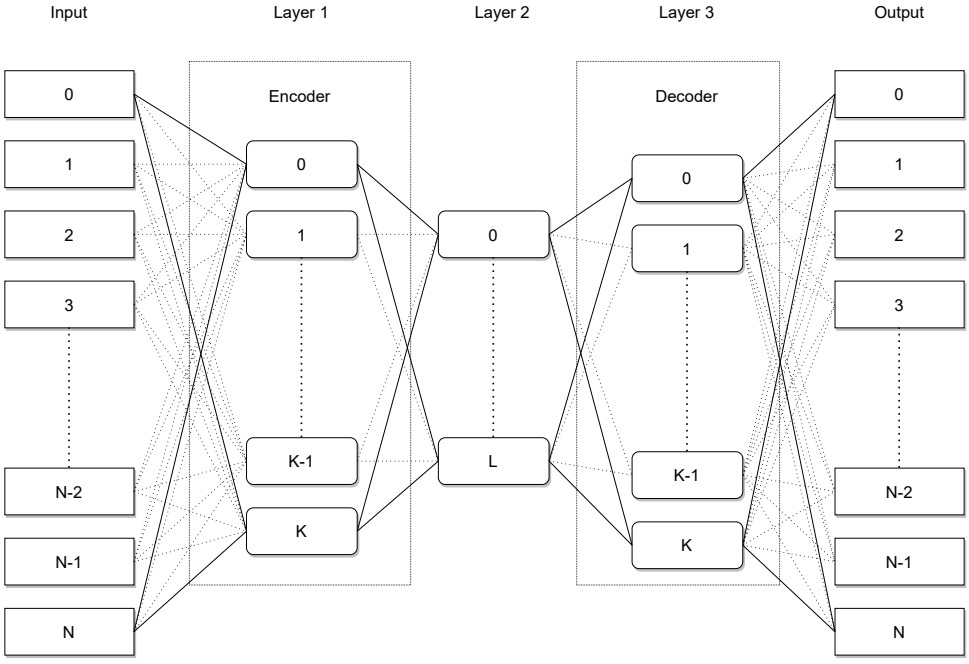

Figure 2.1: This is an illustration of how an autoencoder neural network can be structured.

To evaluate the autoencoder performance, one can use standard error algorithms to measure how far the autoencoder's prediction is from the original data, such as Mean Squared Error, Mean Absolute Error, Binary Crossentropy, or R squared.

2.5 PCA - Principal Component Analysis

Principal Component Analysis (PCA) is a technique for reducing the dimensionality of data vectors by finding their orthonormal axes that keep most of the data's variance when having the data projected upon [Hot33]. Tipping and Bishop [TB99] have a fuller introduction to PCA, which the reader is referred to for additional information.

The number of principal components is an essential aspect of the PCA. This thesis uses the scikit-learn[PVG+12] implementation of PCA, which has three modes of selecting the number of components to use. The default is to use either the sample count or the feature count of the dataset; the smaller is selected. The second way is to specify the desired number of components, and the third is to select the amount of variance that should be explained. The explained variance is the sum of the variance explained by each component.

2.6 Clustering

The data is often not distributed equally throughout the domain in a large dataset. There might be areas of the domain with higher concentrations of data points. Grouping these data points might identify some shared features across them. This process is called clustering.

The dataset is partitioned into smaller segments through some partitioning scheme. This can be decided by the proximity to its neighbors, the proximity to a cluster center (centroids), or another metric. If the clustering is iterative, the clusters are updated after having the data points assigned to them. For instance, the cluster centroid depends on which data points are members of the cluster, and if the centroid is moved, the members might have changed.

A clustered dataset has a couple of metrics that can evaluate the clustering, including whether the number of clusters is correct. Each cluster can also be represented by its centroid. Each point is assigned to one cluster, and it is possible to find the distance from the point to its and the other clusters' centroids. This gives a measure of the centrality of the point in its cluster. If the variation of the distances to the centroid in a single or a few clusters is high, that might indicate that there should have been at least one more cluster.

There are many different families of clustering algorithms, such as Hierarchical, Graph Theory-based, and Neural Networks-based, which use different methodologies for splitting the data set into clusters. The hierarchical algorithms use a tree, while the other two examples use Graph Theory and Neural Networks-based algorithms, respectively[XW05]. Rokach and Maimon[RM05] present a more comprehensive overview of data clustering.

2.6.1 K-Means

K-Means is a Squared Error-based clustering algorithm for partitioning the data into K clusters[XW05]. It consists of three steps, where the last two are repeated until the clustering is stable, i.e., the clustering has converged. K points are generated as the initial cluster centroids in the first step. This can be random points in the domain or existing points from the data. Then the other points are assigned to the closest cluster centroid, which is the second step. The third step is to move each cluster centroid to the mean of the cluster members. If the cluster centroids have changed, go back to step two.

When finding the correct number of clusters for a clustering, such as K-Means, one can use the Elbow method and the Silhouette score. The Elbow method looks at how much of the variance is explained for each number of clusters to identify at which point the return of adding another cluster has diminished so much that it is not worth it anymore[Tho53]. The elbow method can also be used with the inertia of the clustering, which in Scikit-Learn is "[the s]um of squared distances of samples to their closest cluster center[...]"[PVG+12]. Silhouette score is the mean of the silhouette coefficients of each sample. Silhouette coefficients are calculated for each sample by taking the mean distance of the closest cluster it is not a member of, subtracting the mean distance to the sample's fellow cluster members, then dividing it all on the larger of the two[Rou87].

2.7 Time series

When data points have a timestamp, they can be oriented as a time series [BD16, Chapter 1]. A time series has one variable that varies over time in its simplest form. Multiple time series can be put together to see multiple variables in context. If the multiple variables describe different aspects of the same observation, it is a multivariate time series. An example of this is weather observations, where a time series can include one variable for temperature, one for wind speed, and one for rain.

Based on the observations, it is possible to find the seasonality of the data. To use a possibly familiar example, the temperature for the first day of the year, January 1st, is likely more similar to the mean temperature for January 1st, or the mean temperature for days in January, than the average day in a

year. This is at least true in places far from the equator. This property makes it possible to have better estimates for the temperature based on the time it should happen. If the observations see an increase in their observed values, which is not seasonal or still exists after correcting for seasonality, one likely has a trend in the observations[Gut20, Chapter 6.4.4.3].

Another property that a time series can have is stationarity, in which the variance and auto-correlation are constant, and there is no trend or seasonality [Gut20, Chapter 6.4.4.2].

The weather observation time series described above is simple in that its values are numbers, and it is something many people have experienced. Another aspect of time series is metadata, data about the measurements. Going back to the example, this time series is observed at a given observation station, usually in a city or near an airport. The location of the measurement can be used to look for trends that are independent of local variations. For two observations from neighboring towns, their temperature is likely similar for the most part.

2.7.1 Windowing

One way to look at time series from a historical perspective is to use windowing. Here the last N periods are used as the baseline on which aggregation is applied—for instance, looking at a time window, the mean and standard deviation can be calculated for that window. This can be used to forecast the next period by assuming the mean value and standard deviation will estimate the next period.

Windowing functions are also known by other names, including moving and rolling, plus the applied function—for example, the moving or rolling average [Gut20, Chapter 6.4.2]. A moving average can get a smoother estimation based on the assertion that temporally similar values are more similar than values from distant points in time. To keep on the meteorological theme, tomorrow's temperature is likely closer to the mean of the last three days than some months ago.

2.8 Outlier detection

Outlier detection is a method for classifying objects as outside the normal scope and expected dataset distribution. Outliers can be detected in both univariate and multivariate data[HdV08].

2.8.1 Standard Deviation

One detection method introduced in most basic statistic courses is the standard deviation, in which the number of standard deviations a data point is away from

the mean is used to consider whether it is an outlier or not. The equation of the outlier detection is shown in Equation 2.1, where $\bar{X}$ is the sample mean, s is the sample standard deviation, and α is a parameter used to set the outlier limit. This method is usually used on normally distributed data, in which the limit, α, commonly is selected to be ± 3, ± 2.5, or ± 2, which respectively gives 0.13%, 0.62%, and 2.28% outliers[YRF19].

$$T_{threshold} = \bar{x} \pm \alpha * s \tag{2.1}$$

One major disadvantage of using Standard Deviations to detect outliers is that neither function used, the sample mean and sample standard deviation, is resilient against any vast outlier. As covered by [RH11], the sample mean can only resist outliers of $\frac{1}{n}$, which for a sufficiently vast outlier leads to the sample mean being skewed by a single outlier. The same applies to sample standard deviation; one sufficiently vast outlier can skew the measure.

2.8.2 Z-score

Z-score is another measure that is used to detect outliers. Z-scores are relative as opposed to Standard Deviation outlier detection, so two datasets with different sample means and sample standard deviations can use the same Z-score as their threshold, with 2.5 being typical thresholds [RH11]. The Z-score equation is presented in Equation 2.2, where $\bar{X}$ is the sample mean of x and s is the sample standard deviation.

$$z_i = \frac{x_i - \bar{x}}{s} \tag{2.2}$$

As with the Standard Deviation based outlier detection presented in subsection 2.8.1, the sample mean and sample standard deviation are prone to vast outliers skewing the values.

2.8.3 Robust Z-score

To counteract the low resistance to vast outliers in Equation 2.2 Robust Z-score is presented by [RH11] as a replacement that uses the more resistant function median and median of all absolute deviations from the median (MAD). Equation 2.3 shows the robust version of the Z-score, with $\tilde{x}_n$ being the median. MAD is often corrected by multiplying it with 1.483 to remove the bias on the normal distribution. This is ignored here as the dataset it is used on is not a normal distribution, and nor are we comparing the Z-scores between different distributions.

$$z_i = \frac{x - \tilde{x}_n}{mad(x_n)} \tag{2.3}$$

The use of median and MAD in place of the sample mean and sample standard deviation makes this version of Z-score robust against up to 50% outliers.

2.8.4 IQR

Interquartile Range (IQR) is another method for detecting outliers in a distribution. The IQR is first calculated in Equation 2.4 by taking the 75th percentile and subtracting the 25th percentile. The thresholds for detecting outliers are shown in Equation 2.5, where c is used to set the outlier limit[YRF19].

$$iqr = x_{\lceil 3n/4 \rceil} - x_{\lfloor n/4 \rfloor} \tag{2.4}$$

$$T_{min} = x_{\lfloor n/4 \rfloor} - c * iqr \tag{2.5a}$$

$$T_{max} = x_{\lceil 3n/4 \rceil} + c * iqr \tag{2.5b}$$

[RH11] also covers the IQR regarding its resistance against impact from outliers. Due to using the 25th and 75th percentile as its basis, if there are 25% outliers, those percentiles can be an outlier.

Chapter 3
Related works

There is nothing new in automatically processing results from an intrusion detection system. Multiple groups have attempted this, though some differences between the previous work identified and this thesis are covered in the following sections. The structure of this chapter is partly inspired by the work done by [FRC+18], which has covered data types and algorithms in their survey of work in the field. First, we examine how different works use supervision in section 3.1. Then the different data types they have used in section 3.2 before finally covering some of the other algorithmic paradigms used in section 3.3.

3.1 Supervision

The dataset that makes up the basis for this thesis has no labels, which only allows for unsupervised algorithms to be used. Vaarandi[Vaa21] suggested a lightweight, unsupervised algorithm for grouping and clustering Suricata alerts. Alerts with the same external IP are first grouped then clustered based on the signatures present in said groups. Pietraszek and Tanner[PT05] used unsupervised clustering with CLARAty to find the root cause of the most common IDS alerts to resolve the. They also used supervised classification with ALAC for the remaining alerts.

Much work has been done around the supervised classification of IDS alerts, such as Farahnakian and Heikkonen[FH18], and Kannari, Shariff, and Biradar [KSB21], who trained autoencoders on labeled NetFlow. Farahnakian and Heikkonen[FH18] used deep autoencoders to identify critical features representing normal and abnormal behaviors in an imbalanced dataset. They pretrained the first layer as an autoencoder before using supervised training on the compressed output. Kannari, Shariff, and Biradar[KSB21] proposed a sparse autoencoder using swish-PReLU activation to classify attack types, comparing the results with other classifiers. Liu et al.[LGZ+19] used supervision on a deep neural network with ReLU activation to detect network anomalies in NetFlow data.

The area of supervision is not clear cut, as works are also using semi-supervised algorithms. Gharib, Mohammadi, Dastgerdi, and Sabokrou[GMDS19] used a semi-supervised approach that only trained the autoencoder on normal data while ignoring all abnormal data. They used a two-stage autoencoder process where the first, an overcomplete autoencoder, classifies the flows it is confident about using sparsity. In contrast, the second stage, an undercomplete autoencoder, classifies the remaining flows using reconstruction error. Min et al.[MLL+18] proposed the SU-IDS method. This method is pre-training the autoencoder on unlabeled samples, then optimizing for a combination of reconstruction and classification or clustering loss.

Catania, Bromberg, and Garino[CBG12] used Snort IDS to label a dataset. The normal data, which did not trigger any Snort rule, was used to train a Support vector machine (SVM). The trained SVM is then used to detect attacks based on what traffic is dissimilar to the normal data used to train it. As the source data is unlabeled, this is an unsupervised process, while their actual training is semi-supervised, as one class is used to train the model. The algorithm, SVM, is further discussed under section 3.3.

3.2 Data type

While this thesis focuses on alerts from the Suricata IDS, much work focuses on anomaly detection with the network traffic in different data types, such as tcpdump, IP Flows, or SNMP. [CBK09], [HBB+20], and [FRC+18] cover multiple papers that use different formats for their network traffic.

3.2.1 Tcpdump

When one needs a raw copy of network traffic, it is common to use tcpdump. Catania, Bromberg, and Garino[CBG12] used tcpdump to train a Support vector machine (SVM) to identify regular traffic. Cheng, Tay, and Huang[CTH12] compare extreme learning machines' performance to support vector machines for binary and multi-class classifications. The dataset is a tcpdump converted into machine-readable format and trains support vector machines and extreme learning machines for classifications.

3.2.2 IP Flows

If the only part of the network traffic needed is a record of who talked to who at what time, IP Flow is a good choice. Also known as Netflow, IP Flows are metadata records of network traffic, which notes the source/destination IP, TCP or UDP ports for both source and destination, possibly TCP flags such as SYN and ACK, and timestamp of when it happened, unless the flows are time-bucketed[FRC+18]. One standard for IP Flows is IP Flow Information

Export (IPFIX), standardized in RFC 7011 from 2013 by Claise, Trammel, and Aitken[Ait13]. Yeung, Jin, and Wang[YJW07] use NetFlow through a covariance matrix of sequential samples to identify flooding attacks. The covariance matrix could also identify which features were vital in detecting different attacks. Shone, Ngoc, Phai, and Shi[SNPS18] used two datasets, KDD Cup '99 and NSL-KDD, of network flows to train Non-symmetric Deep Auto-Encoders(NDAE) to classify network traffic. A classifier is then created using a stack of NDAEs with a random forest classification algorithm.

3.2.3 Simple Network Management Protocol (SNMP)

SNMP is a standard for retrieving information from and managing network devices. The protocol is a client-server protocol standardized in RFC 1157 by J. Case, M. Fedor, M. Schoffstall, and J. Davin[CFSD90]. SNMP is limited to summarizing packets and bits per interface. This aggregate can be used to detect abnormal traffic flows. Proença, Coppelmans, Bottoli, Alberti, and Mendes[PCB$^+$04] generated digital signatures of the usual traffic situation for different network segments, using linear regression and Hurst parameter to analyze data gathered using SNMP.

3.3 Algorithms

Many different algorithms have been used to look through network traffic, such as statistical, clustering, classification-based, and information-theoretical methods. [FRC$^+$18] covers some different works within these areas.

3.3.1 Statistics

This thesis uses Principal Component Analysis, one of the statistical techniques for anomaly detections. Lakhina, Crovella, and Diot[LCD04] were the first to use PCA for network anomaly detection. They used it to subdivide network traffic volumes into normal and anomalous components. This was done using Origin-Destination flows, extracted from IP flows gathered in two different backbone networks. Camacho, Pérez-Villegas, García-Teodoro and Maciá-Fernández[CPVGTMF16] used PCA on firewall and IDS logs, and on NetFlow, to identify anomalous traffic. The IDS logs were of a different format than what this thesis uses. In addition, they used PCA-based Multivariate Statistical Process Control from the chemical engineering field to create a novel approach to Network Monitoring through Multivariate Statistical Network Monitoring, arguing that this approach is better than the approach used by Lakhina et al. [LCD04].

Fernandes, Rodrigues, and Proença[FRP15] created digital signatures of network segments based on the different flows through the respective network segments. These signatures represent the regular traffic in the network segments

and are generated from the dimensionality reduced output from PCA. Kanda, Fontugne, Fukuda, and Sugawara[KFFS13] used PCA as the basis for their anomaly detection algorithm, ADMIRE, which uses the entropy of different network features to detect anomalies. The anomalies are then categorized into five categories, attack, special, unknown, warning, or benign. Kanda et al. used tcpdump from a backbone network to evaluate ADMIREs performance. All these are using either IP flows or TCPdumps which are significantly different data types than the Suricata IDS alerts used in this thesis.

Others have also used different statistical techniques for anomaly detection. Hamdi and Boudriga[HB07] used wavelet analysis to identify denial-of-service attacks in NetFlow, where the wavelet analysis gives local maxima at points of interest. Callegari, Giordano, Pagano, and Pepe[CGPP11] also used wavelet decomposition on NetFlow traces to look for discontinuities in the time series for the different flows.

3.3.2 Clustering

The PCA and Deep Learning results in this thesis are clustered with K-means. Karami and Guerrero-Zapata[KGZ15] used K-means in hybrid with particle swarm optimization(PSO) to identify abnormal traffic in a Content-centric network by looking at traffic flow in the network. This hybrid algorithm was tested against both K-means and PSO and some methods from other works in the field. It is worth noting that a Content-centric network is significantly different from the traditional host-centric network. Giacinto, Perdisci, Rio, and Roli[GPRR08] tested K-means and other one-class classification algorithms to look for abnormal traffic in different sets of traffic features. They calculated a probability density distribution from each one-class classifier, which they used to calculate a decision criteria for the final classification. Effendy, Kusrini, and Sudarmawan[EKS17] used K-means on a NetFlow dataset to cluster similar traffic and classify them using a naïve Bayes classifier. They compared their method to the use of mean and standard deviation.

Other clustering algorithms are also used, like Mazel, Casas, Labit, and Owezarski[MCLO11], who used sub-space clustering, evidence accumulation, and inter-clustering on NetFlow data to identify DoS, DDoS, scans, and spreading worms in and across subnets.

3.3.3 Classification

The autoencoder used in this thesis is an artificial neural network, which can also be used for classification in supervised training, such as done by Subba, Biswas, and Karmakar[SBK16] on NetFlow from the KDD99 labeled dataset. They preprocessed their source data by converting categorical data to numeric

and normalized the numeric attributes. This is similar to how this thesis has normalized the data, as this is required for the neural network, and the usage difference is mainly in the trained output.

Others have used Support vector machine (SVM), such as Catania, Bromberg, and Garino[CBG12], as mentioned in section 3.2, who used SVM on tcpdump, and Wang, Gu, and Wang[WGW17] used SVM on a NetFlow-based dataset to identify intrusions.

