Explainable AI for Network Traffic Analytics

Srinivas Pokuri (Student ID: 939948)

Liverpool John Moore's University, Liverpool, UK

Research Supervisor : Snehansu Sekhar Sahu

DEDICATION

I owe it to my friends and family members who have encouraged me to do this Thesis for better exposure to problem solving by researching the existing wealth of Literature published by various talented researchers over the years.

ACKNOWLEDGEMENTS

I thank UpGrad Research mentors for providing necessary help in identifying my research topic and providing necessary insights and support on how research is to be carried out.

ABSTRACT

Classifying Network traffic is an important prerequisite for Network applications. Despite research in this area, the progress has been moderate owing to the availability of real representative datasets during various security concerns. (Barut, n.d.) makes us available a new novel dataset and curated datasets of Network flows aggregated features for each of various classes of network flows. There are multiple sets of datasets that cater to malware detection at multiple levels and non-VPN network traffic details specifying the kind of network traffic it is. Here each observation in the dataset is a summary of the first 200 network packets from each of the network sessions, where each network session is from one pcap file. In this Thesis, we reviewed three models XGBoost, Random Forest, Neural Networks, and compared their metrics and their model explanations. Model agnostic explainable AI methods like LIME, SHAP are applied to understand the features that lead to Malware identification and Network Traffic classification. Standard Python libraries like Keras and Sklearn are used for building the models and libraries like LIME and SHAP were used for explaining the models. The Explainability study has been confined to high-level malware detection and network classification owing to the huge number of scenarios that are possible. XGBoost with default parameters without any class balancing proved to be a better model for Malware Detection and Network Traffic Identification. Class balancing the dataset with SMOTE enhanced the F1-Score of the model marginally for binary classification problems like Malware detection, but not for Network Traffic classification which is a multi-class classification problem. SHAP provided better explanations over LIME for all the models including the XGBoost which outperformed the other models. The contribution of this paper is in identifying that the model based on XGBoost performs better for malware detection and network classification, and a comparative study to explain the models with LIME and SHAP explainable AI concluded that SHAP provides better explanations over LIME.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ABBREVIATION	DESCRIPTION
AUC	Area under the curve
CNN	Conventional Neural Networks
DeepLIFT	Deep Learning Important Features
DNN	Deep Neural Networks
GAN	Generative Adversarial Network
LIME	**L**ocal surrogate **I**nterpretable **M**achine learning **E**xplanation
LRP	**L**ayer-wise **R**elevance **P**ropagation.
PDP	Partial Dependent Plot
ROC	Receiving Operating Curve
RELU	Rectified Activation Unit
SVM	Support Vector Machine
VAE	Variational Auto-encoders

1. INTRODUCTION

1.1. Background

Classifying network traffic is an important prerequisite for Network applications. Classifying network traffic is a very important task for Internet Service Providers and Network security teams in an organization. This is needed not only to understand the traffic flows to maintain the systems at optimal health but provide the first line of defense against any malware and exploitation of sensitive information in organizations. Despite extensive research in this area in classifications, the progress on the explainability of the models has been moderate owing to the availability of real representative datasets and various security concerns. (Barut, n.d.) makes available a new novel dataset and curated datasets of Network flows with all features aggregated for each of various classes of network flows.

Several advances have been made in different domains using XGBoost and Neural networks that have achieved pathbreaking accuracies in multi-class classifications. The main drawback of going with Deep neural networks is that they are predominantly backbox models and is hard for humans how the model arrives at those decisions. Understanding how these decisions are made is important to understand to build future-proof systems that are resilient, reliable, and adaptive.

In this Thesis, we review multiple shallow and Deep Learning models for Malware Detection and Network Traffic Classification and compare their metrics and explanations to provide the inferences/conclusions from this study.

1.2. Problem Statement

In domains where the implications of misclassification of data have significant financial and operational implications, the models that are deployed must have the confidence of people and understand why certain decisions are being made by the model. For use cases related to Network traffic analysis, this is more important as the margin of error is quite low. In this paper, we identify a better model for network traffic classification and a better explanation for the machine learning models.

1.3. Aim and Objectives

The aim of this thesis to find a better model for malware detection and network traffic classification and explain it better after comparing it with different explainable AI methods.

1.3.1. Aim

Build models to classify the network traffic for malware detection and general network traffic and find a better model for these tasks. Find an explainable AI method that better suits to explain these models.

1.3.2. Objective

Network Traffic Analytics for Malware detection and traffic classification is generally a tough problem to solve considering that there are very few real datasets available for model training. The network flows dataset is that is made available for the NetML challenge is leveraged for this purpose. This is made up of a novel NetML and curated CICIDS2017 dataset for malware detection and the non-vpn2016 dataset for network classification(Barut, n.d.).

The objective is to classify the network flows for malware detection also classifying the non-VPN network traffic and explain the models using the explainable AI methods.

Network Traffic Classification:

- Classify the Malwares at top-level (malware/benign) on the NetML dataset.
- Classify the Malwares at top-level (malware/benign) on the CICIDS2017 curated dataset.
- Classify the non-VPN network traffic at the top level on the non-VPN dataset.

For all the above three use cases, class imbalances handling might be needed. Classification is done using various models suitable for Network traffic analysis like XGBoost/Random Forest and Deep learning for recommending an appropriate model for these classifications.

Model explainability:

- Explain the above models using various model agnostic explainable AI methods like LIME, SHAP, etc.
- In the case of the Deep Learning model, as needed explanations using the Deep LIFT and Integrated explainable Methods are considered. Verify if these explanations are as good as the explanations for models like Random Forest/XGBoost.

1.4. Research Questions

The specific questions that this thesis tries to find the answers are:

- Is Neural Networks/Deep Learning the best method to classify the network traffic with high accuracy when using the aggregated NetML flows?
- Can Explainable AI methods convincingly explain the above Networks Traffic classification by Neural networks and other shallow methods like XGBoost and Random Forest?

1.5. Scope of Study

The study is confined to the following:

- The dataset is leveraged as is. It is assumed the dataset provided is curated.
- The scope of the study is confined to a maximum of three models Random Forest/XGBoost and Neural Networks based classification.
- While doing the modeling all provided features are considered. No upfront manual selection of the features based on the domain knowledge is considered.
- A maximum of four Explainable AI methods (LIME, SHAP, Integrated Gradients, DeepLIFT) is considered as needed to explain the explanations for the model.
- For each model, explain the local predictions for a set of a few random observations.
- The ML models and Explanation of the models are confined to top-level malware detection and top-level non-VPN network classification owing to many possible scenarios and time.

1.6. Significance of Study

Compared to other areas, Explainable AI has not been applied so exhaustively in Network Traffic Analysis. Many critical network applications are based on good classification network traffic but we should be able to understand why such a classification is being made (Melis et al., 2018; Morichetta et al., 2019). E.g., being able to track and understand if and why there is any malware attack/alarm or tracing the network bandwidth bottlenecks because of a specific application running on the network.

In this thesis the planned contribution is three-fold:

- Use a relatively new/novel dataset and curated datasets made available by the Challenge for building a more accurate model for malware detection and traffic classification (Barut, n.d.).
- To increase the human confidence in the model using explainable AI methods.

- Since the black box models when applied to computer security can be easily evaded by small changes in the content (Calleja et al., 2018). Identify potential vulnerabilities of linear/non-linear models against adversarial manipulations (Melis et al., 2018).

1.7. Structure of Study

The structure of this thesis is as follows:

Chapter 1 presents the background of the study and why Network classification and Malware detection is important and how having a robust understandable explainable AI is important. In section 1.2, the problem statement talks about the potential use cases for Malware detection and Network Traffic classification. The Aims and objectives are discussed in section 1.3, which are primarily to identify the best model and explain it. In section 1.4 the research questions whether neural works can perform better than the other shallow models and whether it can effectively explain the models are mentioned. Section 1.5 specifies the scope of the study while section 1.6 presents the significance of the study. Section 1.7 is a brief of all the structure of this thesis report.

Chapter 2 focuses on the literature survey. Section 2.1 presents the study of Network Traffic Analysis. In this section various dataset formats and the methods that have could be applied for Network Analysis are discussed. Section 2.2 presents various methods that are applied in handling the class imbalances in the dataset and shortlists a couple of potential approaches for class balancing. Section 2.3 reviews various explainable AI methods. Section 2.4 reviews four potential explainable AI methods in detail that are suitable for this use case.

Chapter 3 focuses on the Research Methodology. Section 3.2 describes the dataset, its features and data pre-processing, feature engineering of the dataset, the Models considered and the Model evaluation metrics, the explainable AI methods considered for evaluating the model. Section 3.3 describes the proposed Research Methodology in detail along with the required resources.

Chapter 4 presents the problem analysis in detail. Section 4.2 covers the description of the data along with features in the dataset. Section 4.3 describes the two datasets that are used for Malware detection and section 4.4 describes the dataset used for Network Traffic Classification. Section 4.5 presents the data preparation before model training which includes feature engineering like dropping unwanted columns, missing columns treatment, exploratory data analysis and inferences about the

distribution of the data, data scaling and standardization, and how the dataset would be split between the training and test sets.

Chapter 5 presents the results and the inferences drawn from these results. Section 5.2 looks at various sampling techniques that were considered for class balancing. Section 5.3 shows the results that have been captured from all the three models in scope and also the explanations for these models. These results give us a fair idea about the better models for classification and better explanations that explain these models.

Chapter 6 summarizes the problem, results, and recommendations from these experiments while concluding the thesis. It summarizes the contribution to the body of knowledge and also future recommendations for further improvements and study.

2. LITERATURE REVIEW

2.1. Network Traffic Analysis

The use of TLS causes a challenge in Network Threat Detection as traditional pattern matching techniques can no longer be applied here. In (Anderson et al., 2018), shows that the TLS features can be used to detect and understand malware communication. The paper extracts the TLS network flow features into a 5-feature tuple-like sequence of packet lengths, interval arrival times, byte encryption, and unencrypted TLS handshake information. SVM and L1-Logistic regression have successfully classified 18 classes of malware with 90% accuracy.

Advances in Machine learning have provided a way to leverage it for Network Traffic Analysis (Barut, n.d.). The important methods that have been used in this paper are the SVM, Random forest, and Extreme learning which is a Neural net with a single hidden layer. This paper reveals that SVM results in better accuracy with smaller datasets, while Random Forest is slow in prediction in real-time. ELMs perform better in speed and accuracy (Ahmad et al., 2018). Among the Decision Tree-based classifiers C4.5 is found to be better at classification tasks and this too fares better than the SVM on accuracy.

In general, SVM fares better with high dimensional spaces when there is clear margin separation between classes while it is not suitable for large datasets (K, n.d.).SVM along with Particle Swarm Optimization(PSO) helps in optimizing the performance of SVM to remove redundant and inconsistent features for classification (Dhaliwal et al., 2018).

Tree-based ensemble algorithms had superior success in many classification-based problems in many domains. It is worth considering these as one of the options for classification tasks. The popular algorithms in this area are AdaBoost, Random Forest, and XGBoost. Some of the key features of these algorithms are:

- No need for specific distribution of training data
- Multi-collinearity of data does not impact the performance
- Robust against overfitting by converting weak learners into strong learners by employing Boosting or Bagging.
- More insular to outliers in the data
- Computationally better than SVM and Neural networks.

Boosting and Bagging concepts are key to the implementation of Tree-based ensemble algorithms Boosting describes the combination of many weak learners into an accurate predictive algorithm. It is a sequential algorithm that works on gradient descent and makes predictions for a specified number of iterations on the entire training sample. This iteratively improves the performance of the model with information from the prior iteration (D'Souza, A Quick Guide to Boosting in ML, 2018) The goal of boosting is to improve prediction accuracy.