3.3.4 Information theory

Multiple papers have used entropy on NetFlow to identify anomalous traffic. David and Thomas[DT15] used fast entropy to look for DDoS in the flows. They aggregated over the unidirectional flows and used an adaptive threshold to allow for expected increases in traffic. Behal and Kumar[BK17] used ϕ-entropy and ϕ-divergence of different network features to look for DDoS in the flows. Bereziński, Jasiul, and Szpyrka[BJS15] looked more generally for all kinds of anomalies in the flows by looking at different time-windowed entropy calculations on the source and destination address and port, the duration of the flows, packet size, and number, and the degree of the host. Bhuyan, Bhattacharyya, and Kalita[BBK16] used entropy to select the network features to pass to the clustering stage, which they then used to identify anomalies. Amaral, de Souza Mendes, Zarpelão and Proença Junior[AdSMZJ17] used Tsallis entropy on network flows to detect anomalies. These anomalies were, in turn, classified using signatures for the different anomalies.

Chapter 4
Data and Preprocessing

The primary data source is events from a rule-based Network Intrusion Detection System (NIDS), Suricata. Events are generated from the ruleset that is given to Suricata. The data ingress process at Sikt enriches the events with GeoIP and IP address registry, which adds extra context for both source and destination IP. Further details are presented in the different sections of this chapter. In section 4.1, we cover the identified data sources for possible enrichment and correlation with the alerts. Then section 4.2 focuses on importing alerts from the data store. Structuring the data into a column-based data frame is presented in section 4.3, while section 4.4 covers the data's enrichments and feature derivations. Finally, section 4.5 briefly describes the normalization process, and section 4.6 covers the feature selection for later use.

4.1 Auxiliary Data Sources

A thorough list of possible data sources was identified that could be used to give better context to the alerts. These sources were identified through discussion with analysts at Sikt. The identified Auxiliary Data Sources are shown in the following list:

- Netflow
- GeoIP
- Honeypots
- Server logs
- Passive DNS
- Passive TLS
- Application logs

- Threat intelligence
- Suricata Rules
- Port scans
- Incident management systems
- DHCP-logs
- IP-registry

As mentioned in the introduction of chapter 4, GeoIP and the IP address registry were already used to enrich the source alerts. Threat intelligence was also enriched on some alerts, and some of the rules were generated from the same threat intelligence. Netflow is partly included in Suricata as it logs information about the flows that it sees. The rest of the possible datasets were not available in a usable format, so that additional processing would have been necessary. The already enriched datasets were chosen as the datasets to use.

4.2 Data loading

The Suricata data, covered in section 2.2, was exported from Humio[1], the log store at Sikt. To get the data in a more suitable format and location, a small program was created that searched for alerts from April 1st to August 31st of 2021. To not hit any constraints in Humio, the alerts were loaded one day at a time. Each day was then loaded into a Pandas DataFrame [RJM+21] and stored in Apache Parquet format [Fou21], as shown in Listing 4.1.

It was attempted to load from March 1st, but an issue occurred attempting to load data from March. There are some notable dates from within the selected timeframe. April 1st through April 5th was part of the Easter holiday, May 1st, 13th, 17th, and 24th were holidays. In addition, the months can be roughly divided into tutoring in April, exams in May, post-semester in June, vacations in July, and the start of the new school year in August. All these months were also under the effects of the Coronavirus, with varying restrictions in the different parts of the country and at different institutions.

[1] https://www.humio.com/

```
# client is the client to connect to Humio
query = '"alert.signature_id"=*'

start_time = datetime(year=2021, month=8, day=5)
temp_start = start_time
end_time = datetime(year=2021, month=8, day=31)
resolution = timedelta(days=1)
while temp_start <= end_time:
    print(f"Downloading {temp_start} to {temp_start+resolution}")
    stream = client.streaming_query(
        query,
        is_live=False,
        start=round(temp_start.timestamp() * 1000),
        end=round((temp_start + resolution).timestamp() * 1000),
    )
    date = temp_start.strftime("%Y-%m-%d")
    df_alerts = pd.DataFrame.from_dict(list(stream)).to_parquet(
        f"/data/didrik_master/alerts-{date}.parquet"
    )
    temp_start = temp_start + resolution
```

Listing 4.1: Source code for importing events from the log store into dated parquet files.

4.3 Structuring data

After loading the data, as described in section 4.2, the next step is to structure it. A Python library called Dask[Das21], which includes a DataFrame based on Pandas DataFrames, is used to process the data. There are two common issues when working with large data sets in Pandas DataFrames: they need to fit in memory and are not natively parallelizable[Roc15]. Dask is solving these issues allowing for a similar programming experience to Pandas.

The first stage of structuring the data loads all the parquet files into a Dask DataFrame, setting time as the index and then repartitioning the data into multiple smaller partitions. Each partition is a Pandas DataFrame containing a subset of the data. This stage also drops the "@rawstring"-column, which copies the whole event. The Dask DataFrame can then be saved as a multi-partition Parquet file, allowing more optimized access in later processing.

In the second stage, columns that have been flattened, such that they have an index number like "[0]", get the index resolved, either by dropping the index for columns where there is just one item in the column or by merging the different values into a string where there are multiple indices, but they are the last part of a column name. There are also columns with flattened "objects" from the JSON conversion, but these were not necessary for later processing and, as such, dropped.

The final structuring stage was adding datatype to each column, allowing for datatype-specific processing such as on dates or numbers. This stage also marked columns as categoricals, which allow gathering every possible value of a specific category column.

4.4 Enrichment

Extra columns were extracted from the original columns to help the algorithms with some non-obvious connections between the columns. For example, the "http.http_port" is purely numerical, but it is hard for an algorithm to know that only port 80 is the standard HTTP port, while 443 and 8080 are commonly used with HTTP protocols as well, for HTTPS-redirection and non-standard endpoints, respectively.

Another extracted value is whether the rule that triggered was recent. This was gathered by checking the time difference between the last time the alert was updated and the timestamp of the alert. It is noted as recent if it was less than 90 days since the alert was updated when the alert triggered.

If the alert is a TLS alert, the values of "notbefore" and "notafter" from the certificate are recorded. Having the timestamp of the alert, we can derive whether the certificate was valid at the time of the alert.

Four values based on the flow data in the alerts were also calculated, the average size of packets to and from the server, the ratio of packets towards the server to the packets from the server, and the ratio of bytes towards the server to the bytes from the server.

Finally, the alert signature was split into rule source, rule category, and rule name, as the typical naming scheme for the rule is "Source Category Name[...]".

4.5 Normalization

Some of the categorical data columns, such as "#source_host", are not usable in their original form. These columns can increase value by transforming and encoding them with one-hot encoding, dummy encoders, and the like. This transforms one column of n different categorical values into n binary columns.

Another issue with the numerical columns is that the range of possible values can be too broad. The autoencoder requires values of a similar scale, so we normalized them to be between 0 and 1.

4.6 Selection

The exact selection of columns for each algorithm depends on its requirements. As covered in section 4.5, we chose to normalize the numerical values into a range of 0 to 1. The numerical values are also the only ones directly compatible with the autoencoder, so all the columns used for the autoencoder are either of type Float, Integer, or Boolean (though Boolean needs to be converted to a literal numerical such as Float). This selection can be performed when using the data.

In addition, the data was split into three groups, training, validation, and testing. Training data is about 60% of the data, while validation and testing have 20% each. The split should be deterministic through the "train_test_split" function of scikit-learn[PVG+12] when the "random_state" parameter is 0.

Chapter 5
Methods

This section covers the data analysis pipeline, from the preprocessed data to the outlier detection. This pipeline is illustrated in Figure 5.1.

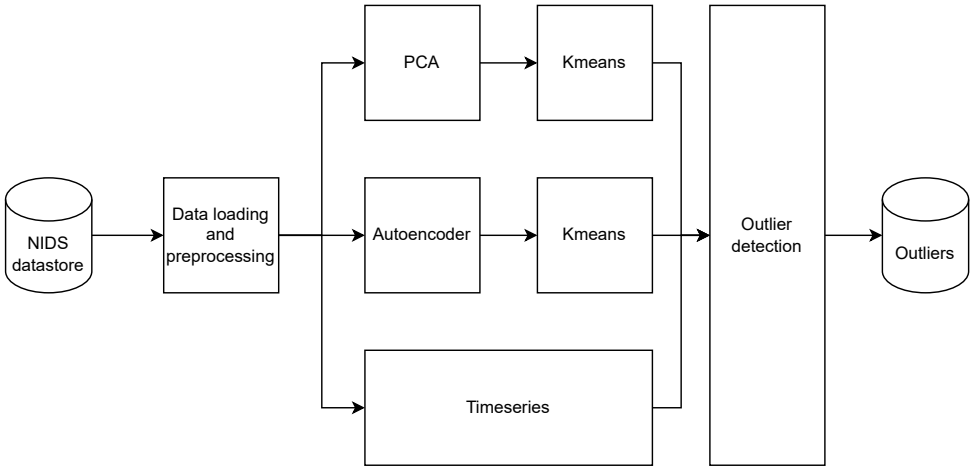

Figure 5.1: Illustration of the data pipeline.

In section 5.1, we cover the use of PCA to reduce the dimensionality of the alerts, while section 5.2 is about the use of a deep learning autoencoder to represent the alerts. These methods are clustered with K-Means, as covered in section 5.3. The alerts are analyzed as a time-series in section 5.4. Before finally, the evaluation method is presented in section 5.5.

5.1 Machine Learning

The machine learning method selected for this thesis is Principal Component Analysis (PCA). The reasoning behind choosing this algorithm is that it reduces the dimensionality of the data. Due to the amount of data in the dataset, dimensionality reduction can significantly improve processing speed. PCA is also

implemented in the libraries used, Pandas[RJM+21] and Dask[Das21], and it works unsupervised, which is a requirement since the dataset is unlabeled. As noted in section 4.6, the features can be selected based on the datatypes of the columns, and the PCA has a similar requirement as the autoencoder in that it requires numerical features. This leaves 92 columns when float, integer, and boolean columns are selected, except the geographic coordinates.

The configuration value in PCA is the number of principal components. For finding the best number of components, we chose a threshold of 95% explained variance. The PCA implementation in Scikit-learn[PVG+12] was used, which allows two ways to compute the number of principal components for a given variance. The first way is to calculate as many principal components as samples or features, use the smaller number, then count up the components until the explained variance passes the threshold. The other way is to give the PCA implementation the threshold directly. Both approaches were tested and returned the same number of principal components, with the same explained variance. There is also an incremental way to implement PCA, which was how the first set of principal components was calculated, while the latter was using standard PCA.

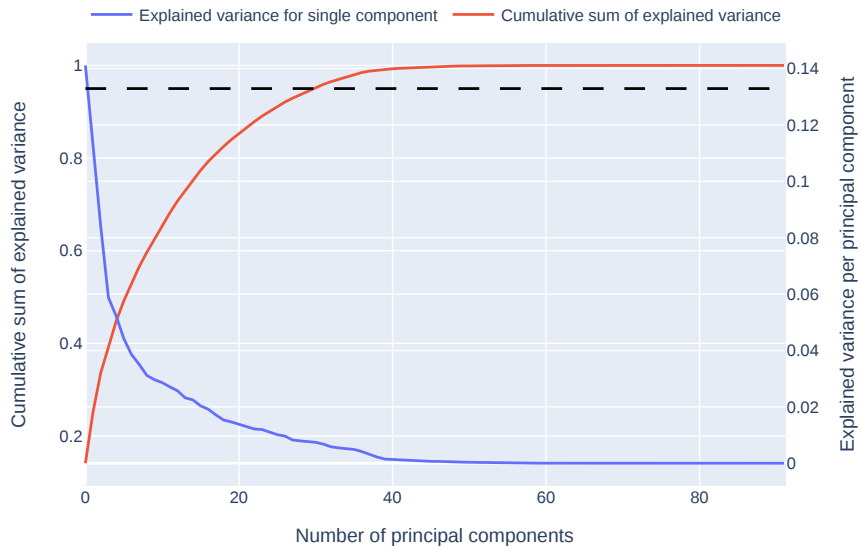

Figure 5.2: This plot shows how the explained variance of the principal components changes with the number of components. Single component variance is on the right axis; the cumulative sum is on the left axis. The dotted line indicates the 95% threshold.

The explained variance and the cumulative sum of explained variances, in percent, are plotted in Figure 5.2. The black dotted line indicates the 95%-threshold for how much variance should be explained in the plot. The intersection between the threshold and the cumulative sum of explained variance is at 31 principal components, which explains 95.2% variance. This was done using the training part of the dataset(N=4 627 171). We then transformed the whole dataset into those 31 principal components.

5.2 Deep Learning

There are multiple approaches within deep learning to find anomalies in the data. Inspired by Wang et al. [WWZZ22], autoencoders were chosen mainly for their ability to work with unlabeled data. The autoencoder is one training method of a neural network, as covered in subsection 2.4.1. As with PCA, 92 features are usable for autoencoding.

Using Keras [C+15], through Scikit-Learn Wrapper for Keras (SciKeras)[B+20], it was possible to create models for autoencoding. The developers of SciKeras have an example for using SciKeras as an autoencoder, including as a Deep Autoencoder[1]. This implementation was used as the basis for the autoencoder in this thesis. The autoencoders have some configuration options. For the loss function, binary cross-entropy was selected, and for the optimizer function, adam was selected. These were chosen as they seemed to work and were used in the example from SciKeras.

A suitable architecture of the neural network was found by searching for the encoding dimension (the smallest layer of the autoencoder) and the hidden layer sizes in a GridSearch. The learning rate for the optimizer was also searched after. 16, 24, and 32 nodes were tested for the encoding dimensions. Zero layers, one of size 32, one of size 64, and two of sizes 64 and 32, were tested for hidden layers. For optimizer learning rate 0.0001, 0.001, and 0.01 were tested. This gave 36 possible combinations to be checked by the Grid Search. The results of this grid search are shown in Table B.1. An encoding dimension of 16 was chosen from the grid search, with a hidden layer of 64 and a learning rate of 0.001. This reduces the number of features from the original 92 to 16.

5.3 Clustering

Both the PCA and the autoencoder reduced the number of features, but the resulting values only represent the original data, not having any meaning on their own. Therefore they were clustered using Kmeans. First, the number of clusters to use had to be determined. This was done using the elbow method

[1] https://www.adriangb.com/scikeras/stable/notebooks/AutoEncoders.html

and silhouette score, see subsection 2.6.1, on ten percent of the combination of training and previous validation part of the data, as there is no need for the validation set distinction further. Ten percent was selected as a sampling rate as any higher sampling rate used too much memory and time.

The inertia of the data from both the PCA and the autoencoder was plotted in line with the elbow method. These plots are shown in Figure 5.3 and Figure 5.4. As can be seen from the plot, there is no clear elbow, so a silhouette analysis was performed.

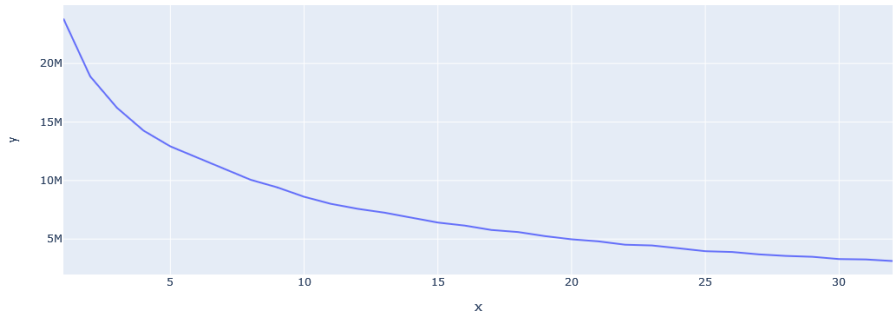

Figure 5.3: This plot shows how the inertia in the clusters of the autoencoder data changes with the number of clusters.

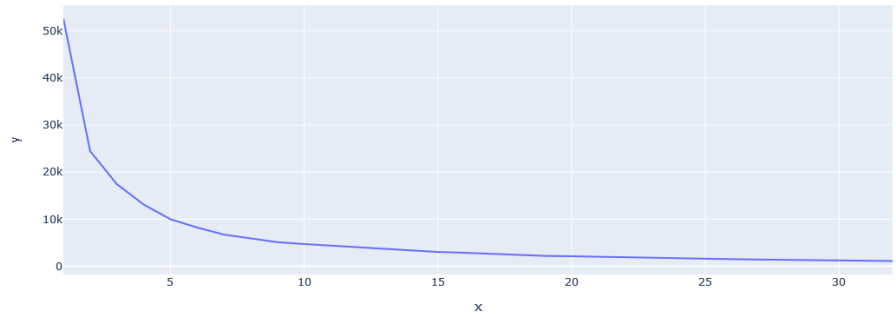

Figure 5.4: This plot shows how the inertia in the clusters from the PCA dimensionality reduction changes with the number of clusters.

Silhouette analysis was also performed on both datasets; the scores are shown in Figure 5.5 and Figure 5.6. As we look for a higher score and lower cluster count, the first local maxima were chosen as the number of clusters to use. For

the data from the autoencoder, this meant k=25, while the data from the PCA gave k=8. For the exact scores, see Table C.1.

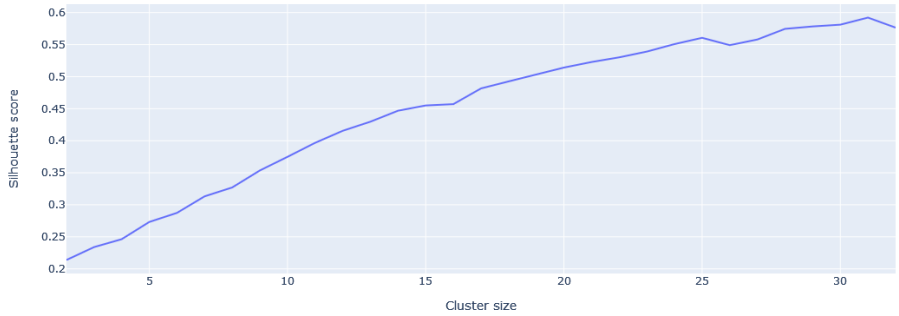

Figure 5.5: This plot shows how the silhouette score in the clusters of the autoencoder data changes with the number of clusters.

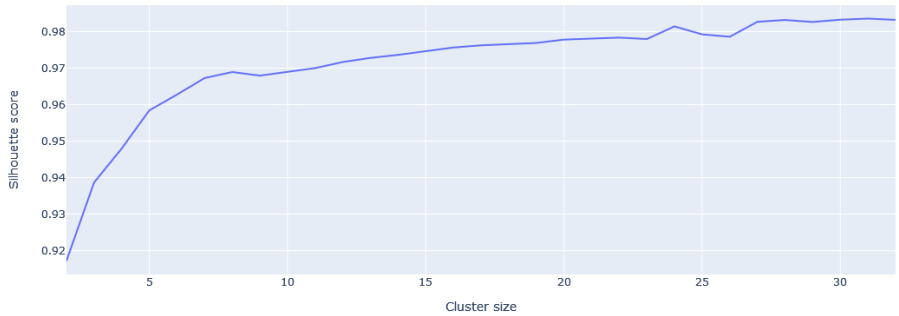

Figure 5.6: This plot shows how the silhouette score of the clusters from the PCA dimensionality reduction changes with the number of clusters.