Bagging is non-sequential learning, also called Bootstrap aggregating refers to reduces the variance of the model by generating additional test data. Then random subsets are drawn from the training sample. These are independent draws and are from the same distribution. Data from each of the draw is used to create decision trees. The aggregate decision from these decision trees is a result of this ensemble model (Freund and Schapire, 1995) & (D'Souza, A Quick Guide to Boosting in ML, 2018) . Note that the concept of bagging is a general concept that can be applied to any ensemble model and not just specific to decision trees. The goal of Bagging is to reduce variance in the model.

Random Forest is based on the bagging approach. It achieves a reduction in overfitting by combining many weak learners that underfit because they utilize the subset of the data from the training sample. Here the trees grow parallelly (Pavlov, 2019). Random Forest along with Nested Dichotomies shows a 99% detection rate and an error rate of just 0.6% (Dhaliwal et al., 2018).

AdaBoost is an early Tree-based gradient boosting ensemble model (Freund and Schapire, 1995). It sequentially grows weak learning decision trees by giving more weight to misclassified data. The result of the AdaBoost algorithm is a weighted majority of all the weak learners. In AdaBoost, all the features are considered for each decision tree while for Random Forest only a subset of features is considered for each decision tree.

Many of the models used have their advantages and drawbacks. XGBoost boost handles many of the drawbacks of existing models. The advantages of using XGBoost for classification are, though some of these features might be also true for other algorithms.

- It is 10times faster.
- Supports parallel processing,
- Allows a wide variety of computing environments parallelization, out of core computing, distributed computing, cache optimization for efficient use of hardware.

- Make a weak learning algorithm into a strong learning one.
- Regularization to avoid overfitting.
- Built-in Cross-validation.
- Better deals with missing values in the input
- Support incremental training
- XGBoost is a flexible classifier with configuration objective functions
- Support for user-defined evaluation metrics for evaluation and rankings.
- Option to save data matrix.
- Extended Tree Pruning, results in deeper and optimized trees.
- Supports three major gradient boosting techniques i.e., standard gradient boosting, regularized boosting, and stochastic boosting.
- It is a Decision tree-based classification model using gradient boosted trees.
- Has an in-built function called F1-Score that lists the features' importance.

All the above properties make XGBoost one of the preferred models for classifications (Dhaliwal et al., 2018). The key parameters of the XGBoost are learning_rate, max_depth, n_estimators, random_state, n_splits, and other regularization parameters like gamma, alpha, and lambda which are used to reduce the overfitting of the model.

Unsupervised learning uses a huge volume of data and has lesser accuracy. To overcome this, a semi-supervised model to handle unlabeled in a co-training (supervised +unsupervised) to enhance the performance of the algorithms. This approach is most useful when new models must be built dynamically on the fly owing to the concept drift or when there is a lot of unlabeled data in the training dataset.

Of late, Deep learning is being applied in multiple areas and its application to Network Traffic Analysis is no exception. Deep learning for Analysis of Network Traffic Measurements (Miguel Marín Freire and Sprechmann Google DeepMind Pere Barlet-Ros UPC Barcelona Tech Director Académico, 2019), showcases exhaustive research that has been carried out in this area. This paper's results show that Deep learning achieves great accuracy in Malware detection and Malware classification. This also focuses exhaustively on using two variant datasets, one based on the Raw packets, where each instance

represents one packet, and using Raw flows where each flow is made up of multiple raw packets assembled as a Tensor. This approach to classification uses Neural Networks for classification.

2.2. Handling Class Imbalance in Dataset

In most of the classification problems, the general issue is that the minority classes are much smaller in number and are subsumed by the majority classes. It is important to get the prediction of these minority classes right, as in most cases the use cases it is important to get the predictions for minority class right. Otherwise, it degrades the performance of inductive learning algorithms like SVM, Neural networks and decision trees, etc., (Moreo et al., 2016).

SMOTE (Synthetic minority Over-Sampling Technique) along with majority class under-sampling has given better results. In this approach over sampling synthetic samples are generated by taking the difference between the feature vector under consideration and its nearest neighbor and multiplying this difference between 0 and 1 and adding it feature vector under consideration. SMOTE forces focused learning and it introduces a bias towards the minority class (Kovács et al., 2020).

ADASYN (Adaptive Synthetic) oversampling approach uses weighted distribution for different minority sampling class examples according to their level of difficulty in training. Here density distribution is used to automatically decide the number of synthetic samples that need to be generated for each minority data example, while in SMOTE it uses generated an equal number of synthetic sample for minority data example (He, H., Bai, Y., Garcia, E., & Li, 2008).

Random Oversampling randomly duplicates the observations from minority classes and adds them to the training dataset. Since this makes exact copies of the observations there is a high likelihood of overfitting and accuracy degrades with real data (Fernández, 2018). Similarly, Uniform Oversampling ensures the oversampling of minority class is done uniformly across the features space of the minority class.

Distributional Random Oversampling generates a new random minority-class synthetic dataset by leveraging the distributional properties of the document. This is primarily used in the context of text classification (Moreo et al., 2016) and may not be relevant for this use case.

Random Under Sampling approach deletes samples from the over-represented class and random sampling might sample from the entire region of feature space if the imbalance is too large (Zehori,

2020). This can make it difficult to learn the boundaries between the minority and majority classes to learn.

The other more advanced deep learning-based generative techniques like Generative Adversarial Networks (GAN) and Variational Auto-encoders (VAE). These generate realistic samples of complex content like images, texts, and sounds. These are not applicable for the use cases under consideration and are not considered further.

2.3. Explainable AI

2.3.1. Models Categorization

From the point of interpretability machine learning models can be a white box or black box.

Whitebox models: Whitebox models are transparent and logic on how they work is easily understood from their linearity, monotonicity where the relation between the input and output features can be easily understood by humans. Linear regression, Logistic regression, and Decision trees are some examples where the model predictions can be understood by humans easily.

*Blackbox Models***:** In the case of Black box models the internal working of the model is hard to understand for the humans and the functional relationships between the input features and the model's output. Deep Neural networks, Gradient boosting, Random forest, Support Vector machines are examples of Black-box models.

In general, the black-box models help generate more accurate predictions as they use more complex logic to arrive at the results and this significantly reduces their interpretability. If it is not for the explainable AI methods a balance must be struck between the model interpretability and the model accuracy by trading accuracy for interpretability.

2.3.2. Local vs Global explanations

Explanation of AI can either be Local or Global. For Local explanations, the individual relevance of each of the features in effecting the output (Melis et al., 2018). To explain the model globally a better way is to take a sample set and then average the relevance of the features (Melis et al., 2018). While this approach works for differential models, for non-differential models like a random forest we take the help of surrogate models, where predictions from a non-differential algorithm are used to train

non-linear models like SVM (Melis et al., 2018). The Global explanation of the model needs to pick instances/observations such that components/features are not redundant (Ribeiro et al., 2016).

2.3.3. Model agnostic methods for Explainable AI / Attribution and Perturbation methods

Perturbation and Attribution explainable AI methods are slightly related in a way. Perturbation methods work by changing the value of inputs in a controlled way to see how it is impacting the outputs. Attribution methods work by masking the inputs and see if that has any impact on the outputs.

With the success of black-box models like Deep Neural Networks (DNN), Conventional Neural Networks (CNN), Gradient Boosting, Random Forest, Support Vector Machines (SVM) there is a need for explainable AI methods for black-box models. Various explainable AI methods to explain the black-box models are Partial Dependent Plot(PDP), Feature importance, Surrogate Model, LIME, SHAP, Anchor (Harry, 2020).

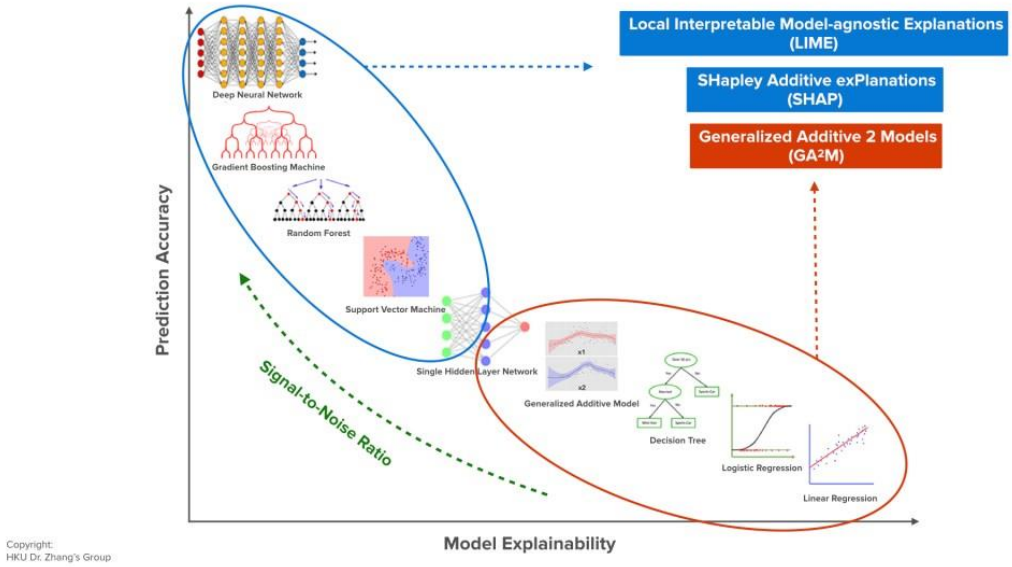


Figure 1: ML Models vs Explainable Models (Shum, 2020)

- *Partial Dependent Plots (PDP):* PDP plots average model output changes when the feature or a set of features are marginalized over other features. This means that a few select feature values are changed with different values while keeping the other features unchanged. Since these plots average one response over the other to give a global view impact of features on

the model response a variation of this method called Automated Dependent Plots (ADP) (Inouye et al., 2019).

- *Individual Conditional Expectation(ICE):* This shows one line per instance when a feature changes. A PDP is an average ICE plot. ICE curves display one curve meaningfully and so feature correlation is not captured (Molnar, 2021).
- *Accumulated Local Effects:* These are unbiased and faster compared to PDP and can work well even when the features are correlated. This can provide better visualization plots (Apley and Zhu, 2020).
- *Permutation Feature Importance:* It permutates feature values and a change in the prediction error is measured. This captures the relationship between input features and the model's targets (Molnar, 2021).
- *Global Surrogate:* For non-differential models like Random Forest we take the help of surrogate models where predictions from non-differential models are used to train non-linear models like SVM (Melis et al., 2018)
- *Local Interpretable Model-agnostic Explanations (LIME):* This method explains predictions of any classifiers by learning the model locally around the individual model's prediction. This can be applied to all types of models including the classification of images. This can be done by randomly picking a few instances to explain the model outputs (Ribeiro et al., 2016).
- *Anchors:* This method uses a perturbation approach using if-then-else rules when inputs are complex within the neighborhood of an instance.
- *Shapley Values:* Tells how to fairly distribute prediction amount the features. This is derived from game theory. This is computationally expensive (Molnar, 2021) .
- *SHapley Additive exPlanations (SHAP):* This is derived from about 6 other existing explainable methods such as LIME, LOCO, Shapley values (Molnar, 2021) . It addresses important features of explainability like Local accuracy, Missingness, and Consistency across different observations(Lundberg and Lee, 2017).
- *Leave one Covariate out (LOCO):* One variable is removed from inputs and the variable that has the largest impact on the model output is the most important one. This deteriorates inaccuracy in the case of complex non-linear models (Lei et al., 2018).

2.3.4. Model-specific Explainable AI methods - for Deep Neural Networks

Most of the model-specific explainable methods are to explain the Deep convolutional networks and these can be categorized as below.

- *Backpropagation-Based Methods:*
  - *Gradient method:* $\frac{\partial S}{\partial x}$ ie., quantifies for each unit change in input dimension, what would the change in prediction $S(x)$ (Sundararajan and Taly, 2018)**.**
  - *Integrated gradients:* (x-$\bar{x}$) $*$ $\int_0^1 \frac{\partial S}{\partial x}(\bar{x} + \mu(x - \overline{\boldsymbol{x}})/\partial$x , where $\overline{\boldsymbol{x}}$ represents baseline input that represents the absence of a feature in original input x (Sundararajan and Taly, 2018).
  - *Layer-wise Relevance Propagation (LRP):* The algorithm starts at the output layer proceeds layer by layer backward and redistributes the relevance of the output across all layers of the Neural network till the input layer (Ancona et al., 2017).
  - *DeepLIFT:* Each feature is assigned an attribution corresponding to the relative effect of the feature on the network compared to activation at some reference input. The baseline can be chosen as zero (Ancona et al., 2017).
  - *Localizations / Saliency Maps:* Constructs attributions by taking the absolute value of the partial derivative of the target output concerning the input features (Ancona et al., 2017)**.**
  - *Deep Taylor Decomposition:* Each neuron is calculated as a function that is decomposed on its input variables. The decompositions of multiple neurons are then aggregated or propagated backward (Montavon et al., 2017). This method produces sparser explanations, assumes no negative evidence in the input, and produces only positive attribution maps (Ancona et al., 2017).
  - The other methods like *Deconvolutional Network, Guided back-propagation, Guided GradCAM, SmoothGrad, VarGrad* are primarily used for explaining models that classify the images. Since in our case we are do not dealing with the input images, these methods are not considered for further study.

- *Forward propagation methods*:
  - *Activation Maps*: Also called channels are widely used with Convolutional Neural networks. Here the features are activated in the forward pass of the network (Vuppala et al., 2020).
  - All model agnostic Perturbation / Attribution methods work here as well. In contrast to DL backpropagation methods, forward propagation transfers the changes in input to the scores that are computed at the very last layers (Salehi, 2019).

2.3.5. Model-specific Explainable AI methods - for RandomForest

Explainable AI methods that specifically address the random forest visualizations exploit the observations from the validation set to identify the important features and similarity between the outputs can be measured (Robnik-Šikonja and Bohanec, 2018). Recently an explainable AI method CRIPS has been proposed that extracts the decision path in each of the trees that contributes to majority classification and then uses frequent pattern mining to identify the most commonly occurring split conditions (Hatwell et al., 2020).

2.4. Detailed review of select Explainable AI Methods

This section primarily focuses on the following Explainable AI methods:

- *Model Agnostic Methods:* LIME and SHAP
- *Deep Leaning specific methods:* Integrated gradients, DeepLIFT.

2.4.1. LIME

LIME is a widely used local interpretable explainable AI method for providing quick explanations for model predictions for a given observation. For each given observation is generates a model that very specifically explains the given observation and is valid only for the data in the neighborhood of the given observation and this explainable model is very close to the actual model predictions (Ribeiro et al., 2016; Visani et al., 2020).

LIME approximates the black-box model f with a simple function g to explain a specific point of interest.

$f: R^P \longrightarrow R$, is the black-box model, for the input set P.

$g: R^p \longrightarrow R$, explainable model, that specifically explains subset points $p \in P$.

Since many explainable AI methods are possible for a given subset of points, the explainable AI is chosen in such a way that $arg\ min_{\ g\ \epsilon\ G}\ L(f, g, \pi_x) + \ \Omega_{(g)}$, where $\Omega_{(g)}$ is the complexity of g, L is the loss function and π_x the weight assigned according to proximity to x.

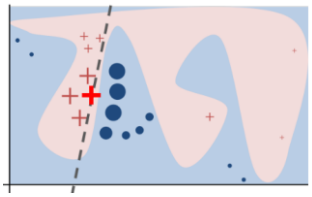


Figure 2: LIME explainable AI model for a given point (Ribeiro et al., 2016)

LIME generates new data points from a multivariate and normal distribution of features in a dataset with an assumption that features are independent of each other and uses these data points to generate an explainable AI method. The locality of a specific data point is identified by assigning weights π_x , using Gaussian Kernel. The new data points are fed into the black-box model and the output is generated from the model. The input data points are rescaled as needed. This new and complete data set along with generated inputs and the outputs from the model are used to generate a simple explainable AI model also called the surrogate model. The explainable AI method is only applicable to explain the specific datapoint and other data points in the same locality and a different explainable AI method is generated. For a compact human-readable explainable AI method, only a few important features are selected using the Lasso technique. The resulting model is a linear explainable AI method with fewer and important features. Then on these standardized features linear model is built. Using ridge regression avoids overfitting. (Ribeiro et al., 2016; Visani et al., 2020) .

The dataset dimension has a significant impact on LIME owing to weights that are applied to the dataset, leading to not being able to distinguish between the near and farther data points in the dataset. This weakness can lead to an inefficient LIME data model when applied to a large dimensionality dataset which is achieved by dimension reduction before applying LIME for black-box model explanation (Visani et al., 2020).

The consistency of LIME explanations AI can be verified using a stability index from the distribution of coefficients that are generated for different model explanations (Visani et al., 2020).

2.4.2. SHAP

SHAP is another local explainable AI method. SHAP presents a unified approached to interpreting the models. For this, it considers a class of additive feature attribution methods that draws parallels between six explainable AI methods. Additive feature attribution methods are explanation model that is a linear function of binary variables that can be expressed as:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z_i$$

To find ϕ , LIME minimizes the objective function $arg\ min_{\ g \in G} L(f, g, \pi_x) + \Omega_{(g)}$. This equation can be solved using penalized linear regression.

DeepLIFT uses $\sum_{i=1}^{n} C_{\Delta xi\Delta t} = \Delta t$, where the model output is seen in terms of difference from the reference input.

Layer Relevance Propagation (LRP) is like the DeepLIFT explanation where the reference activations of all neurons are fixed to zero. This uses a relatively simple function $x = h_x(x')$, where 1 maps to the original input feature, if zero the feature has no relevance in output.

Shapley regression values are of feature importance for linear models in the presence of multicollinearity. This method requires retraining the model on all feature subsets $S \subseteq F$, where F is a set of all features. To compute this effect a model $f_{SU\{i\}}$ is trained with the presence of a feature and another model without the existence of that feature and the prediction from these two models are compared with the current input $f_{SU\{i\}}(x_{SU\{i\}} - f_S(x_S)$, where x_S represents the values of input features in set S and is done for all possible subsets. The Shapley values are weighted values of all possible differences and are represented as:

$$\phi_i = \sum_{S \subseteq F\ \{i\}} \frac{|S|!\ (|F| - |S| - 1)}{|F|!} \left[f_{SU\{i\}}(x_{SU\{i\}} - f_S(x_S)\right]$$

For *Shapley regression values* h_x maps to 1 or 0 to the original input space, where 1 indicates the input is included in the model and 0 indicates exclusion from the model. This an additive attribution method.

Shapley sampling values explain any model by applying sampling approximations to the equation and approximating the effect of removing a variable by integrating over samples from the training dataset. This is also a similar additive attribution method as Shapley regression values.

Shapley quantitative input influence is another additive attribution method like Shapley sampling values the has a broader framework that independently proposes approximation Shapley values.

The common feature of the additive attribution method is the presence of a single unique solution in this class with the following desirable properties:

- *Local accuracy:* When approximating the original model f for a specific input x, the explanation model $g(x')$ should match $f(x)$ for the simplified input x'

  $f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x_i$
- *Missingness:* Requires that the features missing in the input have no attributed impact on the output.
- *Consistency:* If a model changes resulting from some simplified input's contribution increases or stays the same regardless of other inputs, that input's attribution should not decrease. According to this theorem under this property, for a given simplified input h_x there is only one possible additive attribution feature method, which means that the methods not based on Shapley values violate the local accuracy/consistency rule.

The SHAP proposes a unified measure of feature importance. These Shapley values are conditional expectation functions of the original model. While conditioning on a feature, it attributes the change in model prediction to each of the input features. If the features are not independent of each other the order in which the features are added to the explanation model matters and SHAP values for each of ϕ_i are arrived at by averaging all the possible orderings where:

$f_x(z') = f(h_x(z')) = E[f(z)|\, z_S]$

$z' \subseteq x'$, z' vectors are non-zero entries in x'.

S is a set of non-zero indexes in z' and z_S has missing values for features not in set S.

$f_x(z') = f(h_x(z')$ is the SHAP explanation model for simplified input mapping.

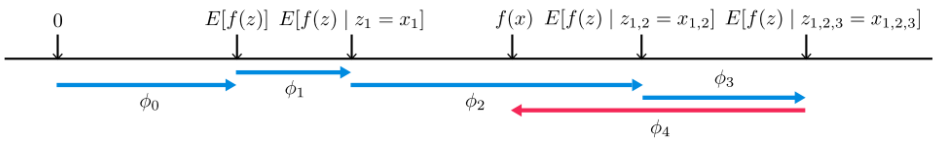


Figure 3: SHAP Explanation (Lundberg and Lee, 2017)

SHAP values are estimated using the Shapley sampling values method. Separate sampling estimates are performed for each feature attribution and it is reasonable with a smaller number of inputs. Kernel SHAP (Linear LIME + Shapley) performs only a fewer evaluations of the original model to obtain similar accuracy. Since LIME is an additive feature estimation method Shapley values are the only possible solution to the equation that satisfies the Local accuracy, missingness, and consistency.

While Kernel SHAP improves the efficiency of model agnostic explanations, SHAP also improves the efficiency of explainable models, by providing model-specific explanation models like:

- *Linear SHAP*: For linear models, we assume input feature independence can be approximated directly from the model's weight coefficients.
- *Low-Order SHAP*: This model is considered to address the drawback of higher complexity when working with linear models.
- *Max SHAP:* Using permutations formulation of Shapley values, the probability that each input increases maximum value over other input can be calculated.
- *Deep SHAP (DeepLIFT + Shapley values)*: Since DeepLIFT is additive feature attribution like other linear models Shapley values are the only ones that satisfy the local accuracy, missingness, and consistency. DeepSHAP combines the smaller SHAP values calculated for the component of the network into SHAP values for the whole network by recursively passing DeepLIFT multipliers defined in terms of SHAP values.

Kernel SHAP and Deep SHAP have been compared against the LIME and the DeepLIFT methods. Kernel SHAP has better efficiency, local accuracy, and consistency when compared to LIME though values between them differ significantly. SHAP explanations are more consistent with Human intuitions when compared to LIME and DeepLIFT (Lundberg and Lee, 2017).

2.4.3. Integrated Gradients

The Explainable AI methods that attribute the output values to the input parameters should minimally satisfy two fundamental Axioms Sensitivity and Implementation Invariance (Sundararajan et al., 2017).

The sensitivity of explainable AI models is the strength of the model to capture small changes in the input to generate a corresponding output. In general, attribution-based explainable AI methods are

difficult to evaluate factually what matters is if that is making sense from the problem perspective. Linear models or other non-linear models that leverage the surrogate linear models to explain the attributions of various input variables are evaluated as a product of model coefficients and their corresponding input variables. For Deep neural networks Gradient method is a simple and similar attribution explainable AI method. The attributions are calculated as the product of gradients and values of input features. Gradients do not adhere to sensitivity as activation functions are used in neural networks that do not proportionately translate inputs to outputs like linear functions. E.g., $f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$. This kind of function cannot be leveraged to calculate the gradients (Sundararajan et al., 2017).