Using the selected cluster sizes, a K-Means clustering was fitted on the training and validation sets, then cluster membership and distances were calculated for the whole dataset. Based on this, three scores were generated, with close to zero being the preferred value. First, an intersection score was calculated by taking 1 divided by the difference between the second closest cluster to the closest cluster for each alert, see Equation 5.1. The reason for this score is that an alert that is very close to being a member of another cluster could be abnormal. The two other scores are statistical from the distance to the cluster. One is clustered; see Equation 5.2, and the other is across the whole dataset; see Equation 5.3. The sample mean, $\bar{x}_k$ and $\bar{x}$, and the sample standard deviation, s_k and s, are calculated. K sets for the first score and one for the second. Each alert

is then scored by taking its distance, subtracting the mean difference, dividing the whole by the standard deviation. D_{ji} is the distance from the i-th point to the j-th closest cluster centroid in the formulas for the scores.

$$score_{intersection,i} = \frac{1}{D_{2,i} - D_{1,i}} \tag{5.1}$$

$$score_{clustered\ distance} = \frac{x_i - \bar{x}_k}{s_k} \tag{5.2}$$

$$score_{distance} = \frac{x_i - \bar{x}}{s} \tag{5.3}$$

5.4 Time Series

All the alerts have a timestamp which allows them to be seen as a time series. What the alerts lack is a single value that represents it over time. Weather observations usually have a value for temperature or humidity that evolves over time. By binning the alerts based on different metadata, the count of alerts with each value is such a value that represents it over time. Therefore the alerts were split into one time series for each dimension value, and the alerts were put into one-hour bins. The metadata that was used to split the alerts were "#source_host", "rule_id", "rule_category", "rule_source", and "alert.category".

A window of 28 days was selected for the time series, from which the mean and standard deviation were calculated. The values were stored in a new time series at the end of each time window. Due to this, the first 28 days cannot be evaluated. For the rest of the one-hour time bins, their score was calculated by taking the count of alerts in each time bin, subtracting the mean count of alerts, then dividing by the standard deviation, keeping the mean and standard deviation separate per group. This created five different scores, and each alert looked up their respective scores from their time bin.

5.5 Evaluation Methodology

In the evaluation, the first step is to identify where to set the threshold per column for what should be considered an outlier. This is done using the outlier detection methods Interquartile Range (IQR) and robust Z-score. These were used on the upper side of the median, where the most prominent outliers were, and on both sides. The resulting thresholds were then used to create a binary representation of whether an alert is an outlier for each column.

Further analysis is primarily done on those alerts considered outlying in at least one column.

5.5.1 Manuel labeling

Since the original dataset has no labels, we decided that parts of the test dataset should be labeled to evaluate the alert scoring. This set contains 1 541 574 alerts. A web page was set up, and two analysts in the CERT at Sikt were asked to classify as many alerts as possible. Some valid criticism of the classification options was raised, which was selected from the ones available in an older system for classifying alerts, Snorby[2], rather than the newer and more accepted ENISA Reference Incident Classification Taxonomy [3].

Each of the analysts used around twenty minutes to classify alerts, and as the process of classifying alerts was time-consuming, only 34 alerts were classified. The classifications' distribution was seven attempted unauthorized access, 9 False positives, three unauthorized user access, and 15 virus infections. The alerts to be classified were given in inverse chronological order, starting with the newest alert. The manually classified alerts were from August 27th to August 31st of 2021. As we wanted to make the classification as easy as possible, there were multiple alerts to select from, and there were no restrictions regarding which alerts the evaluators could classify. The classified ones are a minuscule subset of the total alerts in the period. It is also safe to assume that the classified alerts were the most apparent classifications. The distribution of classifications in the classified alerts is likely not in line with the distribution of actual classifications in the period.

[2] https://github.com/Snorby/snorby
[3] https://www.enisa.europa.eu/publications/reference-incident-classification-taxonomy

Chapter 6
Results and analysis

In the following sections, the results of this thesis are presented. The results are divided into three main parts. In "Identifying Outliers", section 6.1, the aim is to determine where to draw the line for what is an outlier, which is then considered as the true positive, or actionable, alerts. In "Evaluating Outliers", section 6.2, the identified outliers are assessed regarding how outlying they are and how their values correlate. Finally, in "Comparing with Manually Evaluated Alerts", section 6.3, the outliers are compared to the manual classification to indicate how good the outlier detection is.

6.1 Identifying Outliers

The first evaluation stage gathers all the scores from the clustering of PCA and autoencoding and the time series so that each alert and all the scores are mapped together. Indexed by timestamp and the alert ID, the table contains 11 columns, five from time series and three from each clustering.

For each column, we see some infinite values in the time series scores from the data. 10 in "alert.category", 9544 in "rule_source", 9576 in "rule_category", and 11467 in "rule_id". This was a known challenge with the time series analysis. When the standard deviation is 0, the score becomes infinite. This occurs when the alert is seen for the first time. They are capped at the maximum numerical value to process these alerts further. The maximum scores in the different columns are shown in Table 6.1.

Looking at the histograms of the distributions in Figure 6.1 and Figure 6.2, there are many bins without any members. The histograms were set up to use 100 bins to differentiate within the distributions better. The time-series distributions were particularly skewed towards the extremities, with some of them only having members in two bins, one around 0 and one around a significantly higher power of ten. The intersect scores were also skewed towards the extremities, with multiple empty bins, but not to such an extent as in the time series distribution.

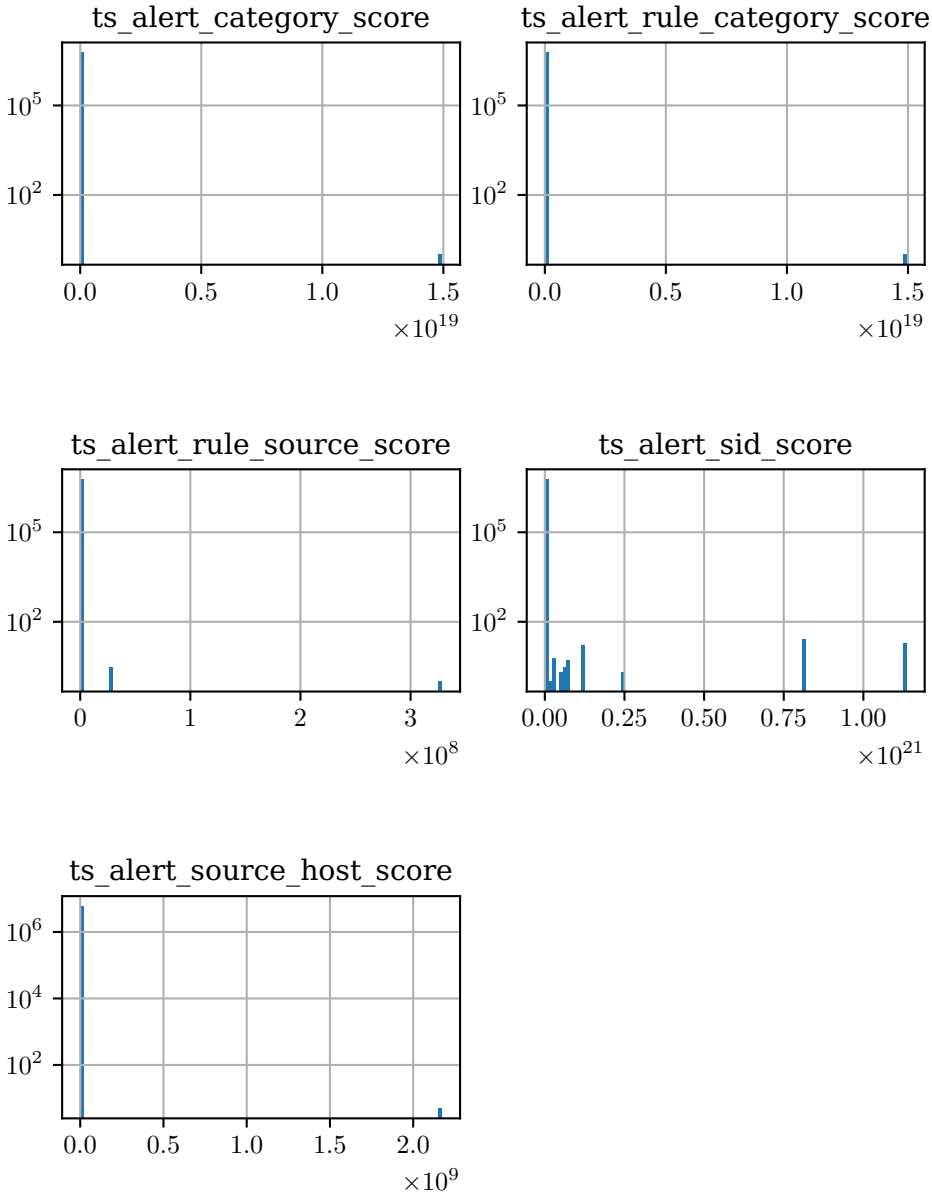

Figure 6.1: Shows the distribution of values in time series columns. There are 100 bins across the X-axis. The Y-axis is logarithmic due to the difference in the number of alerts in each bin.

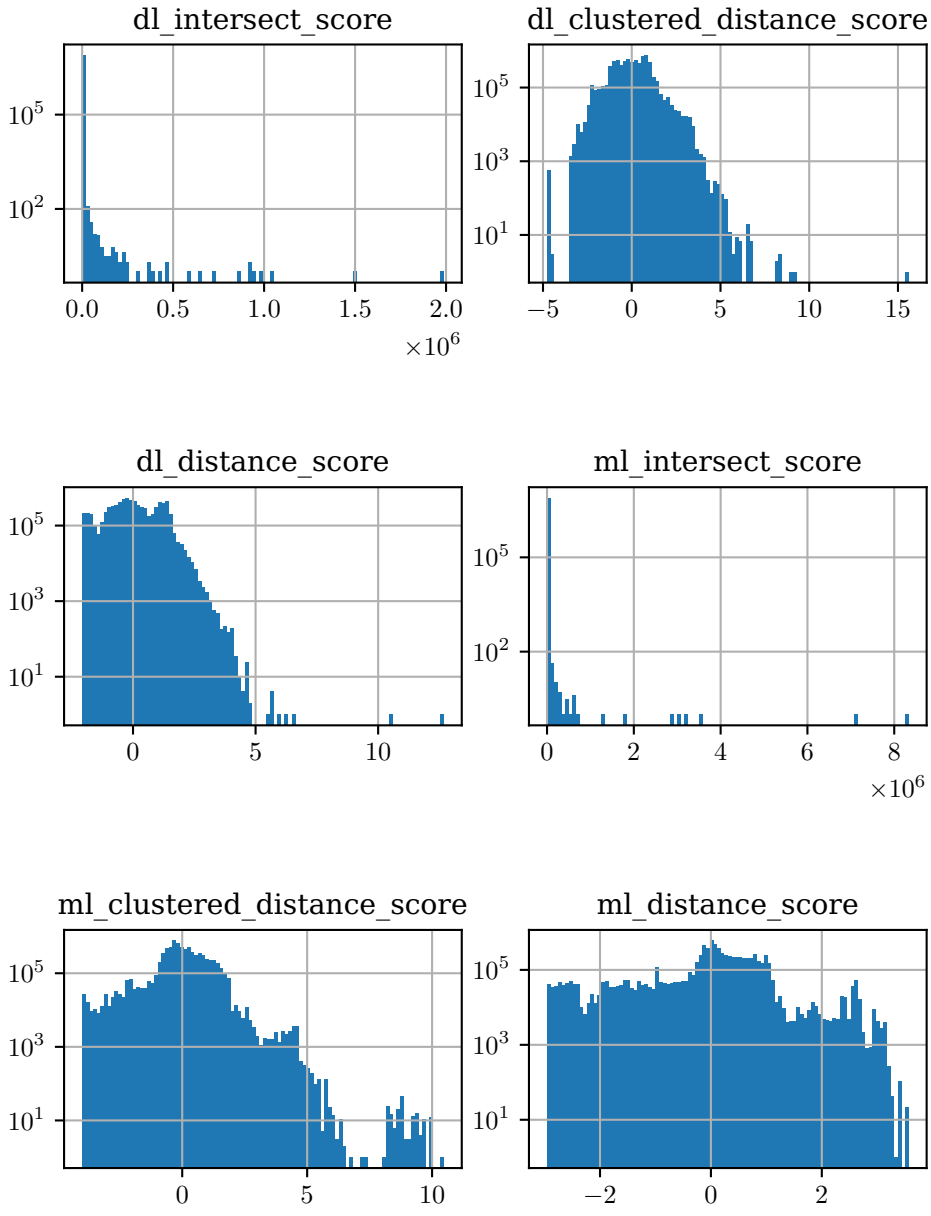

Figure 6.2: Shows the distribution of values in PCA and Autoencoder columns. There are 100 bins across the X-axis. The Y-axis is logarithmic due to the difference in the number of alerts in each bin.

	max
ts_alert_category_score	1.494357e+19
ts_alert_rule_category_score	1.494357e+19
ts_alert_rule_source_score	3.285202e+08
ts_alert_sid_score	1.135711e+21
ts_alert_source_host_score	2.172957e+09
dl_intersect_score	1.987019e+06
dl_clustered_distance_score	1.559742e+01
dl_distance_score	1.267345e+01
ml_intersect_score	8.330280e+06
ml_clustered_distance_score	1.046444e+01
ml_distance_score	3.564804e+00

Table 6.1: These are the maximum values seen in the scores and in which column they belong.

Two different techniques were tested to find where to set the outlier limit for the different columns, a robust implementation of Z-score [RH11] and Interquartile Range (IQR) [YRF19]. The main difference between the robust implementation of Z-score and the typical implementation is that instead of using mean and standard deviation, which can get skewed by a single, vast outlier value, it uses median and median of all absolute deviations from the median (MAD), which can sustain up to 50% outliers. IQR has robustness for up to 25% outliers. There is one parameter for each of the methods. Robust Z-score has the parameter to select the outlier limit; 2.5 was used in [RH11]. IQR has a parameter for how many times IQR should add to the 75th-percentile for the limit. Some different suggestions were 1.5 from [YRF19] and 1.5 as an inner limit, and 3 as an outer limit from [Gut20]. To limit the removal of non-outlier values, the conservative option of 3 was used.

From Figure 6.1 and Figure 6.2, it seems to be a majority of upper outliers, so first, the outlier detection just attempted to remove upper outliers as shown in Figure 6.3, Figure 6.4, Figure 6.5, and Figure 6.6. These outliers were identified through a single iteration of IQR and Z-score.

Looking at the "ts_alert_sid_score" in both IQR (Figure 6.3) and Z-score (Figure 6.5), it seems to be at least one outlier on the lower end of the distribution. A new round of outlier detection was run on both sides of the median, shown in Figure 6.7, Figure 6.8, Figure 6.9, and Figure 6.10.

To compare the results of outlier detection on only the upper side and both sides, the number of outliers detected by each technique can be used. The results

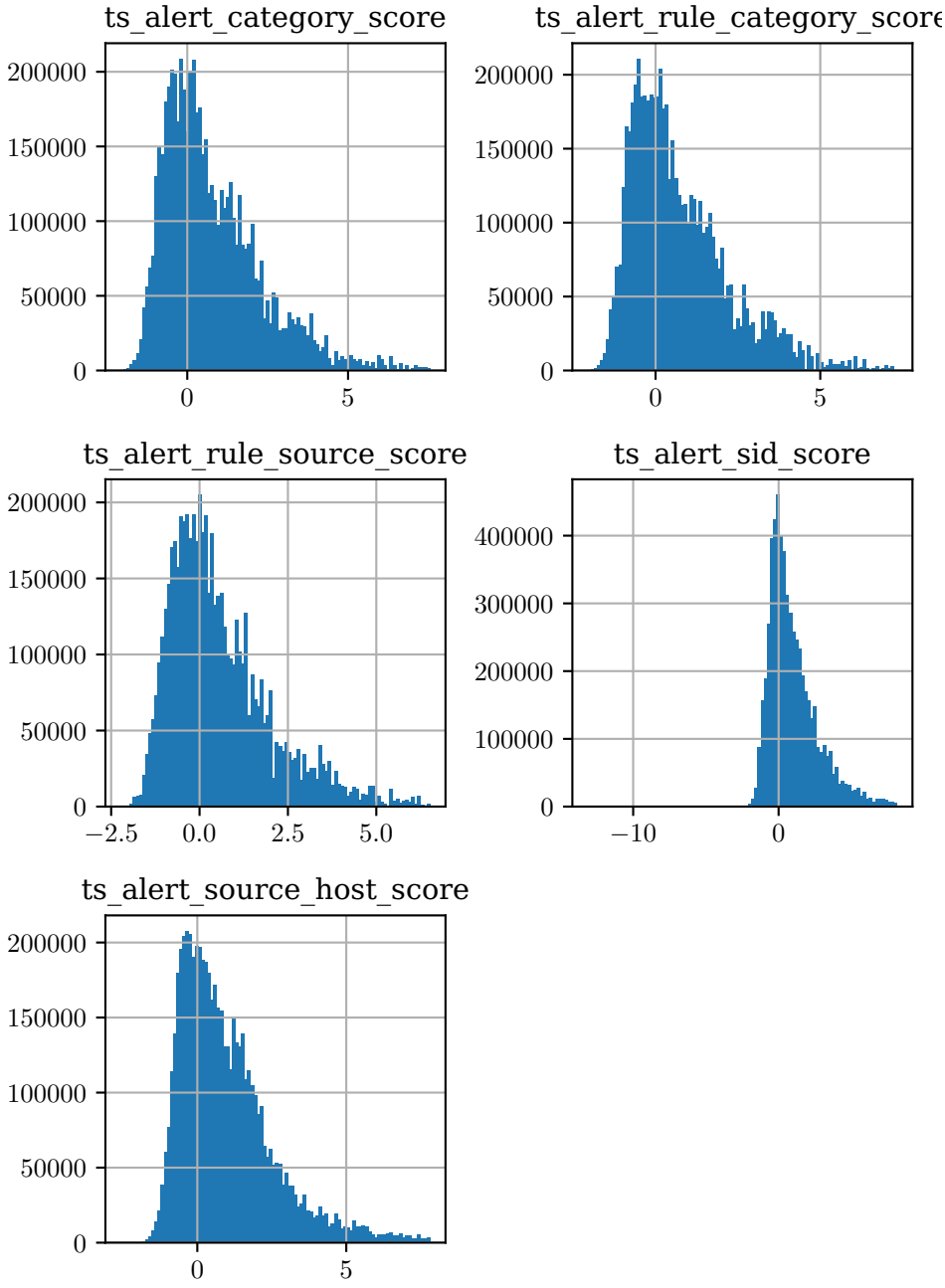

Figure 6.3: Shows the distribution without upper IQR outliers in time series columns. There are 100 bins across the X-axis.

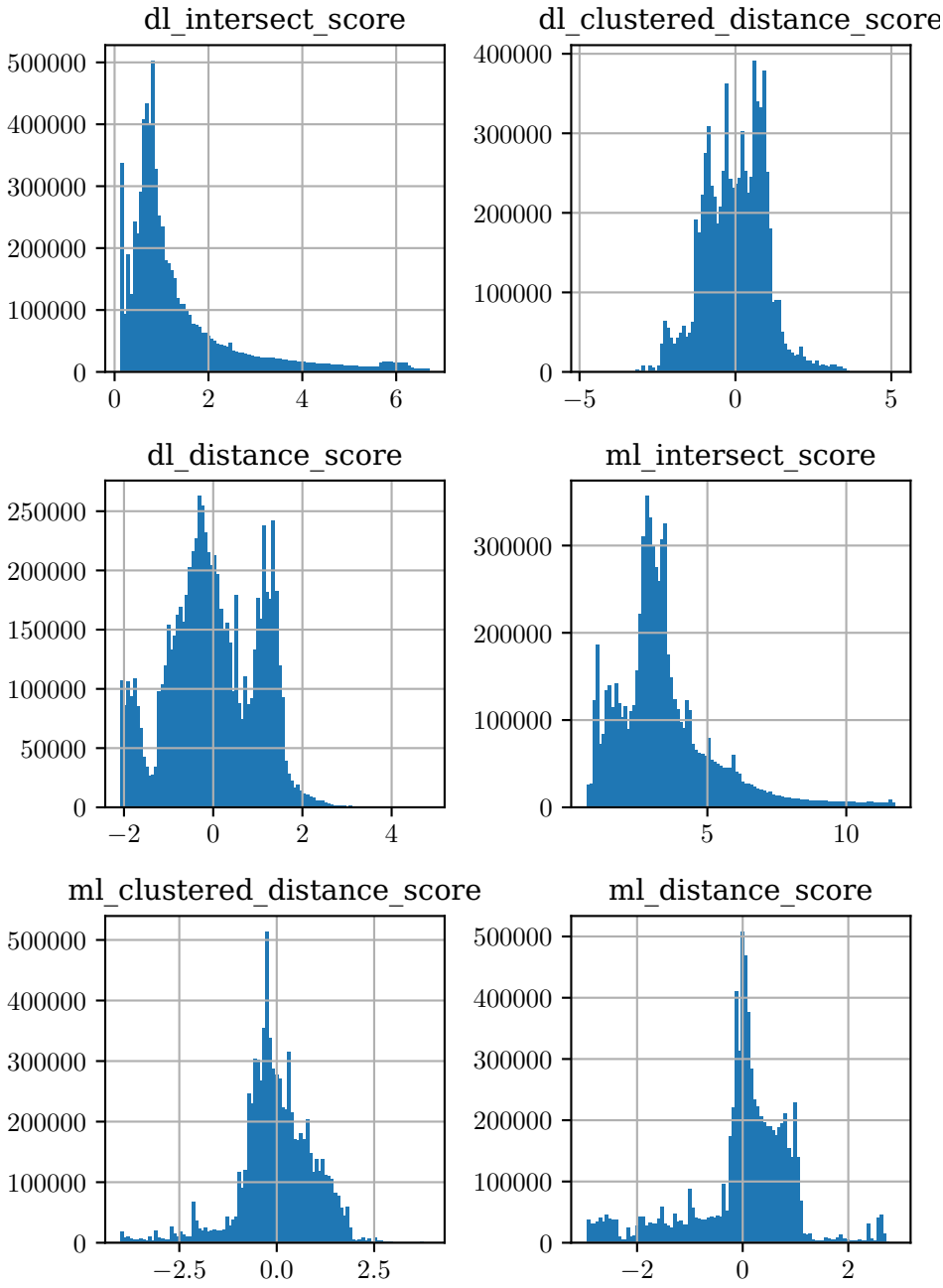

Figure 6.4: Shows the distribution without upper IQR outliers in PCA and Autoencoder columns. There are 100 bins across the X-axis.

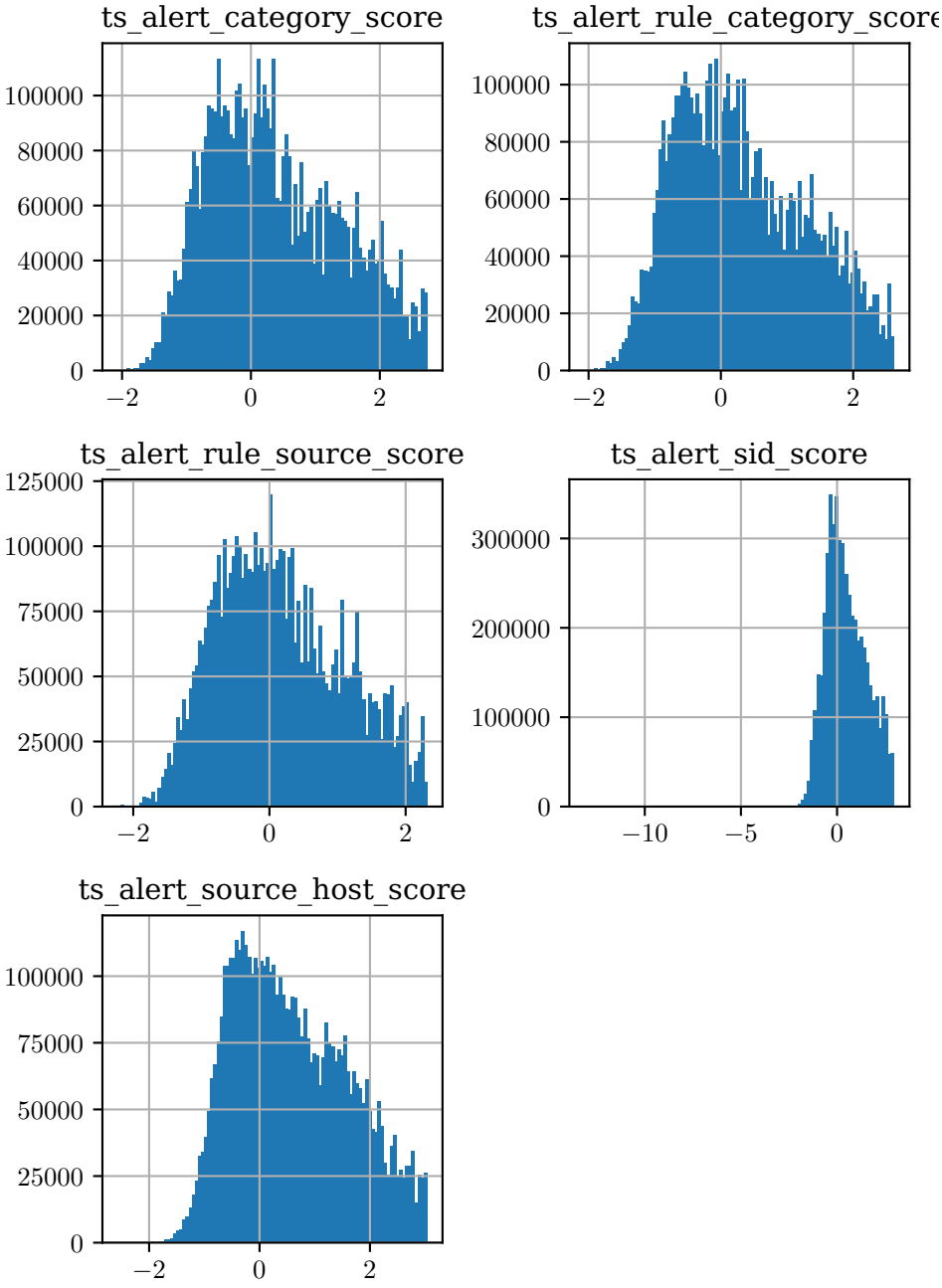

Figure 6.5: Shows the distribution without upper Z-score outliers in time series columns. There are 100 bins across the X-axis.

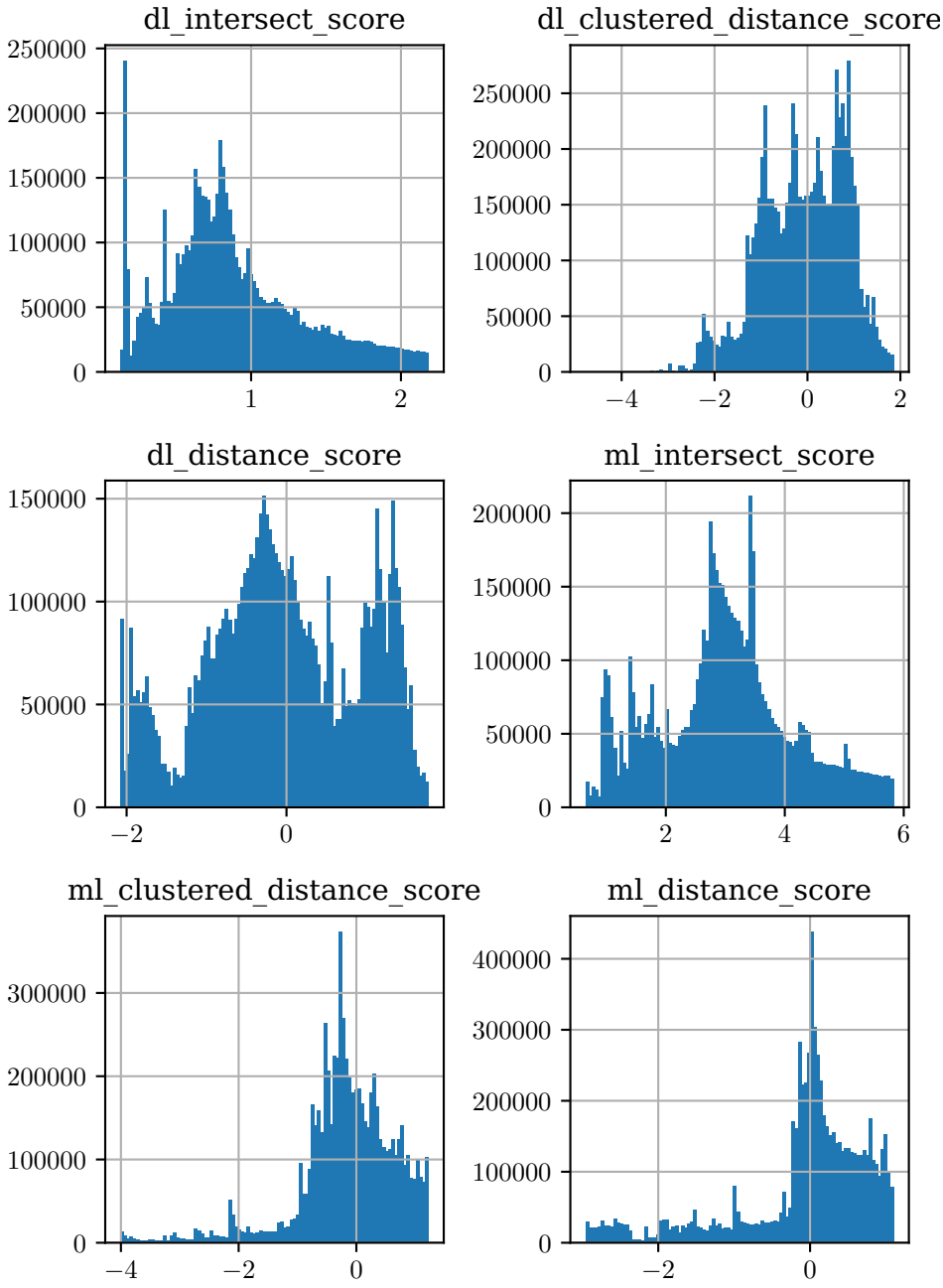

Figure 6.6: Shows the distribution without upper Z-score outliers in PCA and Autoencoder columns. There are 100 bins across the X-axis.

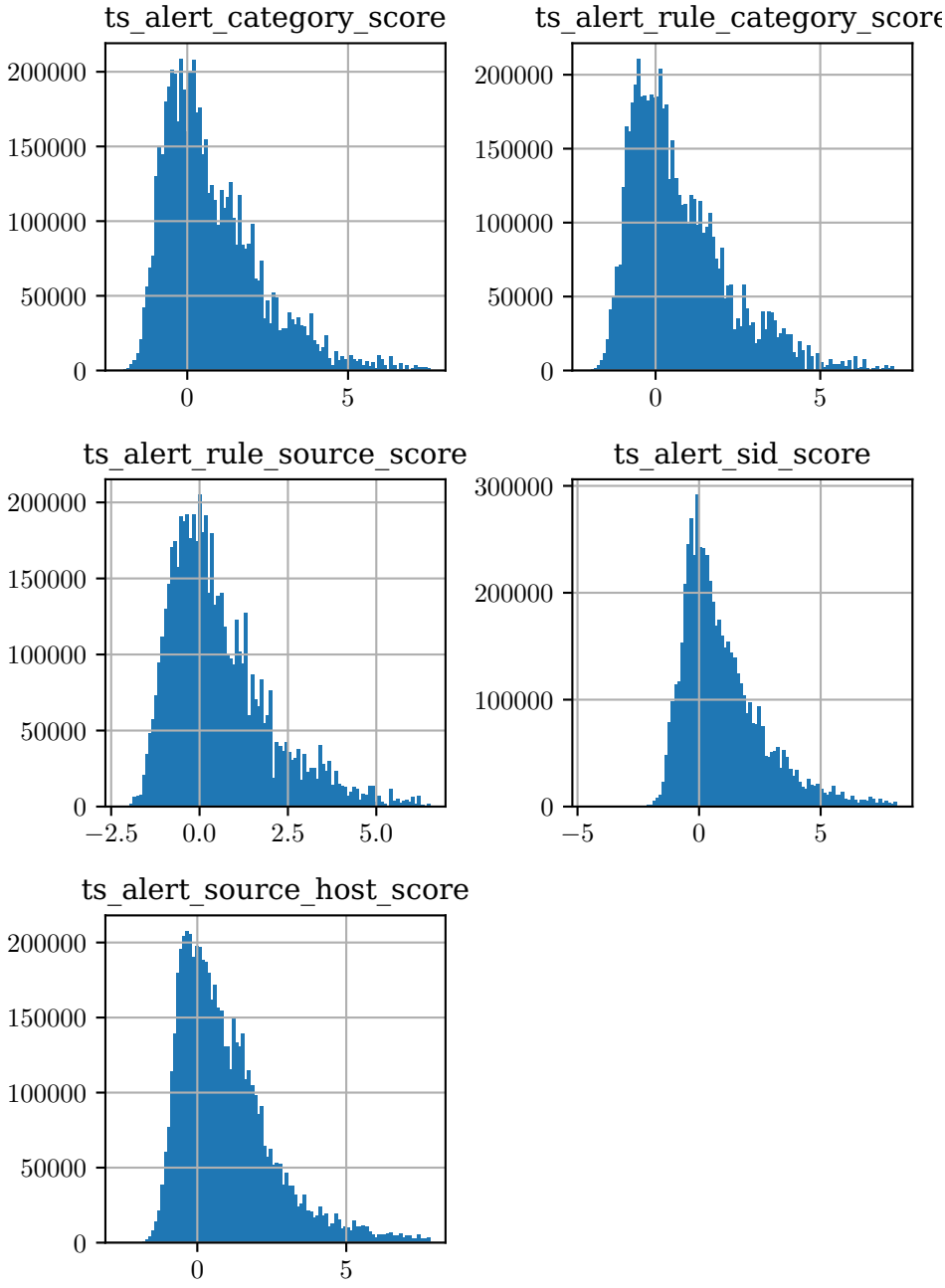

Figure 6.7: Shows the distribution without IQR outliers in time series columns. There are 100 bins across the X-axis.

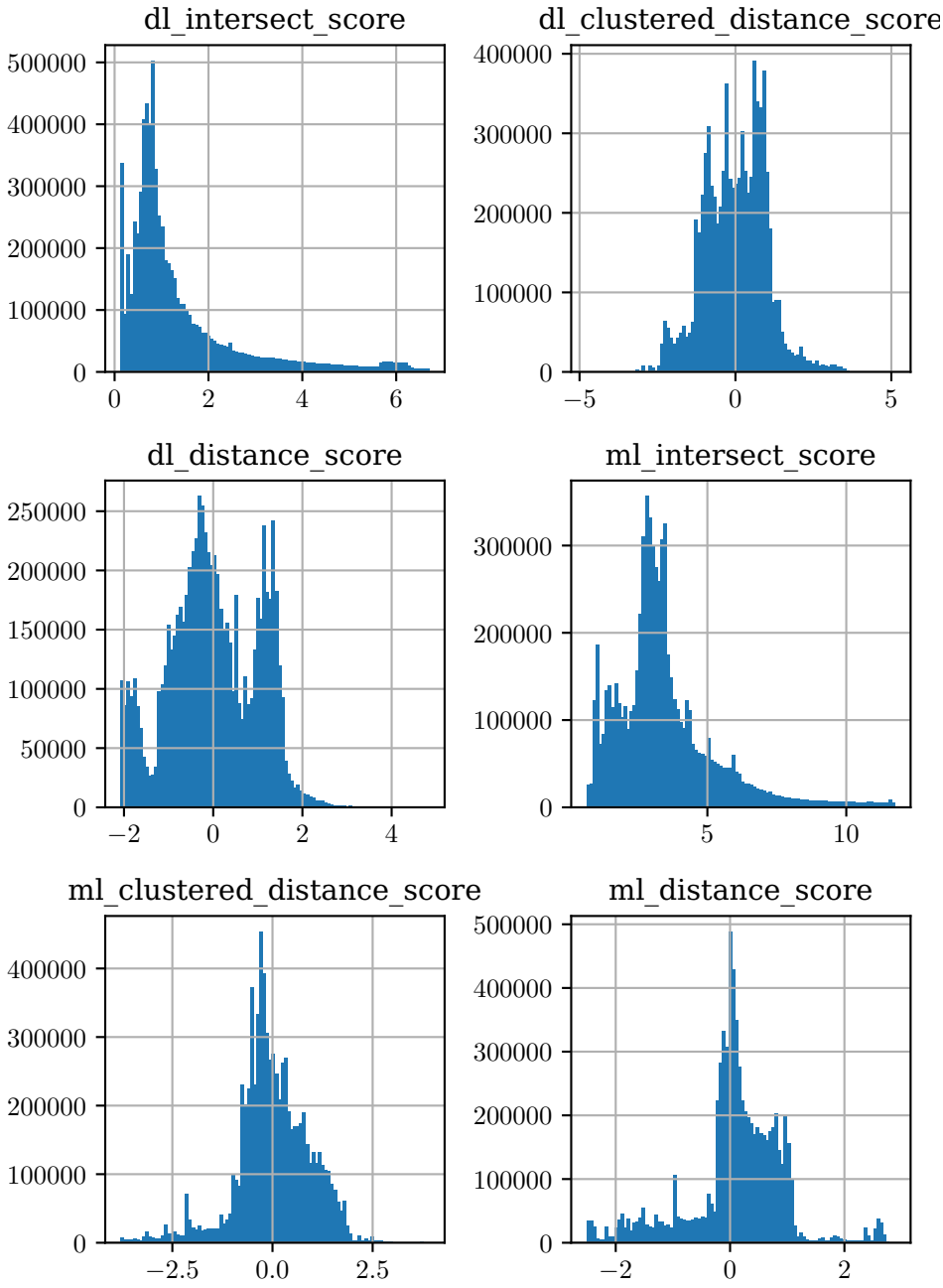

Figure 6.8: Shows the distribution without IQR outliers in PCA and Autoencoder columns. There are 100 bins across the X-axis.