Implementation invariance refers to the ability of the explainable AI to explain just based on model inputs and outputs without any reference to how the model has been implemented. Gradients method adheres to principle as the gradients are simply calculated on the rate of output change concerning the inputs change $\frac{\delta f}{\delta g} = \frac{\partial f}{\partial h} . \frac{\partial h}{\partial x}$. Here the implementation details of the model do not effectively matter on Gradient method explanations (Sundararajan et al., 2017).

Integrated Gradients combines the Sensitivity of LRP or DeepLIFT and the sensitivity of implementation invariance of Gradients. The integrated gradients along the i^{th} dimension for an input x and baseline x' the integrated gradients is calculated as $(x - x') * \int_{\alpha=0}^{1} \frac{\partial F\left(x' + \alpha(x - x')\right)}{\partial x_i}$.

Integrated gradients satisfy the completeness axiom where attributions from each dimension add up the difference of output of model at input *x* and baseline input x_i. For images, the baseline input is a black image while for the text model it could be a zero vector. Similarly, Integrated Gradients also satisfies Linearity axioms where two neural networks are linearly composed and Symmetry-Preserving axiom where two input variables when exchanged results in the same function value i.e., $f(x, y) = f(y, x)$ for all values of x and y (Sundararajan et al., 2017).

One of the drawbacks of this approach is that it is computationally expensive. Additionally, for certain functions importance is based on the combination of features Integrated Gradients method fails as it assigns importance to either one of the features, but not to both of them (Shrikumar et al., 2017).

2.4.4. DeepLIFT

DeepLIFT addresses the Integrated Gradients limitations like the performance and the being able to attribute the importance to a combination of input features. The DeepLIFT replaces gradients with discrete gradients and still uses a modified form of backpropagation. But the idea is that a large, discrete step avoids flat regions, avoiding breakage of sensitivity. Unfortunately, these methods violate a different requirement on attribution methods. (Shrikumar et al., 2017).

The perturbation-based explainable AI methods make perturbations to various inputs and observe the impact on neurons in hidden/output neurons in the network. These methods are computationally inefficient as each perturbation needs a separate feedforward in the neural network. There are many backpropagation methods, but they are fundamentally related and based on gradients. They also underestimate the importance of features that have saturated their contribution to the output (Shrikumar et al., 2017).

DeepLIFT is one of many back-propagation methods that propagate the importance of neurons across all hidden layers back into the input features based on the function connecting those two layers. In one backward pass, we get important scores for all positions in inputs. DeepLIFT relies on the concept of difference from reference activation i.e., the activation on the neutron when presented with neutral input. This way even if the local gradient is zero, DeepLIFT propagates the importance of neurons back into the input features. DeepLIFT generalized to all activations. E.g. for Sigmoid is used as activation functions in neural networks are gating units in Recurrent neural networks the local gradients flattens towards the outer end of the input range. DeepLIFT effectively captures the importance of the input features well. It explains the difference of output from the reference value in terms of the difference of input from the reference value. In DeepLIFT the sum of the difference of output from the reference value at a given neuron also called contribution can be calculated as $\sum_{i=1}^{n} C_{\Delta x i \Delta t} = \Delta t$, i.e., the sum of all the contributions from neurons from the preceding layer. The right value of the reference is based on the use cases/domain knowledge gained in that area. The multiplier definition is defined as $m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x}$ the number of times the contribution of the neuron changed from the reference input. This is like $\frac{\partial x}{\partial t}$. DeepLIFT uses the chain rule to calculate the contribution of inputs while backpropagating crosses the layers. The positive and negative contributions are treated separately as $C_{\Delta y \Delta t} = C_{\Delta y + \Delta t} + C_{\Delta y - \nabla t}$. All these transformations can be applied to linear as well or non-linear functions like ReLU, tanh, or sigmoid operations. Based on the

use case the weightage given to the positive and negative contributions can be the same or different. Shapely values measure the average marginal effect of including input over all possible orderings in which inputs can be included. DeepLIFT can be thought of as a fast approximation of the Shapely values (Shrikumar et al., 2017).

2.5. Summary

Network Traffic Analysis is a difficult problem to address as the margin of error is less. We need an effective and reliable model for identifying malicious traffic and classification of Network Traffic itself so that effective Network applications can be built. After reviewing various papers XGBoost, Random Forest, and Neural Networks have been shortlisted for building the models for Malware detection and Network Traffic Classification. It is also important that we find the best explainable AI method. We explored the model agnostic attribution and perturbation methods like LIME and SHAP and for Deep learning models additionally, model-specific explainable AI methods like Integrated Gradients and DeepLIFT to be explored as needed.

3. RESEARCH METHODOLOGY

3.1. Introduction

The dataset made available for the challenge is a novel NetML dataset and curated datasets from CICIDS2017 and non-vpn2016 datasets. The scope of this study is to do high-level malware detection using the first two datasets. High-level Network traffic classification is on the non-vpn2016 dataset. The focus is achieving high test accuracy in Malware detection and Network Traffic classification using RandomForest, XGBoost, and Deep Learning models. Explainable AI would be used to explain the results from each of the models. This provides more insights into how a particular observation is classified in the Network Traffic (Melis et al., 2018). Explainable AI can be leveraged to iteratively improve the model, just not confine to given explanations for the predictions after the model has been built.

3.2. Research Methodology

3.2.1. Dataset Description

As mentioned earlier, all three different datasets are in JSON format extracted using an accelerated feature extraction library. "It computes the features of a flow up to 200 packets in both directions. Given a raw traffic capture (pcap) file as input to the feature extraction tool, flow features are extracted in JSON format, and each flow sample extract is listed line by line in the output file. Given a raw traffic capture file as input to the feature extraction tool, flow features are extracted in JSON format, and each flow sample is listed line by line in the output file. Metadata features are extracted for each flow sample and TLS, DNS, and HTTP features are extracted only if the flow sample contains packets for the given protocols." (Barut, n.d.). Note here in these datasets the source and destination IPs are masked and the start and end time of packet capture in pcap is replaced with duration. All the datasets are split into 3 sets. 80% training set, 10% test set, and another 10% as challenge/validation set. Since we do not know the results from the test and validation set we consider the training set as a whole set and then split this data into train and test data for this paper.

These datasets are explained in brief below.

NetML dataset (For malware detection and classification) (Barut, n.d.):

Table 1: Data Set Fields Description

Metadata Features
sa: source address
da: destination address
pr: protocol (6 or 17)
src_port: source port
dst_port: destination port
bytes_out: total bytes out
num_pkts_out: total packets out
bytes_in: total bytes in
num_pkts_in: total packets in
time_start: time stamp of first packet
time_end: time stamp of last packet
intervals_ccnt[]: compact histogram of pkt arriving intervals
ack_psh_rst_syn_fin_cnt[]: histogram of tcp flag counting
hdr_distinct: number of distinct values of header lengths
hdr_ccnt[]: compact histogram of header lengths
pld_distinct: number of distinct values of payload length
pld_ccnt[]: compact histogram of payload lengths
hdr_mean: mean value of header lengths
hdr_bin_40: # of pkts with header lengths between 28 and 40
pld_bin_128: # of pkts whose payload lengths are below 128
pld_bin_inf: # of pkts whose payload lengths are above 1024
pld_max: max value of payload length
pld_mean: mean value of payload length
pld_medium: medium value of payload length
pld_var: variance value of payload length
rev_...: flow features of the reverse flow

TLS Features
tls_cnt: number of tls packets
tls_len[]: array of tls payload length
tls_cs_cnt: number of ciphersuits
tls_cs[]: array of ciphersuits value
tls_ext_cnt: number of tls extensions
tls_ext_types[]: array of tls extensions
tls_key_exchange_len: length of tls key exchange
tls_svr_...: tls features advertised by the server
DNS Features
dns_query_cnt: number of dns query records
dns_query_name_len[]: array of dns query name lengths
dns_query_name[]: array of dns query names
dns_query_type[]: array of dns query types
dns_query_class[]: array of dns query classes
dns_answer_cnt: number of dns answer records
dns_answer_ttl[]: array of dns answer âĂIJtime to liveâĂİ values
dns_answer_ip[]: array of dns answer ips
HTTP Features
http_method: http request method
http_uri: http request uri
http_host: http request hostname
http_code: http code
http_content_type: http content_type
http_content_len: http content length

For all the datasets the above features are extracted. For NetML and CICIDs2017 datasets, for each observation, the classification is at the top level which is either benign or malware. For non-VPN datasets, the classifications at the top level are P2P, Audio, Chat, Email, File Transfer, Tor, or Video.

3.2.2. Feature Engineering and Feature selection

Given that we have curated a dataset where all the features needed for model building are already extracted from raw packets no additional engineering on these features is needed. To start with no specific feature selection is employed. We delay the feature engineering to the later parts of the model building, where we decide if any unwanted features are being considered by the model that ought to be removed based on the feature importance from the trained model itself and by leveraging the explainable AI methods to further fine-tune the model by removing unwanted or unreasonable features

from the model. For all the three models that are considered for this paper is it important that we analyze the feature importance from the model also using explainable AI methods, along with the domain knowledge during the feature selection.

3.2.3. Data pre-processing

Normally in data-preprocessing techniques like imputation, outliner treatment, binning, scaling, one-hot encoding, date handling are performed so that they can be fed into the model for better accuracy and robustness.

The selected dataset in JSON format and major preprocessing that is done on the dataset is to convert it into tabular / vector format so that is in a form that can be fed into the models for training. Further on, for this study, no specific additional data preprocessing is needed/ or identified so far the dataset is already curated. Encoding the categorical variables and applying standard scaling on the input variables is considered as needed as part of the implementation.

3.2.4. Models

Below are three models that are used. To start with all the features are used while building the models.

- Balance the dataset using the ADASYN algorithm as needed.
- *ML Models*:
    - Random Forest and XGBoost: Find optimal parameters that can give high accuracy.
    - Deep Learning: Use Neural networks with at least 4-8 hidden dense layers.
- *Explanation Methods*:
    - LIME: Model agnostic explainable AI methods for local interpretable models.
    - SHAP: Good at local explanations, more suitable for Tree base algorithms, and can be applied to all models equally well.
    - DeepLift: Decomposes the prediction of a neural network by backpropagating the contributions of all neurons in different layers in the network to every feature of the input.
    - Integrated gradients: This computes the partial derivative of the model output against the corresponding input feature. This satisfies two fundamental Axioms of Sensitivity and Implementation Invariance.

- Note: The last two model-dependent explainable AI methods are to be used on the Neural Networks model only if it performs better than the other models.

3.2.5. Model Evaluation

Model evaluation is based on multiple metrics like accuracy, confusion matrix, precision, recall, Area under the curve (AUC), and Receiving Operating Curve(ROC). All these metrics are leveraged in evaluating various aspects of classification accuracy.