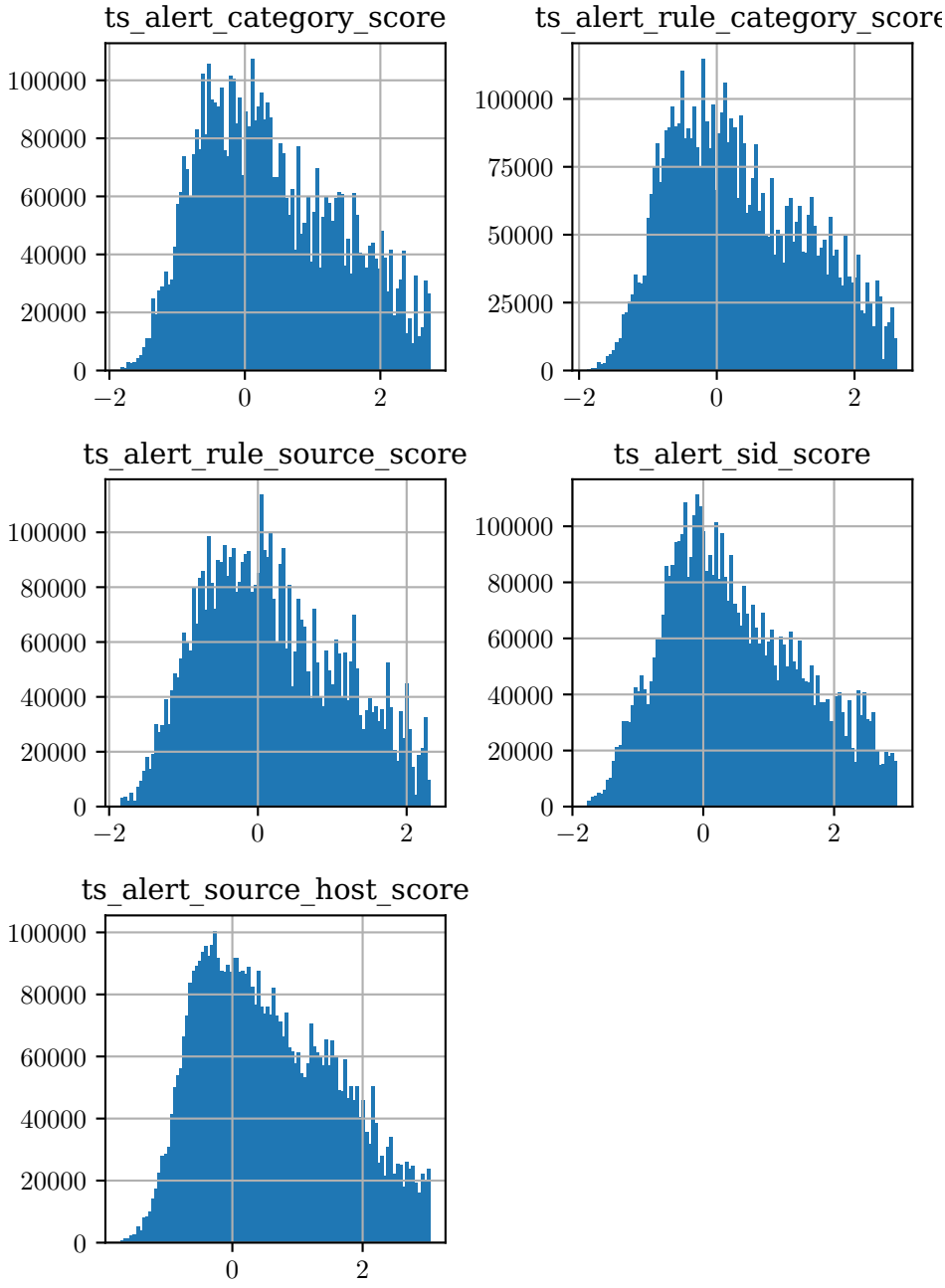

Figure 6.9: Shows the distribution without Z-score outliers in time series columns. There are 100 bins across the X-axis.

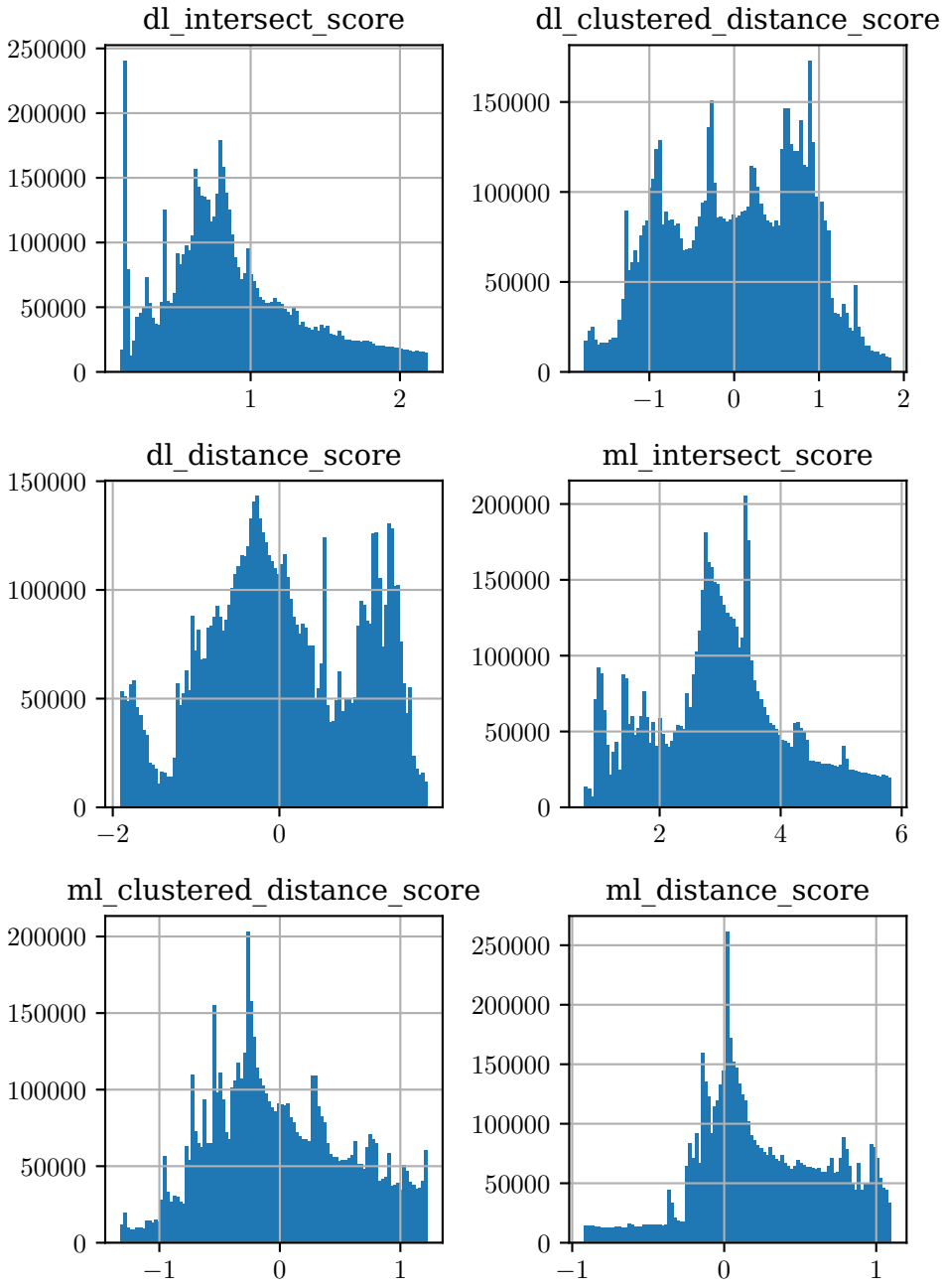

Figure 6.10: Shows the distribution without Z-score outliers in PCA and Autoencoder columns. There are 100 bins across the X-axis.

can be seen in Table 6.2.

Looking at the number and the ratios of outliers and the distribution of non-outliers, Two-sided IQR seems to give the smoothest distribution of non-outliers without too many outliers. Looking at non-outliers for two-sided Z-score in Figure 6.9 and Figure 6.10, there seem to be cliffs in the distribution due to the threshold range being smaller than the range in two-sided IQR, which is seen in Figure 6.7 and Figure 6.8. Based on these points, we chose Two-sided IQR, with the limits shown in Table 6.3.

Scores	IQR				Z-score			
	Single-sided		Double-sided		Single-sided		Double-sided	
Timeseries	Count	Percent	Count	Percent	Count	Percent	Count	Percent
alert category	66713	1.125724	66713	1.125724	718129	12.117801	719534	12.141509
alert rule category	55792	0.942963	55792	0.942963	751796	12.706413	752910	12.725241
alert rule source	38532	0.651242	38532	0.651242	746665	12.619623	751376	12.699245
alert sid	125245	2.117494	125359	2.119421	843490	14.260728	854103	14.440160
alert source host	121007	2.041884	121007	2.041884	705131	11.898451	707849	11.944315
Autoencoder intersect	673165	8.733476	673165	8.733476	1916437	24.863380	1916437	24.863380
clustered distance	248	0.003217	248	0.003217	220931	2.866304	558046	7.239951
distance	10	0.000130	10	0.000130	127462	1.653660	370400	4.805478
PCA intersect	787619	10.218374	787619	10.218374	1473469	19.116423	1498499	19.441156
clustered_distance	16874	0.218919	52398	0.679799	757215	9.823920	1289448	16.728980
distance	20520	0.266221	303032	3.931462	345273	4.479487	1527901	19.822610

Table 6.2: The table shows the number of detected outliers and how big a fraction the outliers take up.

Column	Lower outlier limit	Upper outlier limit
ts_alert_category_score	-6.258553	7.560694
ts_alert_rule_category_score	-6.164582	7.334157
ts_alert_rule_source_score	-5.735031	6.579124
ts_alert_sid_score	-6.453628	8.141691
ts_alert_source_host_score	-6.169317	7.813006
dl_intersect_score	-3.877442	6.702882
dl_clustered_distance_score	-5.142823	5.124152
dl_distance_score	-5.372881	5.578274
ml_intersect_score	-4.299337	11.759377
ml_clustered_distance_score	-3.790836	3.911523
ml_distance_score	-2.508907	2.887623

Table 6.3: The table shows the selected lower and upper outlier limits from Two-sided IQR.

6.2 Evaluating Outliers

After selecting the limits for outliers as shown in Table 6.3, the alerts where at least one column is considered outlying were gathered. This led to a total of 1 828 738 alerts with outliers, divided into 1 150 662 from the training set, 410 687 from the validation set, and 324 541 from the test set. Each outlier alert has between 1 and 7 columns where its value is outside the threshold. From Figure 6.11, most alerts have only one outlying column, and none have outliers in all 11 columns.

The correlation of all the scores is shown in Figure 6.12 as a reference. The correlation between the outlier values, solely the values that pass the thresholds, is shown in Figure 6.13. These are mostly sparsely filled alerts, as the alerts have only one or a few outlying columns, as seen in Figure 6.11. Finally, Figure 6.14 shows the correlation of the alerts with an outlier. These are the same alerts as the previous correlation plot, but all the non-outliers are restored from the original scores.

From the correlation plot in Figure 6.12, there appears to be a strong correlation between clustered and non-clustered distance score for both the PCA ("ml_clustered_distance_score", "ml_distance_score") and the autoencoder ("dl_clustered_distance_score", "dl_distance_score"). There are also indications of correlation between the distance scores of the autoencoder ("dl_distance_score") and the PCA ("ml_distance_score"). The first observation might indicate that using both features is redundant. This similarity might also be explained by K-means trying to reduce all the alert's distances to their closest cluster centroid.

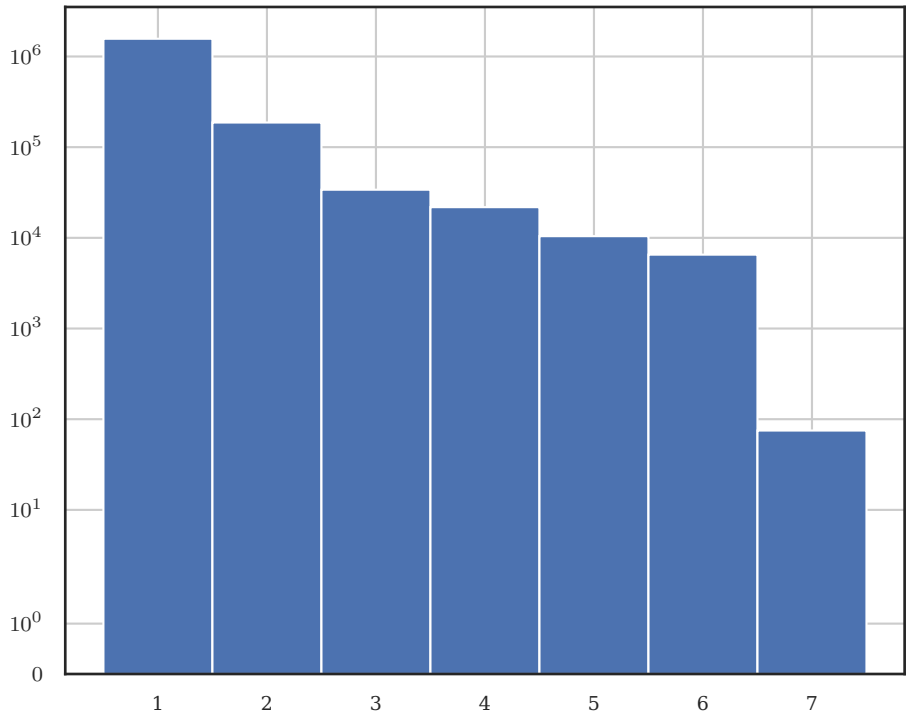

Figure 6.11: Shows the distribution of alerts by the number of outlying columns each alert has. The Y-axis is logarithmic to fit everything on one page.

The second observation indicates that there might be a common feature that both the PCA and the autoencoder identify. However, it could be explained by both the autoencoder and PCA scores being derived from K-means clustering of the original data.

6.3 Comparison with Manually Evaluated Alerts

As covered in subsection 5.5.1, we have a small set of manually classified alerts. All these are from the test set, of size 1 541 574, and as mentioned in section 6.2, there are 324 541 outliers in the test set. This leads to about 21% of the test set being outliers, one-fifth of the original size.

Looking up the small set of 34 manually classified alerts in the outliers, 24 of them were in the outlier set, which corresponds to 70%. 18 out of 25 true positive classifications and 6 out of 9 false-positive classifications were evaluated as outliers. These results can be shown in a confusion matrix, Table 6.4.

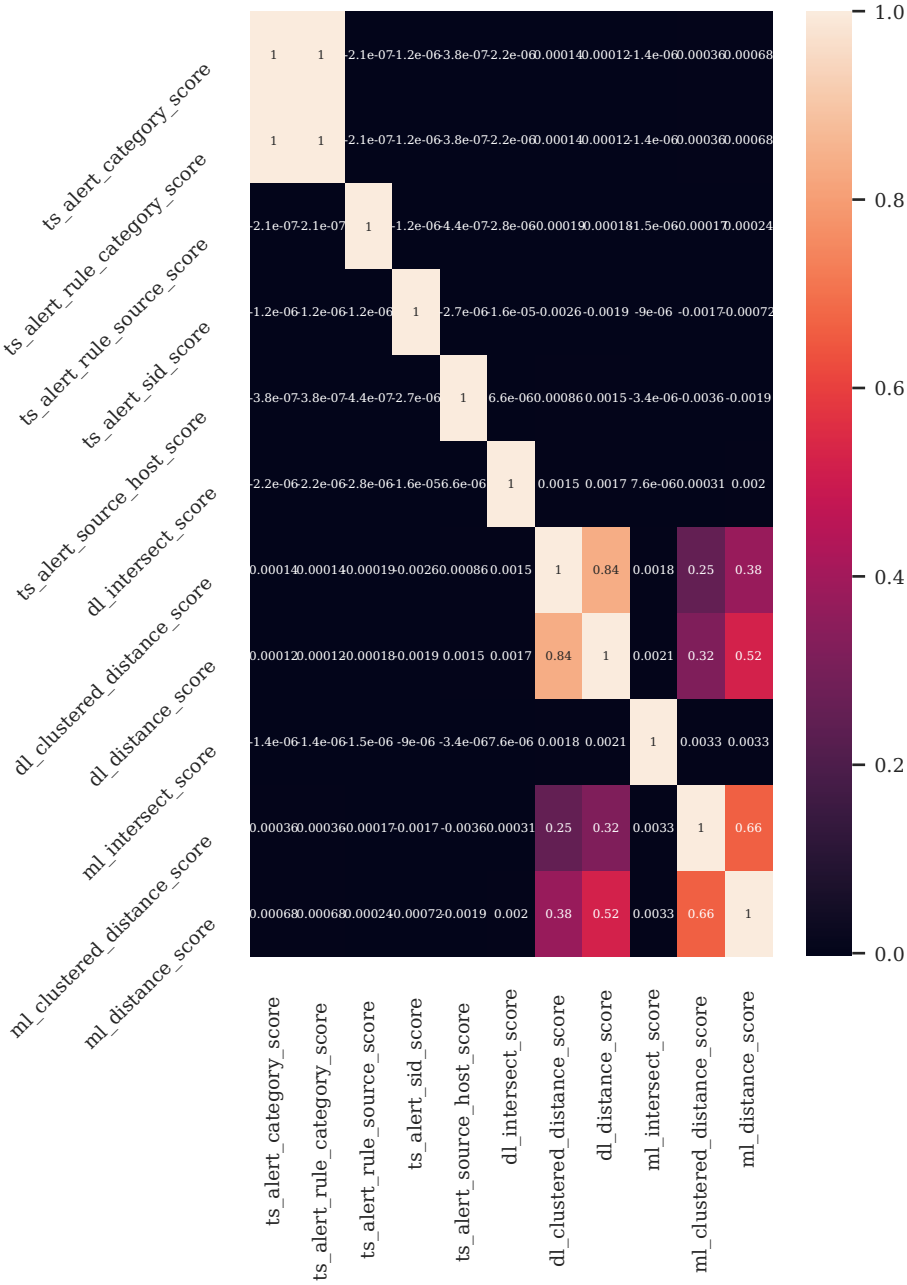

Figure 6.12: Shows the correlation between all columns for all alerts.

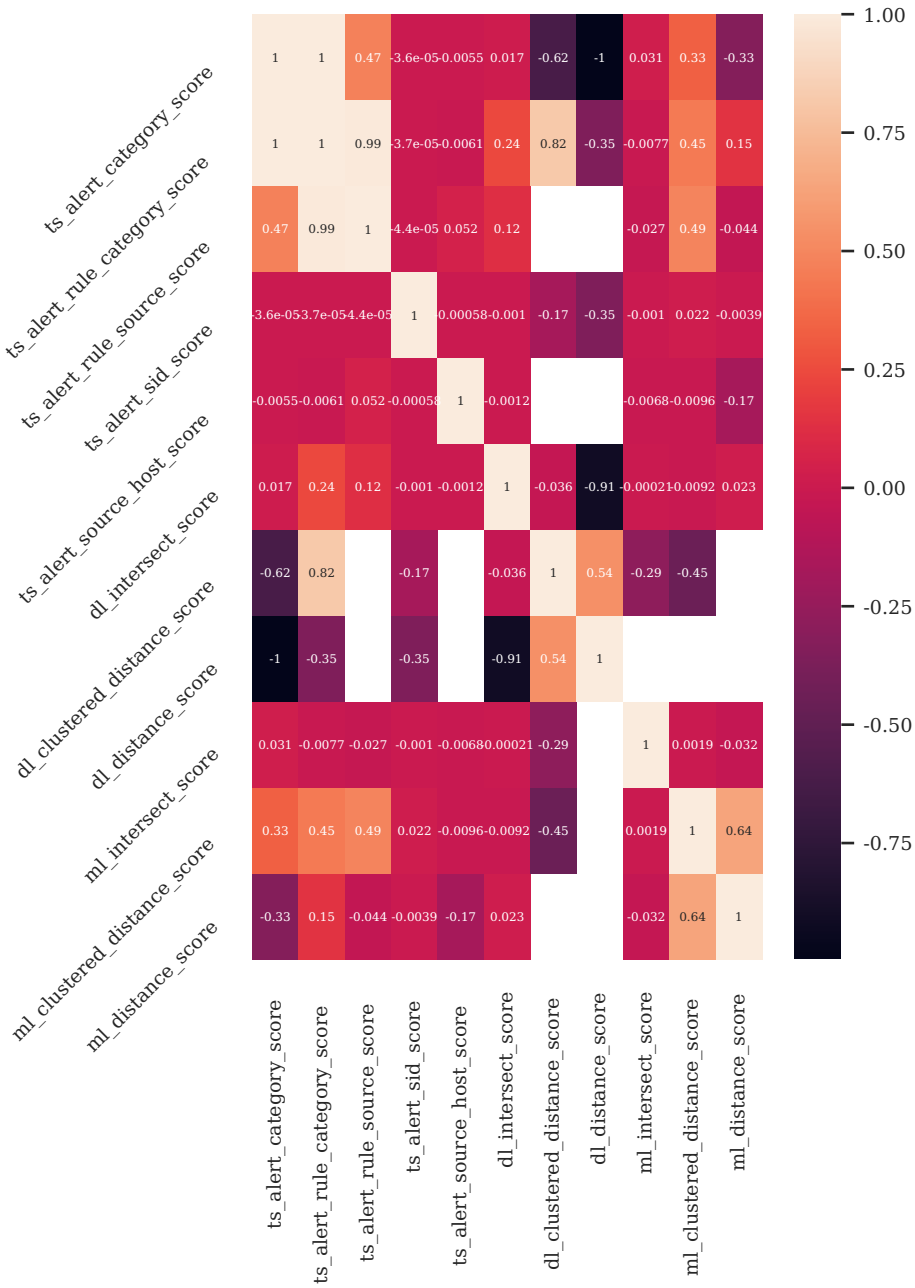

Figure 6.13: Shows the correlation between all columns for the outlying alerts. White boxes are empty.

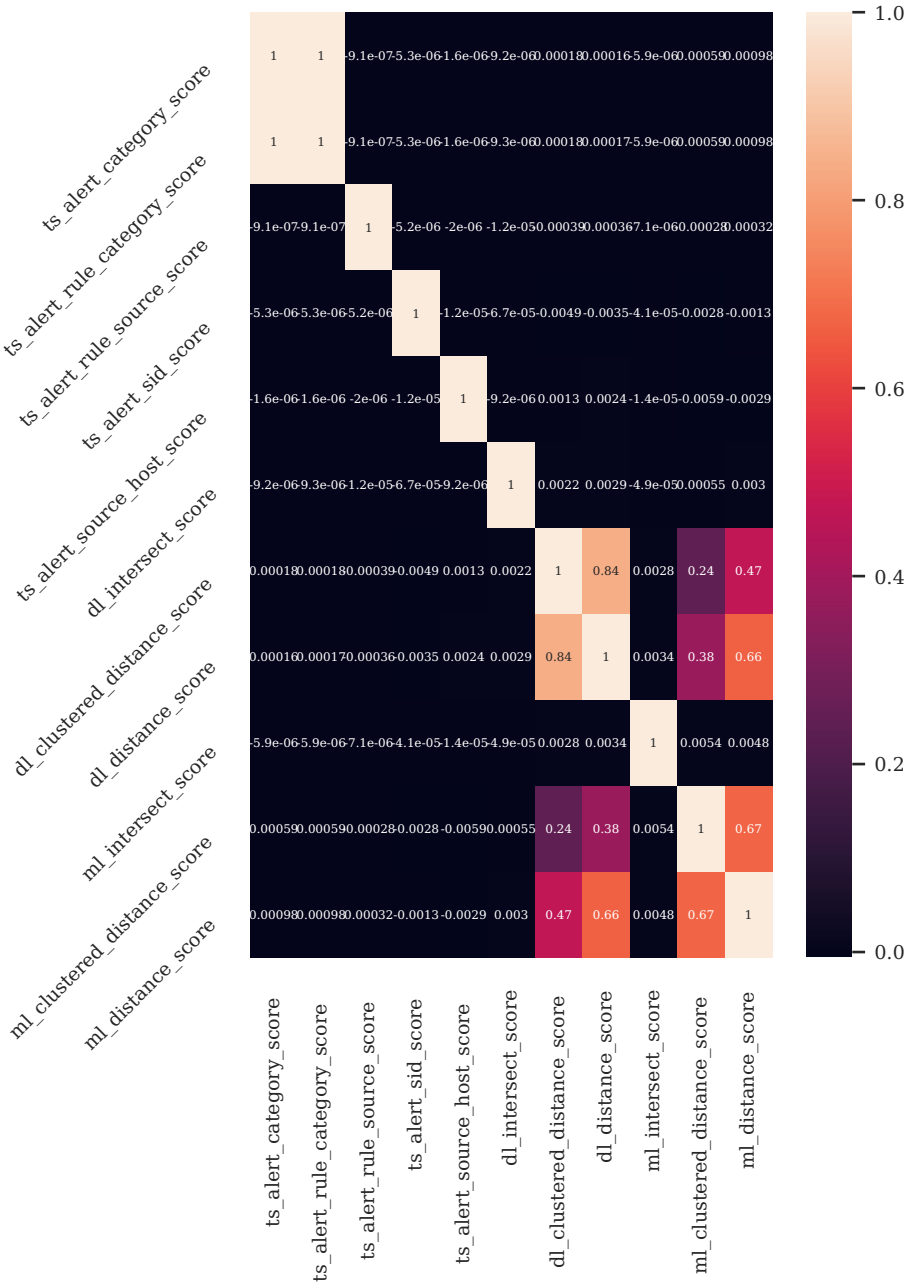

Figure 6.14: Shows the correlation between all columns for all values from the outlying alerts.

Manual evaluation	Outlier prediction: Outlier	Outlier prediction: Non-Outlier	Outlier prediction: Sum
True positive	18	6	24
False positive	7	3	10
Sum	25	9	34

Table 6.4: Confusion matrix shows how the different manual evaluations divide into the outliers.

Looking at the scores for each class of the evaluated alerts, False and True positives, we get the correlation as shown in Figure 6.15 for false positives, Figure 6.16 for true positives, and the distributions of scores as shown in Figure 6.17 for false positives and Figure 6.18.

The correlation plots shows similar correlations between the clustered and non-clustered distance scores both for PCA ("ml_clustered_distance_score", "ml_distance_score") and for the autoencoder ("dl_clustered_distance_score", "dl_distance_score"), similar to the correlations seen in section 6.2. Another interesting observation, a lack of correlation this time, is that "ts_alert_category_score" and "ts_alert_rule_category_score" had perfect correlation in all three correlations from section 6.2, while now having barely any correlation. This could indicate an issue in the processing or that the much smaller datasets of false and true positives might be overridden by the much larger dataset of unclassified alerts.

There seems to be a difference between the max value for some of the features from the distribution plots, Figure 6.15, Figure 6.16, Figure 6.17, and Figure 6.18, though this does not appear solely for the true-positives advantage. The feature "ts_alert_source_host_score" appears with a single value higher in the false-positive set than those in the true positive set, while for most other features where there is a difference, the true positives have a higher max value than in the false-positive set. This does not confirm significant differences between the sets' max value, but it might indicate that "ts_alert_source_host_score" is not a valuable feature for this process. There is also little confidence in these observations as the sample size is small.

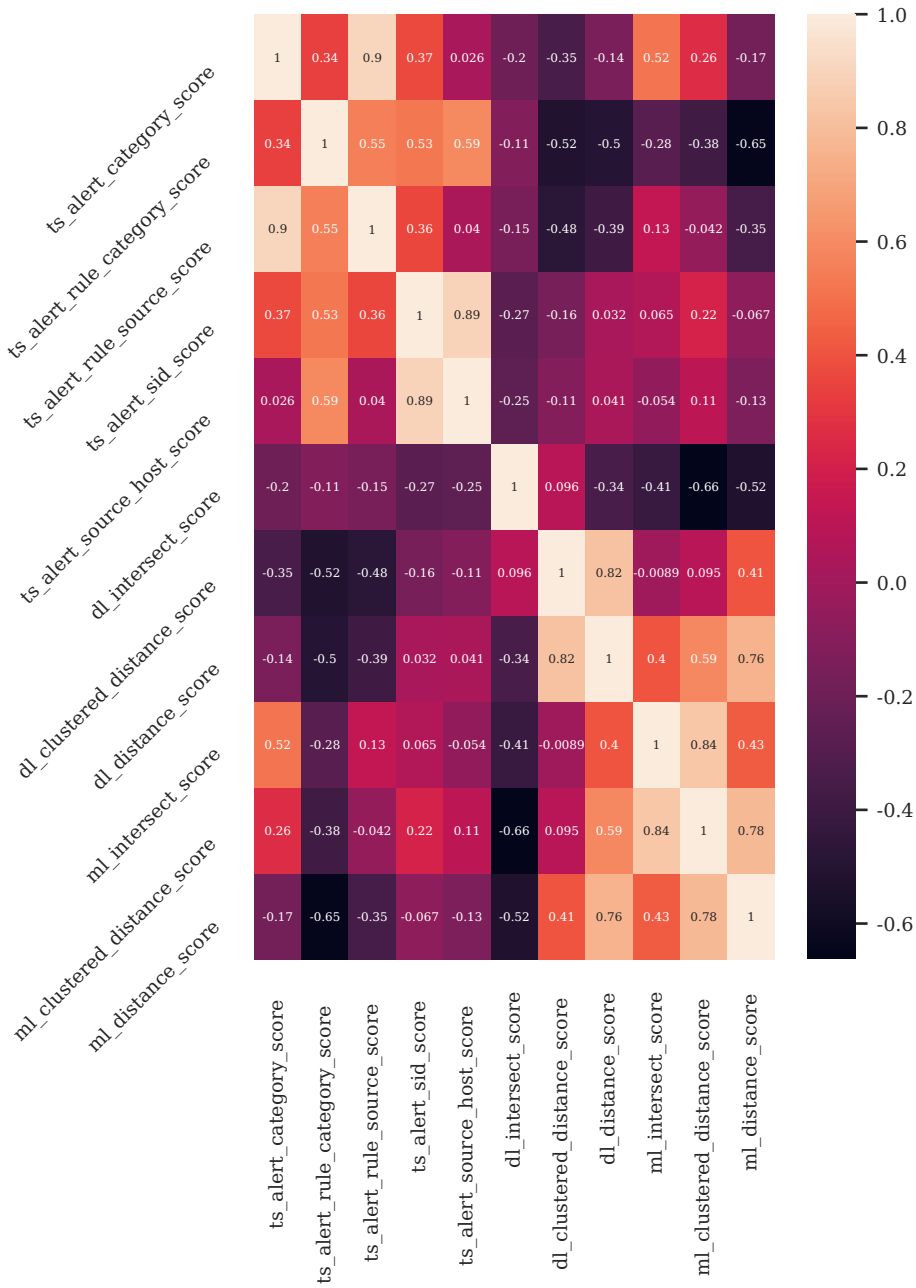

Figure 6.15: Shows the correlation between columns for the alerts manually classified as false positives.

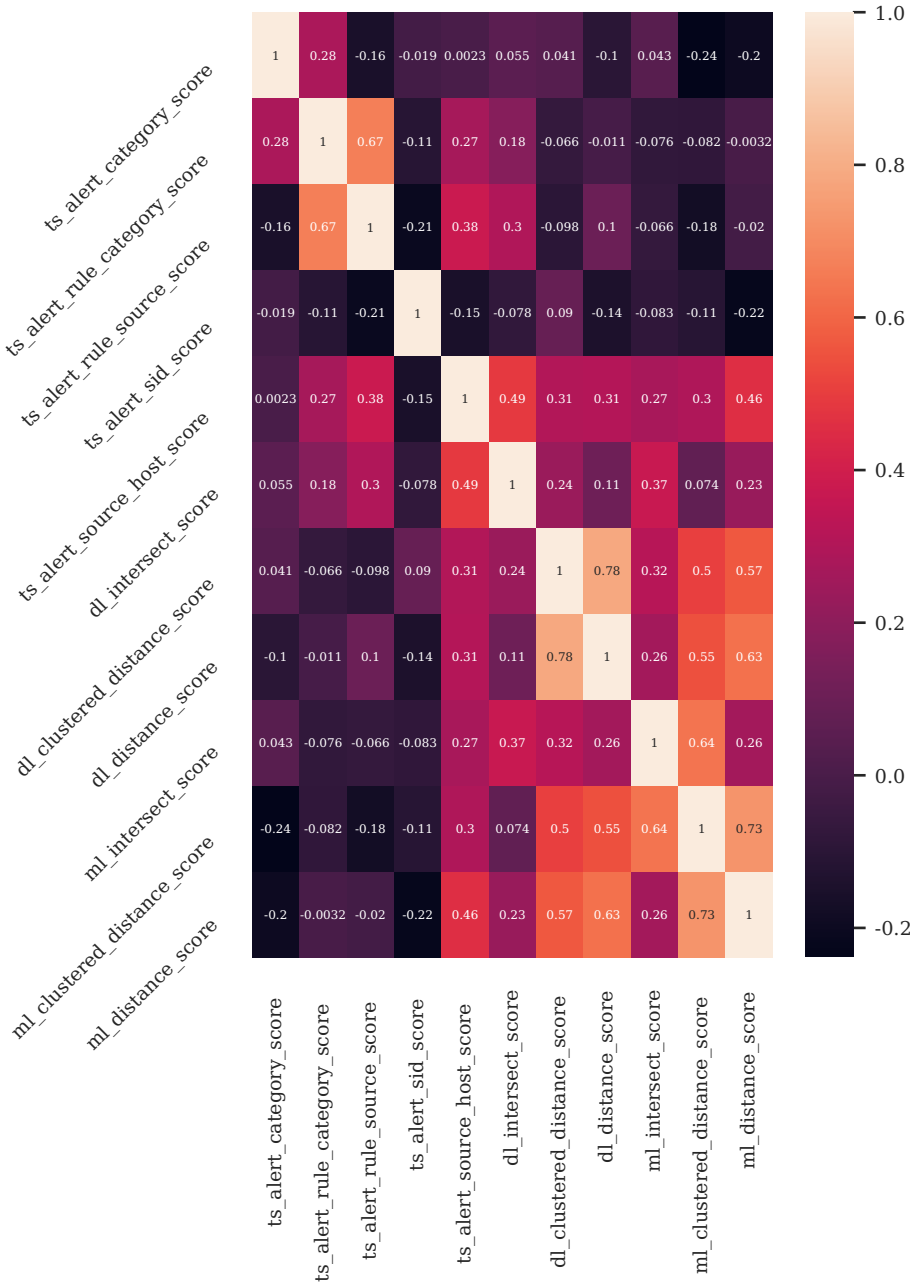

Figure 6.16: Shows the correlation between columns for the alerts manually classified as true positive.

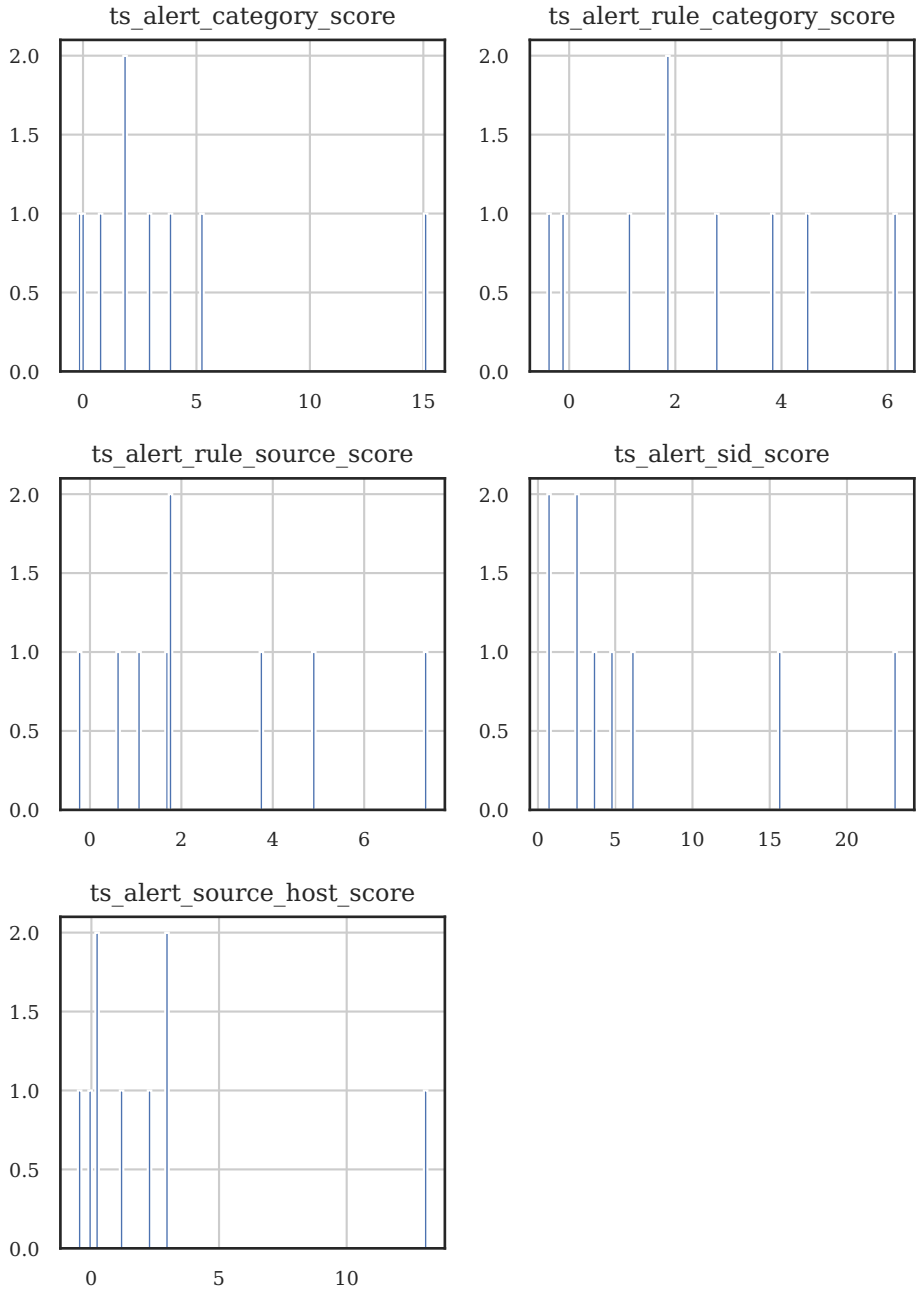

Figure 6.17: Shows the distribution of columns for the alerts manually classified as false positives.