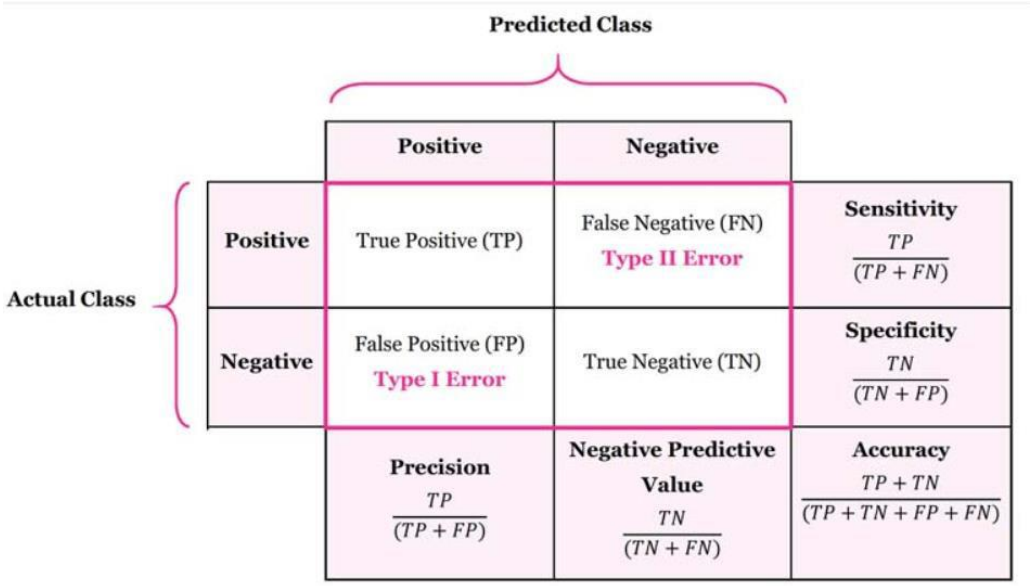


Figure 4: Confusion Matrix (Science, 2019)

- Accuracy: Ratio between the correct predictions and the total observations. This is used for both malware detections/ classifications and network traffic classification,
- Confusion matrix: This a matrix representation of actual vs predicted in the case of binary classification. This is applied to the high-level classification of malware detection based on the TCP flow.
- ROC (Receiving Operating Curve) shows the trade-off between True positive rate and False positive rate. AUC (Area under the ROC Curve) gives the overall performance of a binary classifier.
- Precision & Recall and Sensitivity & Specificity are two widely used binary classification metrics, based on the use case.

- Note that different libraries like Sklearn etc., show these four metrics TP, TN, FP, N in different quadrants.
- F1-Score is a harmonic mean that combines the recall and the precision into one single metric. This score has high importance when high precision and high recall values are needed at the same time. The formula for the F1 Score is $\frac{2*(Precision*recall)}{(Precision+recall)}$.

- Note that the confusion matrix from the Sklearn library is in the below format:
  [TN FP
  FN TP].

3.2.6. Explaining AI

Model agnostic explainable AI methods :

Below explainable AI methods are considered for the scope of this work, based on the wide use and their applicability to Network Traffic Analysis use cases.

- *Local Interpretable Model-agnostic Explanations (LIME):* This method explains predictions of any classifiers by learning the model locally around individual mode prediction. This can be applied to all types of models including the classification of images. This can be done by randomly picking a few instances to explain the model outputs (Ribeiro et al., 2016). LIME is a perturbation method that checks the significance of a feature in the outcome that has been predicted. The explanations change based on the inputs and cannot be considered for a general explanation of the model.
- *SHapley Additive exPlanations (SHAP):* This is derived from about six other existing explainable methods such as LIME, LOCO, Shapley values. It addresses important features of explainability like Local accuracy, Missingness, and Consistency across different observations(Lundberg and Lee, 2017).

Explainable AI methods for deep learning models:

Below Attribution methods for Neural networks are considered as we have structured data and we do not deal with the images in our case study. These are used for study only if the model based on Neural Networks outperforms the other models.

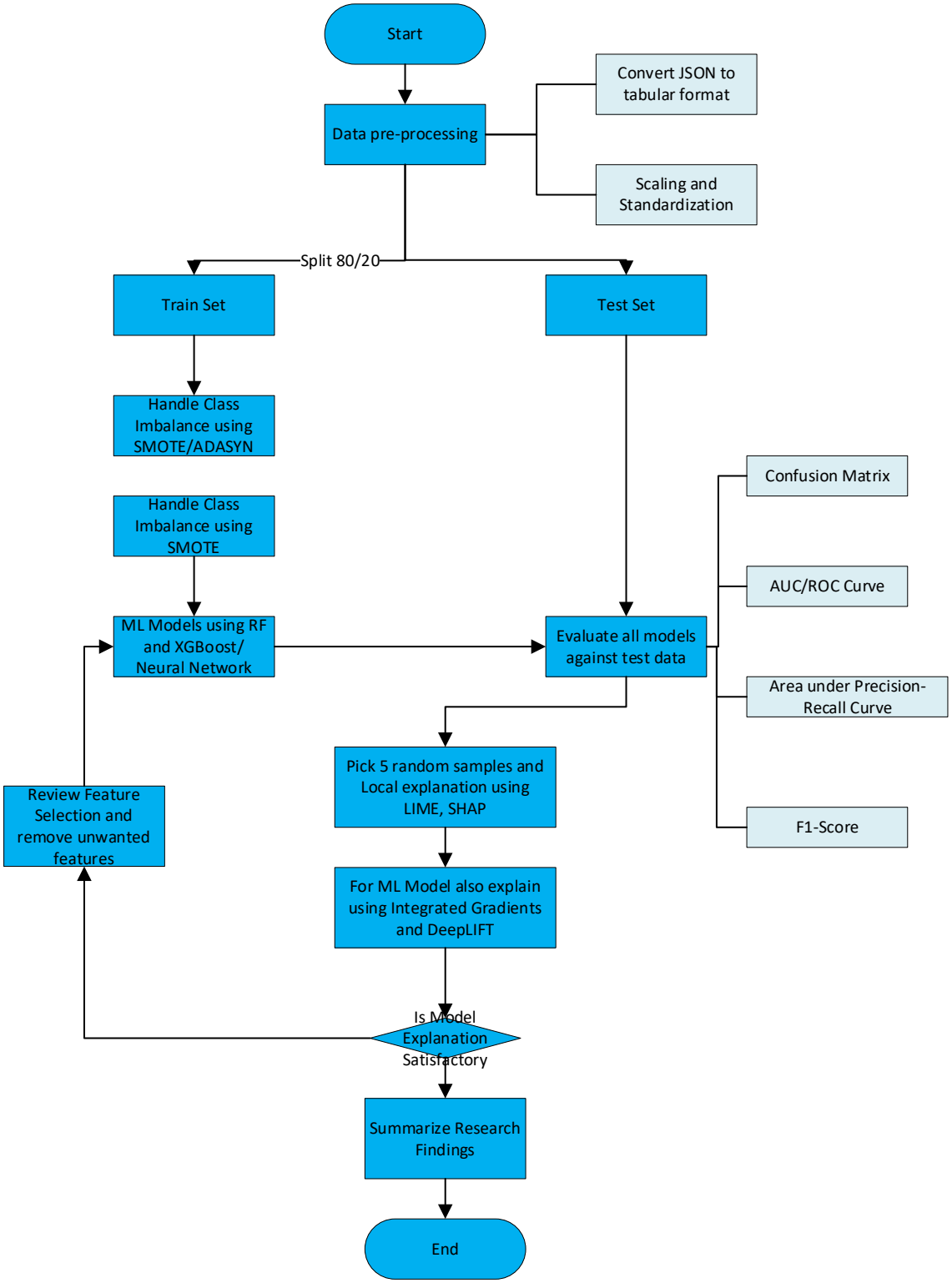
- *Integrated Gradients:* This is recommended for differential models and neural networks where there is a large number of input features, as it can perform faster. This computes the partial derivative of the model output against the corresponding input feature. Integrated Gradients calculates the mean gradient by varying the input linearly from $\bar{x}$ to x. The baseline can be chosen as zero. The attributions add up to the model output minus the model output evaluated at chosen baseline (Ancona et al., 2017).
- *DeepLIFT:* Each feature is assigned an attribution corresponding to the relative effect of the feature on the network compared to activation at some reference input. The baseline can be chosen as zero (Ancona et al., 2017).

Comparative study:

A Comparative study is performed on the model metrics for each of the models and the Explainable AI methods are applied to verify the methods that explain the results well. There are no definite metrics for explainability as evaluation can be at application, human and functional levels (Doshi-Velez and Kim, 2017).

3.3. Proposed Method

Below describes the summary of the Research methodology flow chart.



NOTE: This is the approach applicable to all 3 datasets under the scope of this study.

Generally, no standardization is required when working with models like Decision Tree, Random Forest, Gradient boosting. Though a Neural network does not need standardization there is a good chance of improving the accuracy of the model with standardization (Jaadi, 2019). To ensure the same pre-processing is applied for uniformity sake the data is standardized as needed for variables if the variance in data is noted (Stöttner, 2019).

To start with no feature engineering/feature selection is performed on the data without consideration for the domain knowledge involved in the classification of network traffic.

After the data is split on an 80:20 basis, ADASYN oversampling technique is applied for handling the class imbalance in the training dataset as ADASYN is slightly advanced than SMOTE as the samples that are generated have more randomness to it.

The next step uses the whole generated dataset to build three different models which are based on:

- Random Forest using the Bagging approach.
- XGBoost is an extreme gradient boosting method.
- Neural network with a minimum of 4 hidden dense layers and with dropouts randomly to avoid overfitting.

During the construction of the models, all the possible fine-tuning are considered so that the model does not overfit and is generic enough to handle the noise in the data. In the case of Random Forest the right depth of the tree, the number of estimators, etc. are identified iteratively to find the optimal parameters. For XGBoost similarly, the regularization is achieved using various parameters like gamma, alpha, and lambda which control the split based on expected loss reduction after the split, and L1, L2 regularizations which are based on leaf weights. For Neural networks appropriate regularization methods L1 and L2 is considered. L1 regularization ensures the weights of insignificant variables are set to zero while keeping the loss at an optimally low value and helps in feature selection inherently build into the modeling itself. For neural networks, dropouts are also be used to make sure the model is not overfitted.

A robust model with maximum accuracy is determined using the test data while keeping the false positives and false negatives as low as possible as it an important requirement for malware detection and network classification. ROC/AOC and/or F1-score is used as key metrics to determine the accuracy of the model.

Random samples are picked from the test data and for each of the models, LIME and SHAP which are local and model agnostic methods are used to draw an explanation for the models. Based on the model explanations some features from the dataset might be removed if any of the unwarranted features are being given due importance by the models. The process is repeated till a satisfactory model with high accuracy is achieved along with a reasonable explanation of the model.

Additionally, for a model based on Neural Networks, if it performs better than other models explanation using Integrated Gradients and DeepLIFT is considered.

3.3.1. Required Resources

Below are software and hardware requirements.

Software Requirements:

- Mendeley – For literature review and citations.
- Anaconda Navigator
- Jupyter notebook
- Python
- Python Libraries for modeling:
  - Pandas
  - Numpy
  - Matplotlib and Seaborn
  - Scikit-learn
  - Keras
  - Tensorflow
- Python libraries for interpretability methods:
  - LIME
  - SHAP
  - IntegratedGradients
  - DeepLIFT

Hardware Requirements:

- Windows 10, 64 bit
- 16 GB RAM

- Intel CORE i5, 8th Gen
- GPUs.

3.4. Summary

The methodology is closely aligned with the CRISP-DM methodology with small changes. For each of the three given datasets, each dataset is converted into a tabular format, and standardization of data is applied as needed. Then multiple models are built based on Random Forest, XGBoost, and Neural networks optimizing each model to achieve the best accuracy in classification. For each of these models explainable AI, methods are applied by selecting random samples from the test dataset. For all three models, LIME and SHAP explainable AI methods are applied to understand the differences in how different models arrive at a prediction. Given that Neural Networks is a black-box model, model-specific explainable AI methods are to be applied as needed, if this model delivers better results and if it is felt that additional insights are needed to better understand the model.