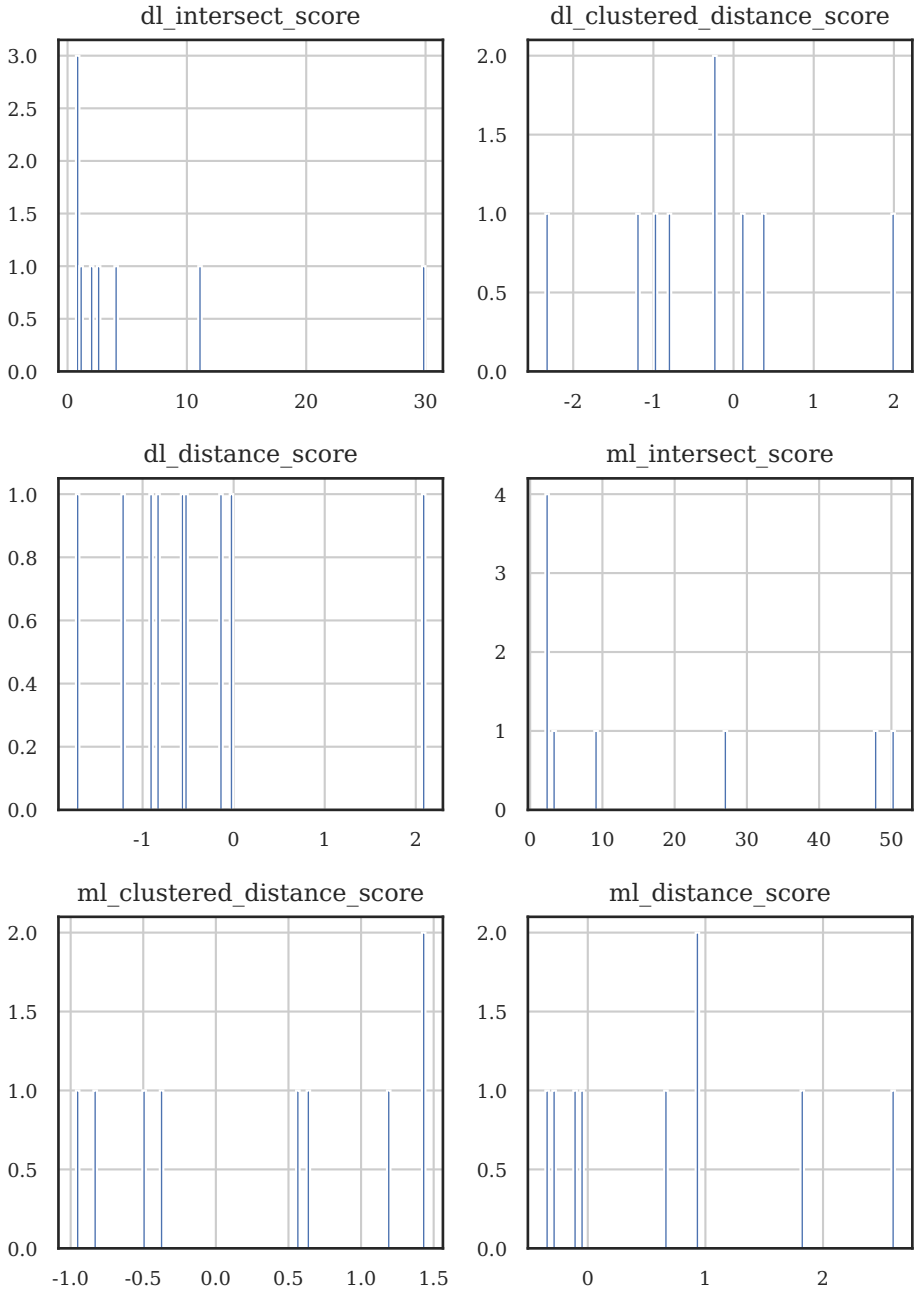

Figure 6.17: Shows the distribution of columns for the alerts manually classified as false positives (cont.).

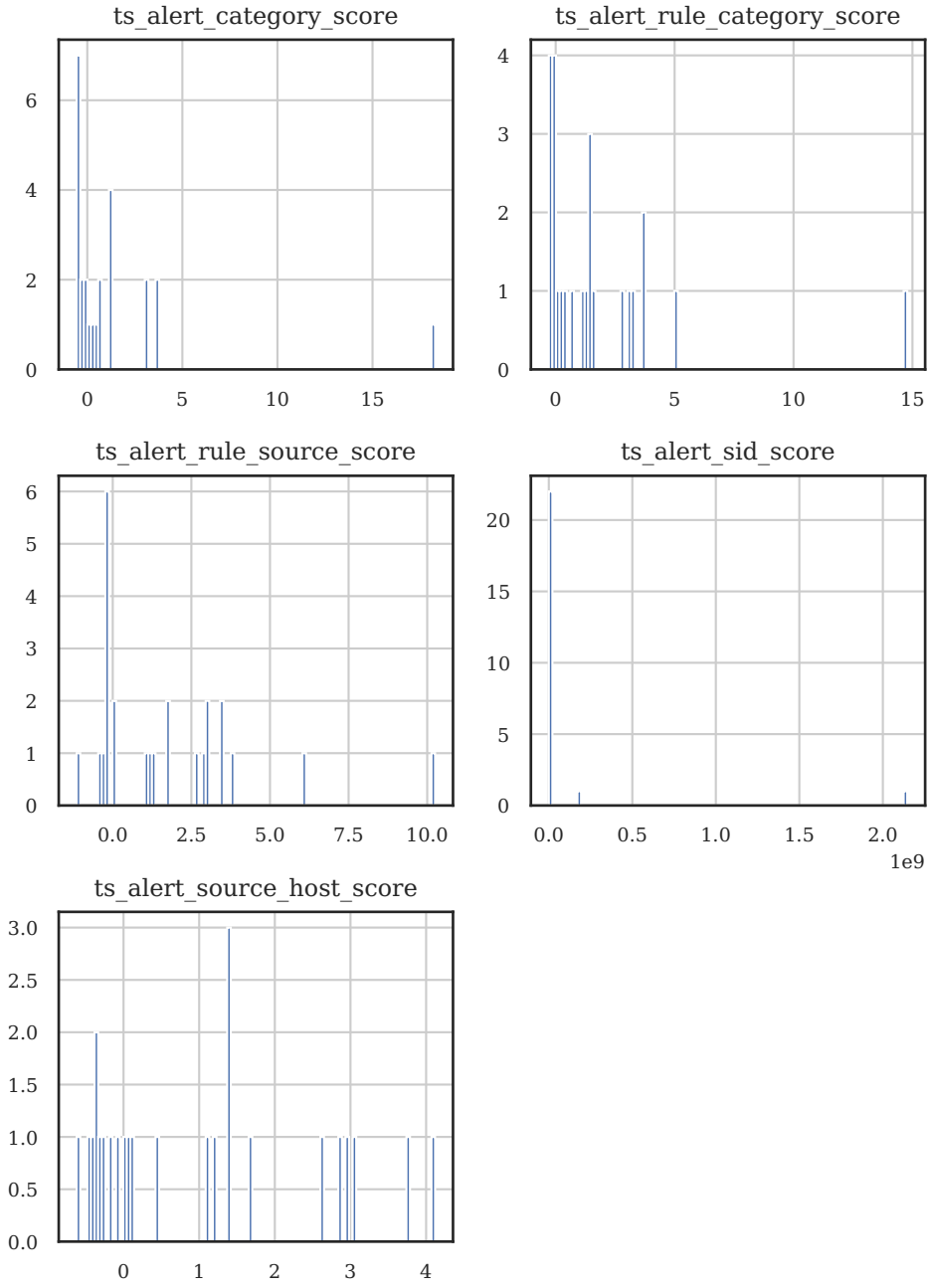

Figure 6.18: Shows the distribution of columns for the alerts manually classified as true positive.

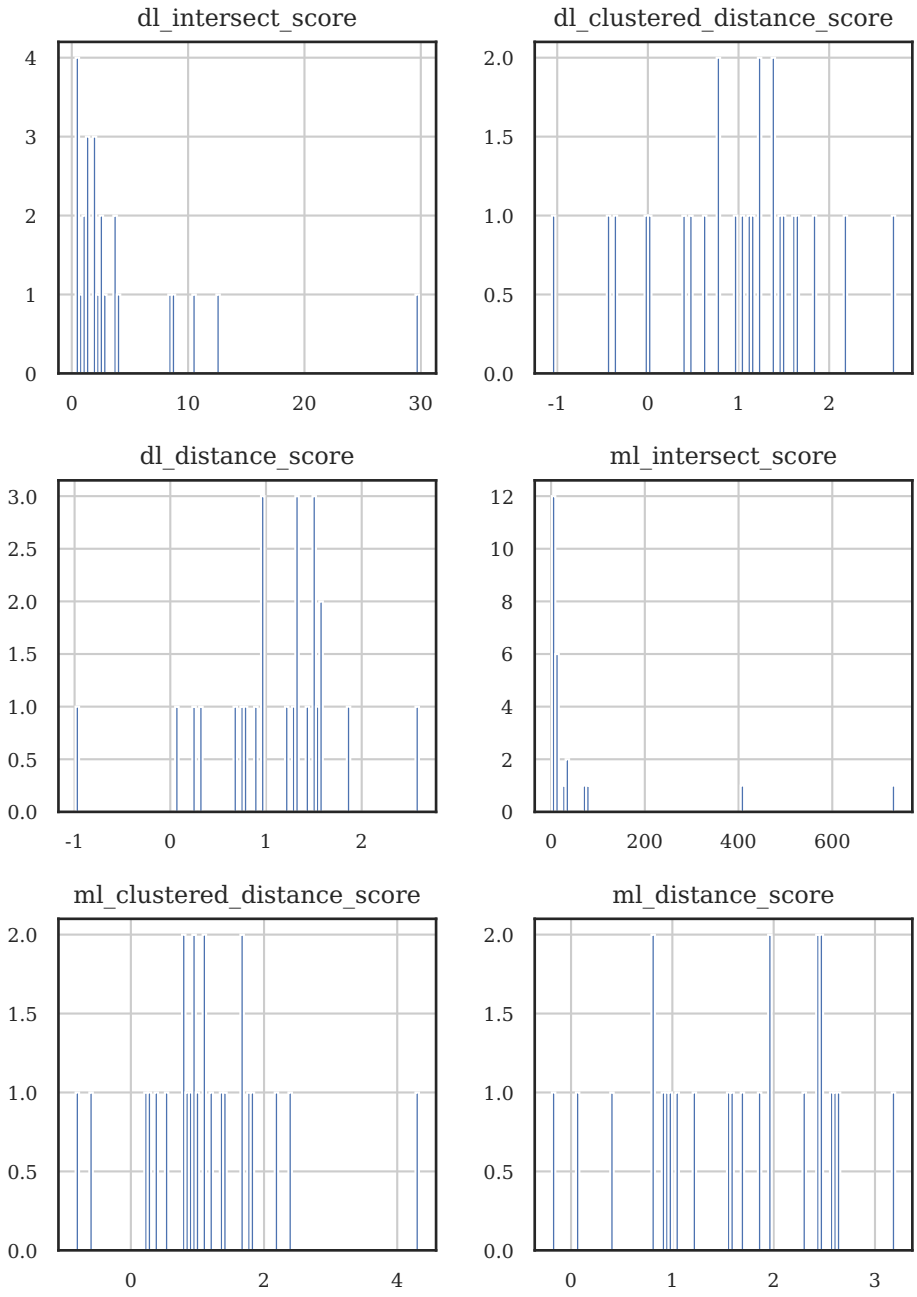

Figure 6.18: Shows the distribution of columns for the alerts manually classified as true positive (cont.).

Chapter 7
Discussion

The problem description contains a rough outline of the processes in this thesis. The overall goal was to produce a feed of actionable data from unclassified alerts out of Sikt's NIDS based on data science methods. First, we cover the objectives from the Problem Description in section 7.1, then Future Works in section 7.2.

7.1 Fulfillment of Research Objectives

The first step of the process, or sub-goal as we consider it, was to identify auxiliary data sources. The hope is that this will provide additional features for the alerts, improving the automatic evaluation of the alerts. The identified data sources are presented in section 4.1. Even though this thesis was limited to the already enriched data sources, it would certainly be interesting to look at additional or different enrichments. If the scope changes to, for instance, look at presumed attackers, one might sort out brute force attackers who have contacted the honeypots. An IP that regularly signs in to a monitored application with a single user or is known to be a NAT gateway at a campus could likewise be filtered out as likely legitimate traffic.

The second sub-goal was to engineer and normalize the NIDS data. One error in this process was that the normalization happened before we split the train, validation, and test data sets. As such, the data sets have affected each other[MMB+21]. Specifically, for each column, the highest value of the whole dataset became one while the lowest became zero. So if the highest value is in the training set, then the test data gets skewed down, as its max value would be smaller than one. This limitation also affects the use of new datasets. A new dataset with a higher max-value or a lower min-value for any of the columns would be outside the range required by the autoencoder. So any new datasets would require normalizing the whole dataset again and training the autoencoder again.

Another issue was that the source data store of the alerts did many changes

to the data structure, such as flattening lists, which made it harder to process the alerts. We worked around this issue, but it wasted unnecessary time. The final point that we will cover about data engineering is that many of the columns were in another format than what the autoencoder and PCA could use. It would increase the information available to the algorithms if the non-numerical features were available too, but we had to skip this due to time constraints.

The final sub-goal was to create classifiers based on machine learning, deep learning, and statistical inference. We did not complete this goal fully as we only made a single classifier based on the scores. Other classifiers could be based on subsets of the 11 different scores. The causes for this not being completed is a lack of time and a lack of manually classified alerts. The latter point is connected to the overall goal of this thesis, which was to have a classifier that could be evaluated with regards to the recall of actionable data, preferably with high precision. These observations are statistically insignificant due to the limited number of manually classified alerts. The validity of these observations outside of this thesis can not be claimed. Any attempt to split the classifier into smaller, simpler classifiers, building upon fewer scores, would only be guesswork.

The previously covered sub-goals make the basis for the primary goal of this thesis, creating classifiers that can sort out a feed of alerts that have a higher concentration of actionable data, with limited false positives. This can be considered as partly achieved. We created only one classifier, but it was not feasible to evaluate the recall or precision as the small set of manually evaluated alerts did not represent the whole dataset, and the actual distribution of the alerts is unknown. The only conclusion one can draw is that the outlier set was significantly smaller than the source set, about one-fifth. Most of the manually classified alerts went into the smaller set regardless of their classification. It would be preferable to do a root cause analysis to identify why these were all grouped, but again time was limited.

After the evaluation, another set of outliers was suggested, based on the outputs from the PCA and Autoencoder. We did not have time to use these, although it would be interesting to see the results, especially when compared to the results after K-Means.

7.1.1 Manual classification of alerts

Manually classifying alerts was necessary to evaluate the results. The methodology around gathering classifications could probably be a master's thesis by itself, looking into how one should present the alerts and which classification options to have for a consistent, unbiased classification if that is at all possible.

As mentioned in subsection 5.5.1, there was some criticism of the possible

classifications by the evaluators. The main complaint was that the set of classes was different from the ENISA Reference Incident Classification Taxonomy and not compatible with that taxonomy either. The ENISA taxonomy is one that the evaluators were more familiar and comfortable with, and they would likely have managed to classify more alerts if the ENISA taxonomy was used.

Other issues with getting the manually evaluated alerts were that the evaluators could freely select from the list of alerts. The list was sorted by newest first which skewed the manually classified alerts to the end of the timeframe. The evaluators also skipped some of the more demanding, possibly more valuable alerts and thereby skewed the classifications. Forcing a random sampling of the alerts would likely yield a better distribution of the classified alerts, possibly at the cost of fewer classifications getting done. Another option, if time had permitted, would have been to pick several alerts from the outliers and the non-outliers, then randomly shuffle them, and ask for those specific alerts to be classified. This could have distributed the classifications better.

7.2 Future works

Many ideas, suggestions, and challenges have been seen throughout this thesis. The following section discusses some ideas that were too large or outside this thesis's scope.

Starting with the maybe most obvious future work, repeating this process once again on an entirely new or larger dataset would likely give further indications for the feasibility of this process, keeping in mind that this new dataset would also need labels for the final evaluation. The end of the used dataset was August 31st, 2021, which means that six months of new data is already available at the time of this thesis's submission. It could be interesting to use the process on a dataset from a different organization to see how, if at all, this process would hold up in, for instance, a corporate network.

It has already been covered that an expanded set of labeled data would help evaluate the outliers from this thesis. However, another advantage of a more extensive collection of labeled data would be the possibility of supervised training classifiers. This could be added at different stages in the current process. Another way could be to take samples from both the outlier and non-outlier populations and classify them, leading to insights into the actual distribution of true and false positives.

How one should classify alerts is another topic that should be looked further into, regarding everything from the classification taxonomy, the user interface for the classification, and the classifications' day-to-day use, to the classifications' bias, distribution, and inaccuracy. Some of the related works, such as

Vaarandi[Vaa21], used groupings of alerts before the classification, possibly leading to higher classification rates but at the cost of higher inaccuracies if the grouping is invalid.

Finally, we would suggest that anyone with a large number of alerts, especially those who get new alerts regularly, incorporate a process for labeling these alerts. The labeled alerts would also allow other methods to be used, and they could be validated more easily. Having the labeling process as part of the typical workflow can also identify shared features of the classes that can be used to optimize the automated classification.

Chapter 8
Conclusion

This thesis has managed to represent the alerts in a significantly compressed form, both through PCA and the autoencoder, in a reproducible way which would allow for new data sets and likely for larger data sets. The use of statistical outlier detections allows unlabeled data to be split and classifiers to be built using sets of outlier detectors.

A data pipeline for the processing was also created as part of this thesis. We can reuse it in whole or in parts for later processing. The pipeline did produce a feed of outlier alerts which was the primary goal of this thesis. One recommendation for Sikt to likely get better performance, and better knowledge about the performance, is to create a process for labeling at least some of the alerts.

The algorithmic classification of alerts would significantly improve the manual classification in use today. While this thesis shows some promising results regarding this automation, there are still challenges surrounding the classification taxonomy and the lack of manual classification, though these are surpassable challenges.

The possible benefits of automation are pretty significant. Reducing the number of alerts to evaluate manually frees up time to investigate the remaining alerts better and improve security through previously not prioritized ways.

References

[AdSMZJ17] Alexandre Aguiar Amaral, Leonardo de Souza Mendes, Bruno Bogaz Zarpelão, and Mario Lemes Proença Junior. Deep IP flow inspection to detect beyond network anomalies. *Computer Communications*, 98:80–96, January 2017.

[Ait13] P. Aitken. Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information. Technical Report 7011, September 2013.

[B+20] Adrian Garcia Badaracco et al. SciKeras. https://www.adriangb.com/scikeras, 2020.

[BBK16] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. A multi-step outlier-based anomaly detection approach to network-wide traffic. *Information Sciences*, 348:243–271, June 2016.

[BC19] Natã M. Barbosa and Monchu Chen. Rehumanized Crowdsourcing. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12. ACM, May 2019.

[BD16] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer International Publishing, September 2016.