4. ANALYSIS

4.1. Introduction

After the overall research methodology has been identified from the literature review, in this section further detailed analysis on the datasets is done and the details of applying the above research methodology are captured. Any minor additions or changes that are discovered while implementing the changes are addressed in this section.

4.2. Dataset Description

For the scope of this thesis, we are dealing with two datasets NetML, CICIDS for implementation models for malware/benign network traffic classification, and a non-VPN dataset for network traffic classification.

4.3. Datasets for malware detection

NetML is one of two datasets that has been used for detecting malware in network traffic flows. The JSON NetML dataset is converted to a tabular format for doing further analysis and working with standard tabular dataset formats. Each of the observations in the dataset is aggregated from the first 200 packets network traffic flow in both directions. The dataset has 121 columns/features captured from the Network Traffic flows. Since this is a curated dataset there are no null columns and no further Null treatment is needed. Here 80% of the observations are malware while the rest are benign traffic.

CICIDS is the second dataset used for detecting malware. This is similar to the NetML data and but captured from a different source. Here 80% of the observations are malware while the rest are benign traffic.

4.4. Datasets for Network traffic analysis

The non-VPN dataset has seven classes of non-VPN network traffic. The dataset also has 121 columns/similar to the other two datasets. Since this is a curated dataset there are no null columns and no further Null treatment is needed. The seven classes in this dataset are P2P, audio, chat, email, file_transfer, tor, video.

4.5. Data Preparation

In this section, all the steps that are involved in preparing the data for building the models for 3 different datasets are elaborated.

4.5.1. Drop unwanted Features

In the first iteration the source and destination ports (src_port and dst_port) are dropped from the source dataset as having these ports for malware or traffic classification does not make much sense and we want to base our classification only on the actual network traffic flows. All other variables are continuous variables and no additional handling is needed for these variables.

4.5.2. Treatment of Missing Values

No missing value treatment is needed.

4.5.3. Univariate and Bi-Variate Analysis

This analysis was done on all three different datasets. The list of features is huge univariate analysis has been added in the Appendix. The frequency distribution of all features in the dataset indicates they are continuous variables as these are the values that have been extracted from the network traffic flows. Further analysis also revealed that they are no significant outliers that might warrant their removal from the initial set of features that are considered for modeling. Any subsequent of removal unwanted features are analyzed further after the models are built and using the Explainable AI. The histograms from the univariate analysis can found in the Appendix sections.

The correlation between the variables reveals that in a majority of the cases there is no significant correlation. In a few cases where there is a positive and negative correlation, it is difficult to decide which of those correlated variables are important to be retained. Given that the short-listed models for our work are Random Forest, XGBoost, and Neural Networks these models are not significantly impacted by multicollinearity in data (De Veaux and Ungar, 1994). Considering these two points all the features are considered for the first iteration of model building. Refer to the Appendix section for the heatmaps.

4.5.4. Scaling and Standardization

Univariate analysis reveals the big range in values for a significant number of the features and is different for most of the features. These large values lead to the model learning large weights leading to greater instability of the model. To address this feature scaling needs to be applied. Given that almost all the features are not normally distributed MinMax Scaler is used to scale the data instead of Standard Scalar.

4.5.5. Splitting of the original dataset

The datasets are split into train and test data in an 80:20 ratio.

4.6. Exploratory Data Analysis and Visualization

Various exploratory data analysis has to be performed on complete to draw further insights from the datasets. Since the data is not normally distributed no further insights could be obtained and no further data preprocessing can be performed.

The univariate analysis using distribution plots and box plots does not reveal significant insights and no outlier treatment is possible as well, while the bivariate analysis revealed that most of the features are not significantly correlated except a few features. All these plots are listed down in the Appendix section.

4.7. Summary

The dataset is converted from JSON format to tabular format. Since it is a curated dataset no additional treatment on the columns is needed. Two columns related to port numbers have been dropped as it is not so relevant for Malware detection and Network Traffic Analysis. Univariate and Bi-variate analysis has been performed on the datasets. Given that these datasets are not normally distributed no significant insights could be drawn. Given the wide variance in the values for each of the features, MinMax Scaler is used on all features during Model building. Hence it has been decided to use all the features for modeling, except for src_port and dst_port are dropped.

Once the dataset has been split, Class Imbalance in the training dataset is handled using ADASYN and SMOTE to verify if better accuracy and F1-Score can be achieved.

5. RESULTS AND DISCUSSIONS

5.1. Introduction

After the above Data-preprocessing steps as mentioned, the class imbalance in the data is handled using ADASYN and SMOTE. These models are built using XGBoost, Random Forest, and Neural Networks to achieve greater accuracy and low false positives and negatives.

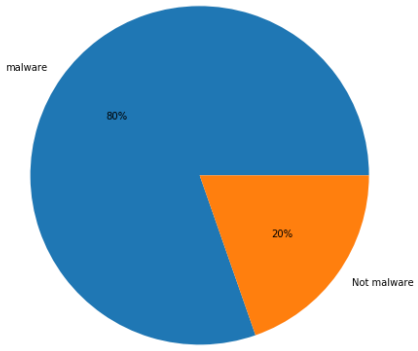
The dataset is imbalanced as there are a greater number of malware samples than benign samples.

5.2. Evaluation of Sampling Methods

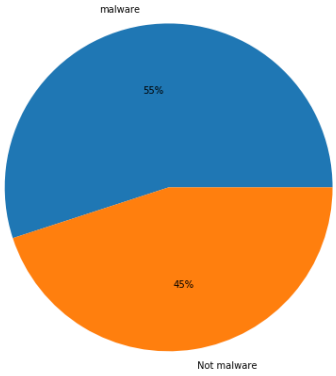
5.2.1. Class Imbalance

Below is the ratio of different classes in the dataset. This data is class balanced using the ADASYN class balancing technique on all three datasets on the NetML dataset. As the dataset is not normally distributed the dataset is also class balanced using the SMOTE to compare how models based on these two different approaches fare along with the models that build without any class balancing.

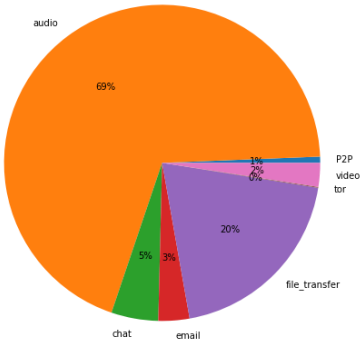
NetML Dataset



CICIDS Dataset



Non-vpn Dataset



5.3. Model Evaluation and Explainable AI

5.3.1. NetML Dataset – Machine Learning Models

For the NetML dataset from the below results, it can be concluded that XGBoost with default parameters performs marginally better than Random Forest and Neural Networks. The model with default parameters achieved significantly high accuracy and AUC. This data has also been class balanced using ADASYN and SMOTE and it is evident that the SMOTE Accuracy/AUC is almost the

same or has improved marginally. The F1-Score for the XGBoost model based on SMOTE is also better than the F1-Score for the model that is not class balanced. ADASYN, given the nature of the dataset, which is not normally distributed, has resulted in reduced accuracy and is not suitable for this dataset.

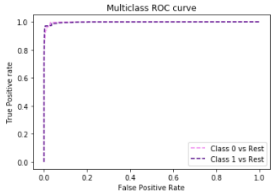
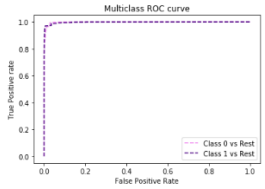
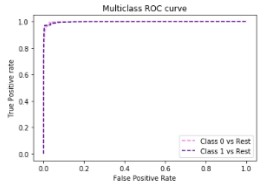
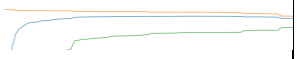
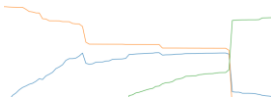
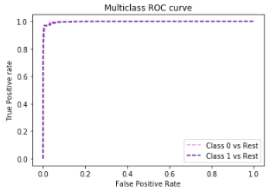
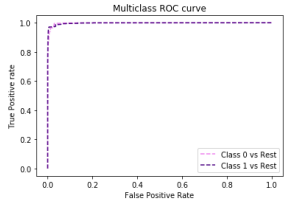
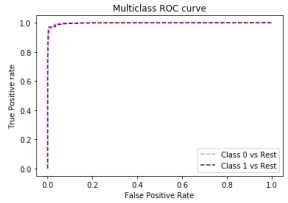
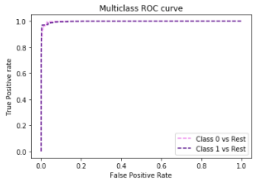
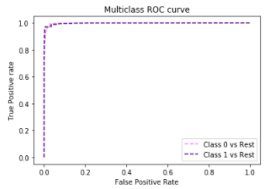
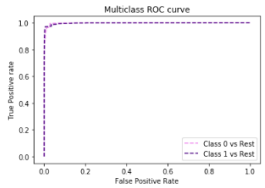
For the XGBoost and Random Forest models, the validation of the model has been done using the *Stratified K-Fold validation* so that the accuracy of the model is better accessed while building the model. Stratification shuffles the training data after a new split of data is done. This significantly reduces the chance of overfitting the model during the training validation itself. Since this is a binary classification problem, an optimal probability cut-off for the class probabilities predicted by the model can be arrived at by plotting the sensitivity, specificity, and accuracy of the model which are in red, blue, and green colors respectively in the captured results in the below table. *GridSearchCV* did not lead to finding optimization parameters that are better than the XGBoost and Random Forest default parameters. While evaluating the data using the test data this probability cut-off is used to classify the observations into the malware of benign. These results revealed there is no significant overfitting of the model.

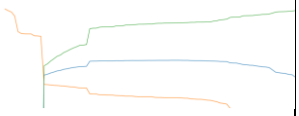
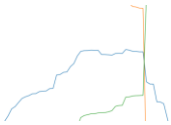
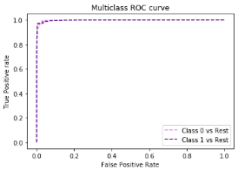
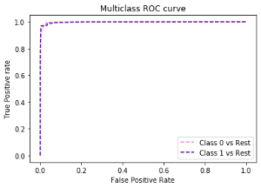
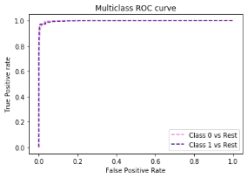
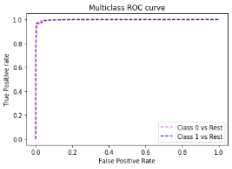
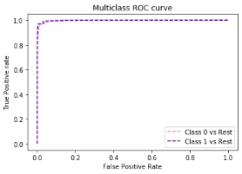
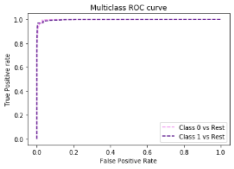
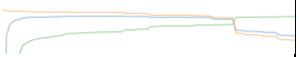
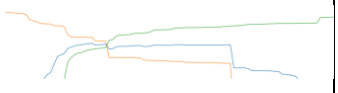
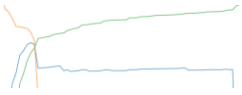
The architecture of the Neural Networks model has *8 dense hidden layers*, where the number of neurons in each of the hidden layers is equal to the number of neurons in the input layers. This approach helps in minor incremental improvement in the accuracy of the model. The output layers *softmax* as the activation function as we need the sum of probabilities as one, for a given observation. The dropouts are not considered as no overfitting of the model is observed when verified with the test data. Though the actual expectation is for Neural networks to perform better than XGBoost, the results show that for the given tabular dataset Neural networks did not bring any additional benefits in classification tasks either from the speed of the training or improvement accuracy of the model.

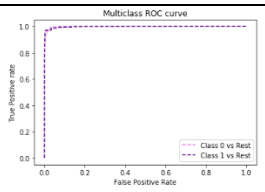
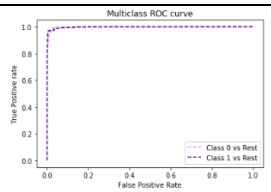
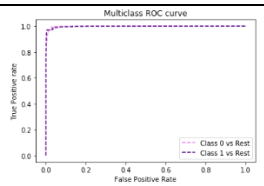
Note that the confusion matrix from the Sklearn library is in the below format:

[TN FP
FN TP].

Table 2: Results from Models for NetML Dataset

Description		**XGBoost**	**RandomForest**	**Neural Network**
Original Dataset without Class balancing				
Training Metrics – Original dataset	**AUC & ROC**	**AUC**: 99.99% 	**AUC**: 99.98% 	**AUC**: 99.82% 
	Optimal Probability Cut-off	0.88: 	0.92: 	0.82: 
	Confusion Matrix	[[60613 267] [776 248158]]	[[60679 201] [1940 246994]	[[60548 332] [8143 240791]]
Test Metrics - Original dataset	**AUC & ROC**	**AUC :** 99.98% 	**AUC:** 99.97% 	**AUC:** 99.81% 
	Confusion Matrix	[[15042 73] [193 62146]]	[[15066 49] [544 61795]]	[[15029 86] [2047 60292]]
	F1-Score	0.9912 , .9897	0.9807 , 0.9952	0.9337 , 0.9826
Class balancing using ADASYN				
Training Metrics - with	**Average K-Fold (5 Folds) AUC & ROC**	AUC: 99.65% 	AUC: 98.64% 	AUC: 98.17% 

ADASYN	**Optimal Probability Cut-off**	0.13: 	0.61: 	0.63: 
	Confusion Matrix	[[244838 4322] [5946 242988]]	[[233510 15650] [9214 239720]]	[[14981 134] [2640 59699]]
Test Metrics - with ADASYN	**AUC & ROC**	AUC: 99.96% 	AUC: 99.65% 	AUC: 99.70% 
	Confusion Matrix	[[15062 53] [1504 60835]]	[[15092 23] [2375 59964]]	[[14981 134] [2640 59699]]
	F1-Score	0.9508, 0.9873	0.9264 , 0.9803	0.9154, 0.9772
Class balancing using SMOTE				
Training Metrics – with SMOTE	**Average K-Fold (5 Folds) AUC & ROC**	AUC: 99.99% 	AUC: 99.91% 	AUC: 99.73% 
	Optimal Probability Cut-off	0.7: 	0.3: 	0.08: 
	Confusion Matrix	[[15045 70] [158 62181]]	[[246887 2047] [3316 245618]]	[[244759 4175] [8286 240648]]
	AUC & ROC	AUC: 99.98%	AUC: 99.91%	AUC: 99.84%

Test Metrics – with SMOTE				
	Confusion Matrix	[[15045 70] [158 62181]]	[[14969 146] [810 61529]]	[[13811 1304] [247 62092]]
	F1-Score	0.9924, 0.9981	0.9690, 0.9922	0.9468, 0.9876

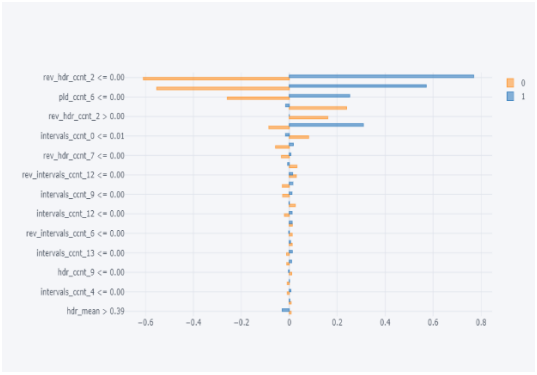
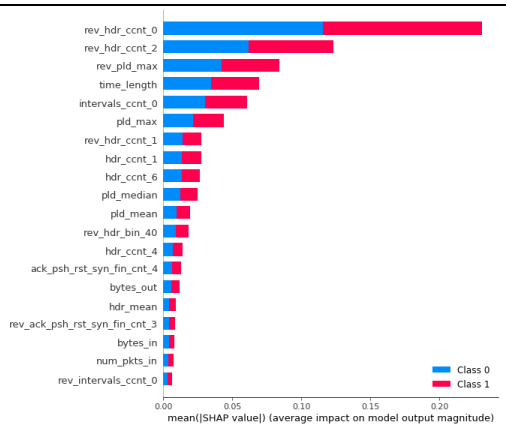
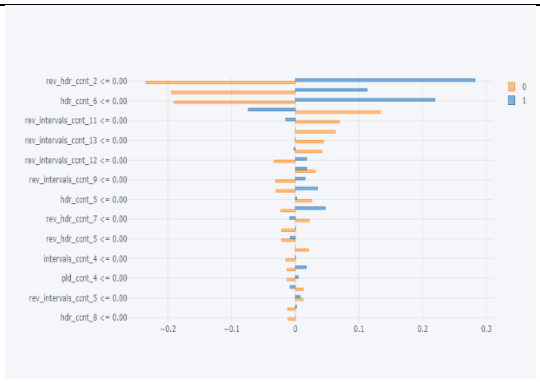
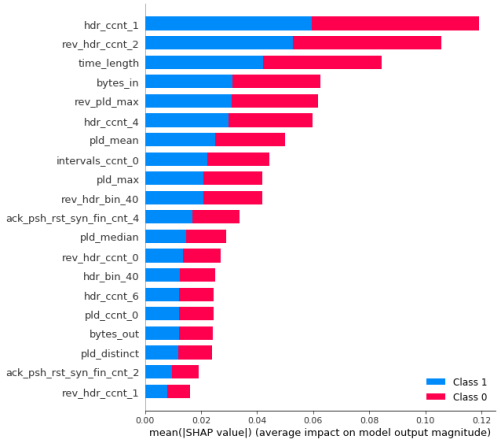
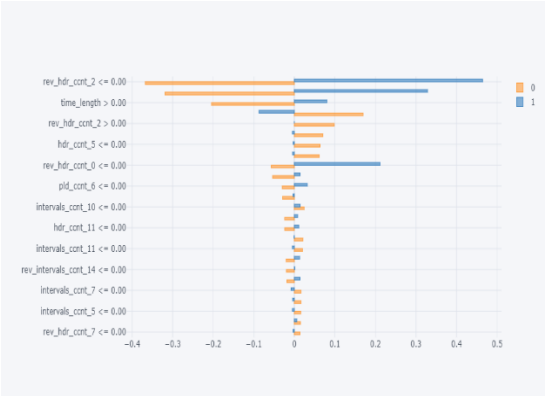
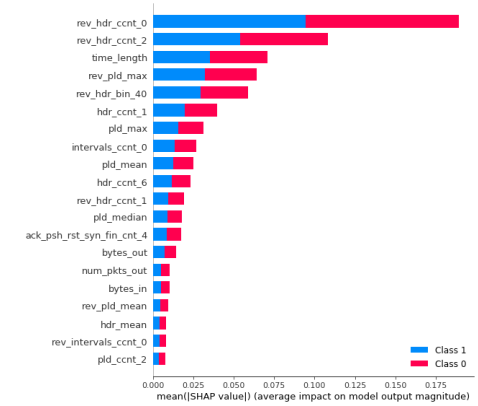
5.3.2. NetML Dataset – Explainable AI Methods

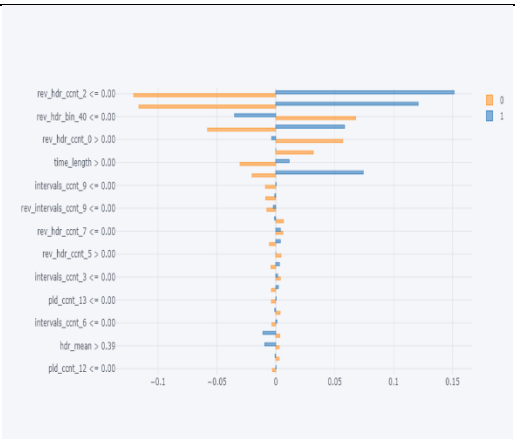
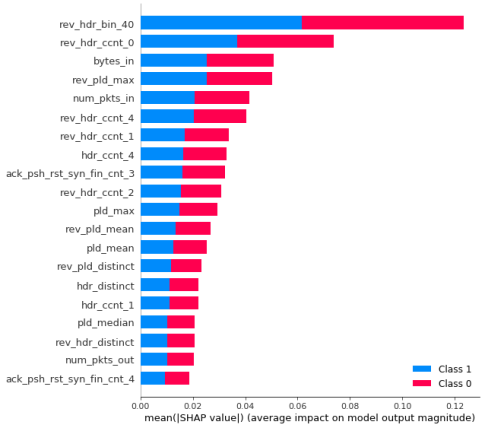
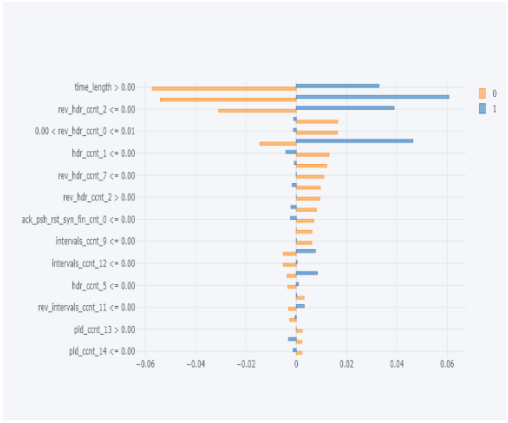
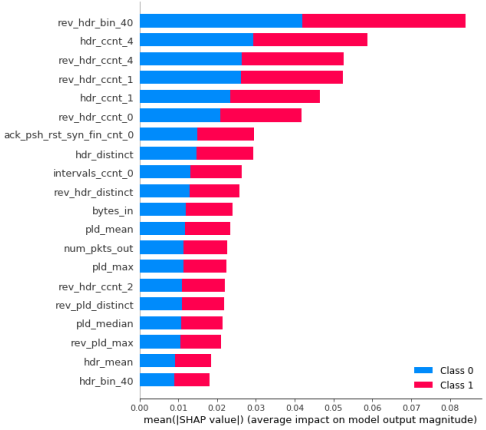
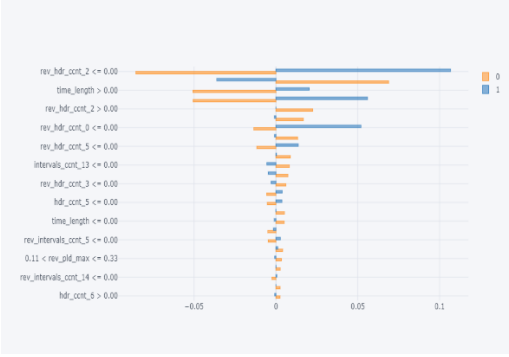
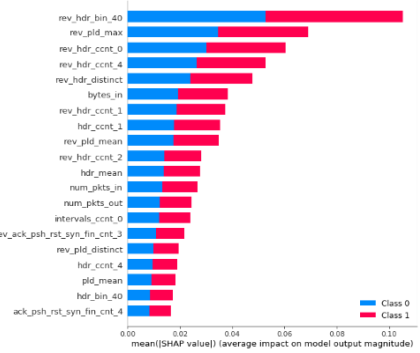
The below table depicts the explanations for different models using the 100 random observations from the test data. It is clear from this that the LIME explanations are different from the SHAP explanations they also overlap to a moderate extent. The most important features also differ in the order of importance between both the explanations.