[BJS15] Przemysław Bereziński, Bartosz Jasiul, and Marcin Szpyrka. An entropy-based network anomaly detection method. *Entropy*, 17(4):2367–2408, April 2015.

[BK17] Sunny Behal and Krishan Kumar. Detection of DDoS attacks and flash events using novel information theory metrics. *Computer Networks*, 116:96–110, April 2017.

[C+15] François Chollet et al. Keras. https://keras.io, 2015.

[CBG12] Carlos A. Catania, Facundo Bromberg, and Carlos García Garino. An autonomous labeling approach to support vector machines algorithms for network traffic anomaly detection. *Expert Systems with Applications*, 39(2):1822–1829, February 2012.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection. *ACM Computing Surveys*, 41(3):1–58, July 2009.

[CFSD90] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). Technical Report 1157, May 1990.

[CGPP11] C. Callegari, S. Giordano, M. Pagano, and T. Pepe. Combining sketches and wavelet analysis for multi time-scale network anomaly detection. *Computers & Security*, 30(8):692–704, November 2011.

[CPVGTMF16] José Camacho, Alejandro Pérez-Villegas, Pedro García-Teodoro, and Gabriel Maciá-Fernández. PCA-based multivariate statistical network monitoring for anomaly detection. *Computers & Security*, 59:118–137, June 2016.

[CTH12] Chi Cheng, Wee Peng Tay, and Guang-Bin Huang. Extreme learning machines for intrusion detection. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, IEEE, June 2012.

[Das21] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2021.

[DT15] Jisa David and Ciza Thomas. DDoS attack detection using fast entropy approach on flow- based network traffic. *Procedia Computer Science*, 50:30–36, 2015.

[EKS17] David Ahmad Effendy, Kusrini Kusrini, and Sudarmawan Sudarmawan. Classification of intrusion detection system (IDS) based on computer network. In *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, pages 90–94. IEEE, IEEE, November 2017.

[FH18] Fahimeh Farahnakian and Jukka Heikkonen. A deep auto-encoder based approach for intrusion detection system. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 178–183. IEEE, IEEE, February 2018.

[Fir14] FireEye. How many alerts is too many to handle? https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html, 2014.

[Fou21] The Apache Software Foundation. *Parquet 2.9.0*, October 2021.

[FRC+18] Gilberto Fernandes, Joel J. P. C. Rodrigues, Luiz Fernando Carvalho, Jalal F. Al-Muhtadi, and Mario Lemes Proença. A comprehensive survey on network anomaly detection. *Telecommunication Systems*, 70(3):447–489, July 2018.

[FRP15] Gilberto Fernandes, Joel J.P.C. Rodrigues, and Mario Lemes Proença. Autonomous profile-based anomaly detection system using principal component analysis and flow analysis. *Applied Soft Computing*, 34:513–525, September 2015.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, November 2016. http://www.deeplearningbook.org.

[GMDS19] Mohammed Gharib, Bahram Mohammadi, Shadi Hejareh Dastgerdi, and Mohammad Sabokrou. Autoids: auto-encoder based method for intrusion detection system. *arXiv preprint arXiv:1911.03306*, November 2019.

[GPRR08] Giorgio Giacinto, Roberto Perdisci, Mauro Del Rio, and Fabio Roli. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. *Information Fusion*, 9(1):69–82, January 2008.

[Gut20] William F. Guthrie. Nist/sematech e-handbook of statistical methods (nist handbook 151), 2020.

[HB07] Mohamed Hamdi and Noureddine Boudriga. Detecting Denial-of-Service attacks using the wavelet transform. *Computer Communications*, 30(16):3203–3213, November 2007.

[HBB+18] Hanan Hindy, David Brosset, Ethan Bayne, Amar Seeam, Christos Tachtatzis, Robert Atkinson, and Xavier Bellekens. A taxonomy and survey of intrusion detection system design techniques, network threats and datasets. June 2018.

[HBB+20] Hanan Hindy, David Brosset, Ethan Bayne, Amar Kumar Seeam, Christos Tachtatzis, Robert Atkinson, and Xavier Bellekens. A Taxonomy of Network Threats and the Effect of Current Datasets on Intrusion Detection Systems. *IEEE Access*, 8:104650–104675, June 2020.

[HdV08] Mia Hubert and Stephan Van der Veeken. Outlier detection for skewed data. *Journal of Chemometrics*, 22(3-4):235–246, 2008.

[Hot33] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933.

[KFFS13] Yoshiki Kanda, Romain Fontugne, Kensuke Fukuda, and Toshiharu Sugawara. ADMIRE: Anomaly detection method using entropy-based PCA with three-step sketches. *Computer Communications*, 36(5):575–588, March 2013.

[KGZ15] Amin Karami and Manel Guerrero-Zapata. A fuzzy anomaly detection system based on hybrid PSO-kmeans algorithm in content-centric networks. *Neurocomputing*, 149:1253–1269, February 2015.

[KSB21] Phanindra Reddy Kannari, Noorullah C. Shariff, and Rajkumar L. Biradar. Network intrusion detection using sparse autoencoder with swish-PReLU activation Model. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13, March 2021.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LCC+18] Ying Lin, Zhengzhang Chen, Cheng Cao, Lu-An Tang, Kai Zhang, Wei Cheng, and Zhichun Li. Collaborative alert ranking for anomaly detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, October 2018.

[LCD04] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM Computer Communication Review*, 34(4):219–230, August 2004.

[LGZ+19] Zhiqiang Liu, Mohi-Ud-Din Ghulam, Ye Zhu, Xuanlin Yan, Lifang Wang, Zejun Jiang, and Jianchao Luo. Deep Learning Approach for IDS. In *Advances in Intelligent Systems and Computing*, number China, pages 471–479. Springer Singapore, December 2019.

[LRU20] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, January 2020.

[MCLO11] Johan Mazel, Pedro Casas, Yann Labit, and Philippe Owezarski. Sub-space clustering, inter-clustering results association & anomaly correlation for unsupervised network anomaly detection, 2011.

[MLL+18] Erxue Min, Jun Long, Qiang Liu, Jianjing Cui, Zhiping Cai, and Junbo Ma. SU-IDS: A semi-supervised and unsupervised framework for network intrusion detection. In *Cloud Computing and Security*, pages 322–334. Springer International Publishing, 2018.

[MMB+21] Nikhil Muralidhar, Sathappah Muthiah, Patrick Butler, Manish Jain, Yu Yu, Katy Burne, Weipeng Li, David Jones, Prakash Arunachalam, Hays 'Skip' McCormick, and Naren Ramakrishnan. Using antipatterns to avoid mlops mistakes. June 2021.

[PCB+04] M. Lemes Proença, C. Coppelmans, M. Bottoli, A. Alberti, and L. S. Mendes. The hurst parameter for digital signature of network segment. In *Telecommunications and Networking - ICT 2004*, pages 772–781. Springer, Springer Berlin Heidelberg, 2004.

[PT05] Tadeusz Pietraszek and Axel Tanner. Data mining and machine learning—Towards reducing false positives in intrusion detection. *Information Security Technical Report*, 10(3):169–183, January 2005.

[PVG+12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research (2011)*, 12:2825–2830, January 2012.

[RH11] Peter J. Rousseeuw and Mia Hubert. Robust statistics for outlier detection. *WIREs Data Mining and Knowledge Discovery*, 1(1):73–79, January 2011.

[RJM+21] Jeff Reback, Jbrockmendel, Wes McKinney, Joris Van Den Bossche, Tom Augspurger, Phillip Cloud, Simon Hawkins, Gfyoung, Matthew Roeschke, Sinhrks, Adam Klein, Terji Petersen, Jeff Tratner, Chang She, William Ayd, Patrick Hoefler, Shahar Naveh, Marc Garcia, Jeremy Schendel, Andy Hayden, Daniel Saxton, J. H. M. Darbyshire, Richard Shadrach, Marco Edward Gorelli,

Fangchen Li, Matthew Zeitlin, Vytautas Jancauskas, Ali McMaster, Pietro Battiston, and Skipper Seabold. *pandas-dev/pandas: Pandas 1.3.4*, October 2021.

[RM05] Lior Rokach and Oded Maimon. Clustering methods. In *Data Mining and Knowledge Discovery Handbook*, pages 321–352. Springer-Verlag, 2005.

[Roc15] Matthew Rocklin. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In Kathryn Huff and James Bergstra, editors, *Proceedings of the 14th Python in Science Conference*, pages 130–136. SciPy, 2015.

[Rou87] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, nov 1987.

[SBK16] Basant Subba, Santosh Biswas, and Sushanta Karmakar. A neural network based system for intrusion detection and attack classification. In *2016 Twenty Second National Conference on Communication (NCC)*, pages 1–6. IEEE, IEEE, March 2016.

[SNPS18] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. A deep learning approach to network intrusion detection. *IEEE transactions on emerging topics in computational intelligence*, 2(1):41–50, February 2018.

[TB99] Michael E. Tipping and Christopher M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, February 1999.

[Tho53] Robert L. Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, dec 1953.

[Vaa21] Risto Vaarandi. A stream clustering algorithm for classifying network IDS alerts. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 14–19. IEEE, IEEE, July 2021.

[WGW17] Huiwen Wang, Jie Gu, and Shanshan Wang. An effective intrusion detection framework based on SVM with feature augmentation. *Knowledge-Based Systems*, 136:130–139, November 2017.

[WWZZ22] Shaoyu Wang, Xinyu Wang, Liangpei Zhang, and Yanfei Zhong. Auto-AD: Autonomous Hyperspectral Anomaly Detection Network Based on Fully Convolutional Autoencoder. *IEEE Trans. Geosci. Remote Sens.*, 60:1–14, 2022.

[XW05] R. Xu and D. WunschII. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[YJW07] Daniel S. Yeung, Shuyuan Jin, and Xizhao Wang. Covariance-matrix modeling and detecting various flooding attacks. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(2):157–169, March 2007.

[YRF19] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. Outlier detection. In *Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing - AIIPCC '19*. ACM Press, 2019.

Appendix A

Preprocessed column features

Column name	Datatype
#source_host_{institution}-mp	uint8
http.http_port	float64
alert.metadata.updated_at	datetime64[ns]
tls.notbefore	datetime64[ns]
tls.notafter	datetime64[ns]
@ingesttimestamp	datetime64[ns]
#repo	object
#source	object
@id	object
@timezone	object
alert.signature	object
alert.metadata.policy	object
community_id	object
destination.ipreg.comment	object
destination.ip	object
destination.ip_misp	object
dns.query	object
dns.query.rrname	object
email.attachment	object
email.cc	object
email.from	object
email.to	object
email.url	object

Column name	Datatype
files[0].sha256	object
files[0].sha1	object
files[0].md5	object
files[0].filename	object
files[1].filename	object
flow_id	object
http.content_range.raw	object
http.hostname	object
http.http_response_body_printable	object
http.xff	object
http.url	object
http.http_response_body	object
http.redirect	object
http.http_request_body	object
http.http_request_body_printable	object
http.http_refer	object
labels.source	object
payload	object
payload_printable	object
rdp.server_supports	object
smb.client_dialects	object
smb.client_guid	object
smb.dialect	object
smb.server_guid	object
smtp.helo	object
smtp.mail_from	object
smtp.rcpt_to	object
source.ip_misp	object
source.ip_tor	object
source.ip	object
source.ipreg.comment	object
tunnel.dest_ip	object
tunnel.src_ip	object
tls.ja3s.hash	object

Column name	Datatype
tls.ja3s.string	object
tls.ja3.string	object
tls.ja3.hash	object
tls.sni	object
tls.version	object
tls.subject	object
tls.fingerprint	object
tls.issuerdn	object
tls.serial	object
TLP	category
alert.action	category
alert.signature_id	category
alert.category	category
alert.severity	category
alert.metadata.affected_product	category
alert.metadata.attack_target	category
alert.metadata.cve	category
alert.metadata.deployment	category
alert.metadata.former_category	category
alert.metadata.impact_flag	category
alert.metadata.malware_family	category
alert.metadata.mitre_tactic_id	category
alert.metadata.mitre_tactic_name	category
alert.metadata.mitre_technique_id	category
alert.metadata.mitre_technique_name	category
alert.metadata.service	category
alert.metadata.signature_severity	category
alert.metadata.tag	category
alert.metadata.performance_impact	category
app_proto	category
app_proto_expected	category
app_proto_orig	category
app_proto_ts	category
app_proto_tc	category

Column name	Datatype
destination.ipreg.name	category
destination.geo.country_iso_code	category
destination.geo.country_name	category
destination.geo.city_name	category
destination.geo.region	category
destination.ipreg.domain	category
dns.query.type	category
dns.query.rrtype	category
email.status	category
event_type	category
files[0].magic	category
files[0].state	category
files[1].state	category
http.protocol	category
http.http_method	category
http.http_user_agent	category
http.http_content_type	category
in_iface	category
labels.log_type	category
labels.nifi_node	category
labels.source_host	category
metadata.flowbits	category
proto	category
smb.command	category
smb.status_code	category
smb.status	category
source.ipreg.domain	category
source.ipreg.name	category
source.geo.country_iso_code	category
source.geo.country_name	category
source.geo.city_name	category
source.geo.region	category
tunnel.proto	category
alert.metadata.created_at	datetime64[ns]

Column name	Datatype
flow.start	datetime64[ns]
#humioBackfill	object
traffic.id	object
traffic.label	object
parent_id	object
command	object
filename	object
rdp.event_type	object
rdp.tx_id	object
rdp.protocol	object
rdp.cookie	object
ssh.server.software_version	object
ssh.server.proto_version	object
dns.answer.rrtype	object
dns.answer.type	object
dns.answer.version	object
dns.answer.qr	object
dns.answer.rrname	object
dns.answer.rcode	object
dns.answer.flags	object
dns.answer.id	object
flow.pkts_toclient	float64
flow.bytes_toclient	float64
flow.pkts_toserver	float64
flow.bytes_toserver	float64
packet_size_toclient	float64
packet_size_toserver	float64
packet_count_ratio	float64
packet_size_ratio	float64
stream	float64
http.status	float64
destination.port	float64
source.port	float64
vlan	float64

Column name	Datatype
alert.rev	float64
alert.gid	float64
@timestamp.nanos	float64
dns.query.id	float64
dns.query.tx_id	float64
tx_id	float64
files[0].size	float64
files[0].start	float64
files[0].end	float64
files[0].tx_id	float64
files[1].size	float64
files[1].tx_id	float64
http.length	float64
http.content_range.size	float64
http.content_range.start	float64
http.content_range.end	float64
icmp_code	float64
icmp_type	float64
smb.tree_id	float64
smb.session_id	float64
smb.id	float64
tunnel.dest_port	float64
tunnel.src_port	float64
tunnel.depth	float64
destination.geo.location.lat	float64
destination.geo.location.lon	float64
source.geo.location.lat	float64
source.geo.location.lon	float64
rule_source	category
rule_category	category
rule_name	object
standard_http_port	float64
recent_alert	float64
valid_tls	float64

Column name	Datatype
files[0].gaps	float64
files[0].stored	float64
files[1].gaps	float64
files[1].stored	float64
tls.session_resumed	float64

Table A.1: Table of columns in the preprocessed dataset

Appendix B

Results from Deep learning grid search

Encoding dimension	Hidden layers	Learning rate	Ranking	Mean test score	Std. test score
16	[64]	0.001	1	-0.165998	1.905004
24	[64]	0.001	2	-1.257419	3.976922
16	[64]	0.0001	3	-2.090995	4.294758
16	[32]	0.001	4	-2.274016	5.666658
16	[64, 32]	0.01	5	-3.673282	6.906616
32	[64]	0.001	6	-3.875934	9.149172
32	[64, 32]	0.01	7	-4.596101	8.692238
32	[64, 32]	0.001	8	-4.938465	10.955588
24	[64, 32]	0.01	9	-6.230541	12.018193
32	[]	0.0001	10	-6.325708	13.230967
16	[32]	0.01	11	-7.581906	15.217436
32	[64, 32]	0.0001	12	-9.464108	16.572367
32	[32]	0.01	13	-9.529388	19.276476
24	[32]	0.001	14	-15.660022	30.483318
16	[]	0.001	15	-16.122285	31.594243
24	[64, 32]	0.0001	16	-19.313878	25.117049
24	[32]	0.01	17	-22.128573	45.060381
24	[64]	0.01	18	-24.152763	31.849961
16	[]	0.0001	19	-29.104775	58.794986
16	[64, 32]	0.0001	20	-32.811528	66.011121
24	[]	0.0001	21	-36.685598	71.457139

Encoding dimension	Hidden layers	Learning rate	Ranking	Mean test score	Std. test score
24	[64, 32]	0.001	22	-60.618297	111.287029
24	[64]	0.0001	23	-66.696466	134.409492
32	[]	0.01	24	-71.468067	141.611194
16	[64]	0.01	25	-81.752602	163.462126
32	[]	0.001	26	-94.465566	190.240253
24	[32]	0.0001	27	-103.595391	208.118309
16	[64, 32]	0.001	28	-109.244383	139.725759
32	[32]	0.0001	29	-135.546500	272.376380
32	[64]	0.01	30	-160.112720	320.423461
24	[]	0.001	31	-394.743015	789.688911
16	[]	0.01	32	-475.501450	895.965637
32	[64]	0.0001	33	-1551.502418	3104.334806
16	[32]	0.0001	34	-3201.840068	6209.746174
32	[32]	0.001	35	-30404.570295	59876.614974
24	[]	0.01	36	-170463.923104	340781.559690

Table B.1: Results of the gridsearch on autoencoder parameters.

Appendix C

Silhouette scores for the number of clusters

# of clusters	Autoencoder silhouette score	PCA silhouette score
2	0.2138016912354948	0.9171638278900758
3	0.23394384624981876	0.9386463222293818
4	0.2463130765432287	0.9479645010668418
5	0.27320828298051053	0.9584125499911166
6	0.2873570402814553	0.9627329480018663
7	0.31316207065619694	0.9672663254196728
8	0.32706191893495745	0.968914281292246
9	0.35356989549730305	0.9679339067743158
10	0.3748460491885769	0.9689633274611283
11	0.3967233523236475	0.9699899196361742
12	0.4154962393302681	0.9716602392495362
13	0.4297071505261106	0.9727944314884122
14	0.4468431742273549	0.973617705117332
15	0.4550070859825622	0.9746974017760884
16	0.4570741531332751	0.9756323749814796
17	0.4815845625316916	0.9762282650245693
18	0.4925566253429344	0.9765850977681703
19	0.5031645034414371	0.9768783180330085
20	0.5141385843371551	0.9777985589555297
21	0.5228294437481573	0.9779937765630465
22	0.5301115632207679	0.9783475786545442
23	0.5391109079277132	0.9779490787980156

# of clusters	Autoencoder silhouette score	PCA silhouette score
24	0.5507689448410427	0.9814132642441666
25	0.5605811703328178	0.9792239371854449
26	0.549144456833908	0.9785755214760392
27	0.5579710153007769	0.9826725908685388
28	0.5745833645669183	0.9831701005914942
29	0.5783541195934765	0.9826219037722687
30	0.5811029712781661	0.9832499797381794
31	0.5921428423749202	0.9835485682004098
32	0.5764927596978244	0.9832071042179379

Table C.1: Silhouette score from the search for number of clusters.

Didrik Frimann Barroso Koren

Improving security visibility with event correlation

NTNU
Norwegian University of
Science and Technology