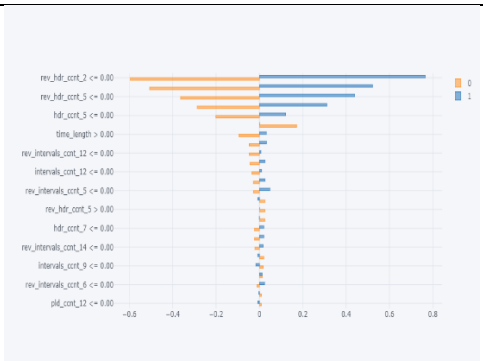
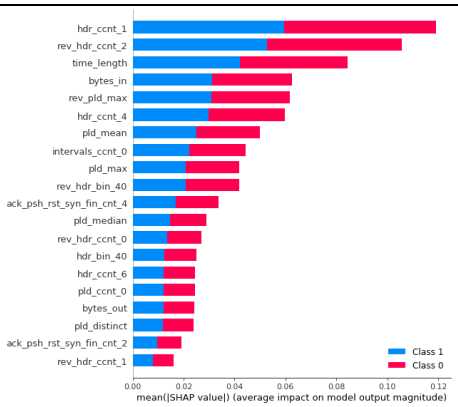
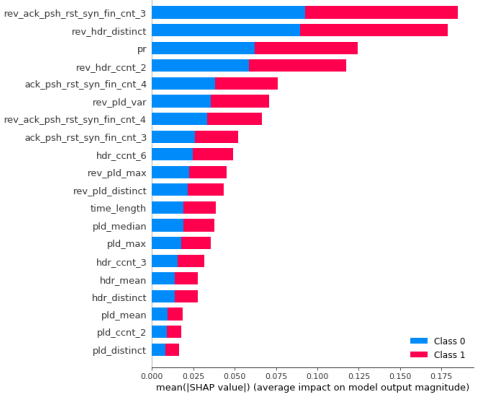
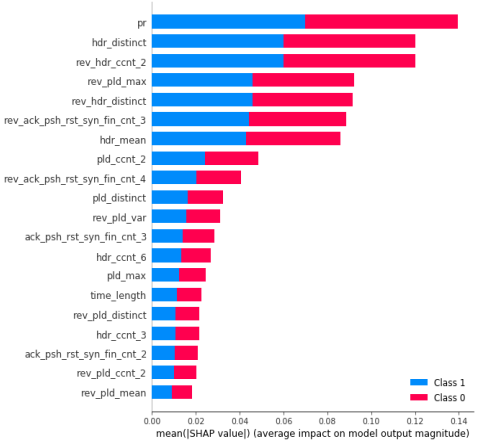
Explanations for the models built from the original dataset and the class balanced datasets ADASYN and SMOTE are compared for each of the XGBoost, RandomForest, and Neural Networks algorithms. it is quite evident that the explanations for models that are not class balanced and SMOTE-balanced models are almost similar and are more logical as it includes some of the key features like rev_hdr_ccnt_0,rev_hdr_ccnt_2,timelength, bytes_in, _bytes_out. The explanations for models built using ADASYN are different and models have lower accuracy.

The summary from these explanations is using either the original data set or class balanced dataset using SMOTE is leading to similar model AUC and similar model explanations. The F1-Score for the models built with class-balanced datasets using SMOTE is slightly better. Hence for Malware classifications XGBoost using SMOTE is good for better accuracy and better explanations.

Table 3 : Model Explanations for NetML Dataset

Model	LIME Explanations	SHAP Explanations
XGBoost Default parameters		
XGBoost with ADASYN		
XGBoost with SMOTE		

Random Forest Default Params		
Random Forest - ADASYN		
Random Forest - SMOTE		

Neural Networks Default		
Neural Networks with ADASYN		
Neural Networks with SMOTE		

5.3.3. CICIDS Dataset – Machine Learning Models

Similar to the NetML dataset, for the CICIDS dataset it is evident that XGBoost with default parameters performs marginally better than Random Forest and Neural Networks. The default

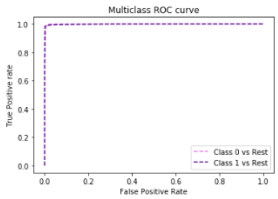
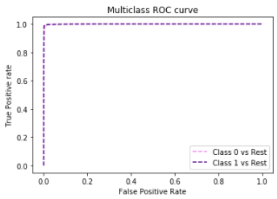
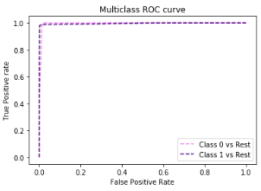
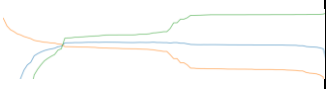
parameters achieve significantly high accuracy and AUC. Models were built with this dataset after class balancing using ADASYN and SMOTE. It revealed that the SMOTE Accuracy/AUC is almost the same or has improved marginally when compared to models built using the default dataset. The F1-Score for SMOTE is close to the F1-Score for models built with the original dataset, either marginally smaller or larger. Models built using ADASYN have resulted in reduced accuracy and are not suitable for this dataset. The F1-Score for the XGBoost model based on SMOTE is close to the F1-Score for the original dataset, either marginally smaller or larger. If the intention is to achieve as high accuracy as much possible SMOTE is more suitable to class balance the data, else the using default dataset to build the model is also good enough.

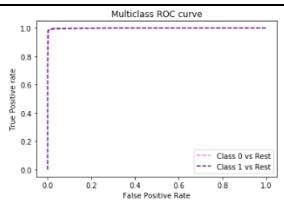
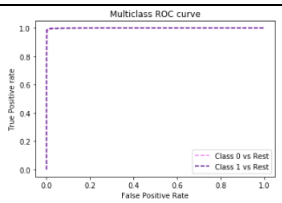
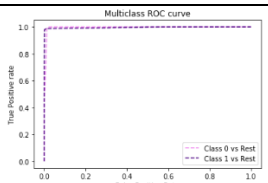
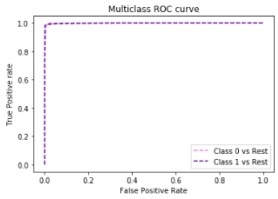
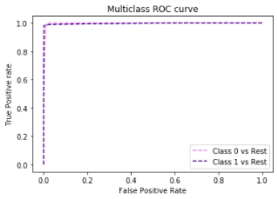
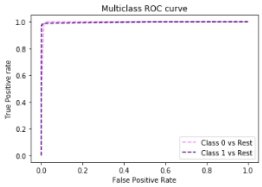
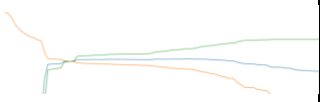
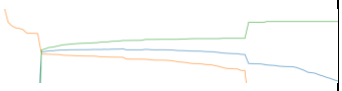
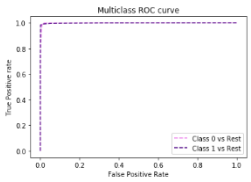
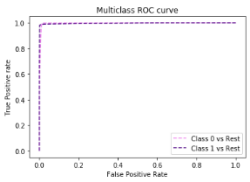
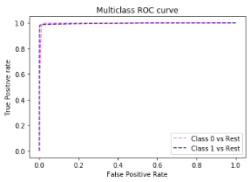
The approach to building the XGBoost, RandomForest, and Neural Networks models is very similar to the approach applied for the NetML dataset that is mentioned in section 5.3.1.

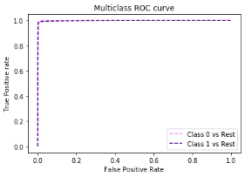
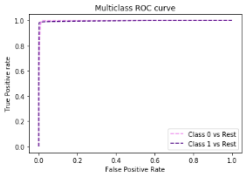
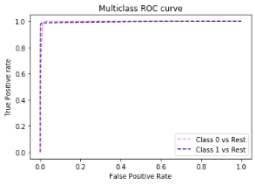
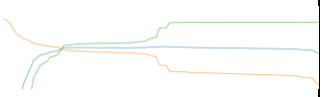
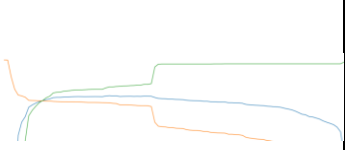
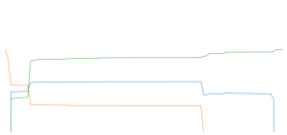
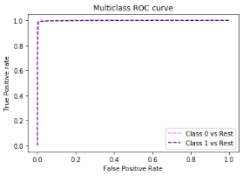
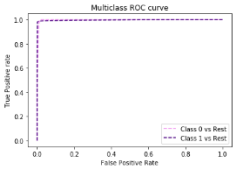
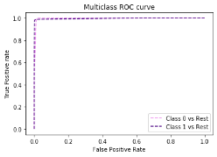
Note that the confusion matrix from the Sklearn library is in the below format:

[TN FP
FN TP]

Table 4 : Model Results for CICIDS Dataset

Description		**XGBoost**	**RandomForest**	**Neural Network**
Original Dataset without Class balancing				
Training Metrics – Original dataset	**Average K-Fold (5 Folds) AUC & ROC**	AUC: 99.89% 	AUC: 99.94% 	AUC: 99.62% 
	Optimal Probability Cut-off	0.2 	0.42 	0.22
	Confusion Matrix	[[157650 1292] [1857 192093]]	[[157698 1244] [1396 192554]]	[[157075 1867] [2306 191644]]
Test Metrics -	**AUC & ROC**	AUC: 99.84%	AUC: 99.81%	AUC: 99.61%

Original dataset				
	Confusion Matrix	[[39185 328] [527 48184]]	[[39188 325] [503 48208]]	[[39079 434] [590 48121]]
	F1-Score	0.9822 , 0.9912	0.9855 ,	0.9870 , 0.9914
Class balancing using ADASYN				
Training Metrics - with ADASYN	**Average K-Fold (5 Folds) AUC & ROC**	AUC: 99.71% 	AUC: 99.52% 	AUC: 99.46% 
	Optimal Probability Cut-off	0.2 	0.12 	0.1 
	Confusion Matrix	[[191044 2959] [2963 190987]]	[[190844 3159] [3373 190577]]	[[190752 3251] [3518 190432]]
Test Metrics - with ADASYN	**AUC & ROC**	AUC: 99.83% 	AUC: 99.67% 	AUC: 99.67% 
	Confusion Matrix	[[39353 160] [800 47911]]	[[39367 146] [861 47850]]	[[39405 108] [926 47785]]
	F1-Score	0.9879	0.9873 , 0.9900	0.9870 , 0.9892
Class balancing using SMOTE				

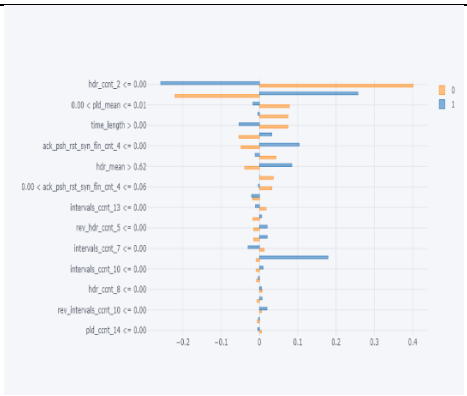
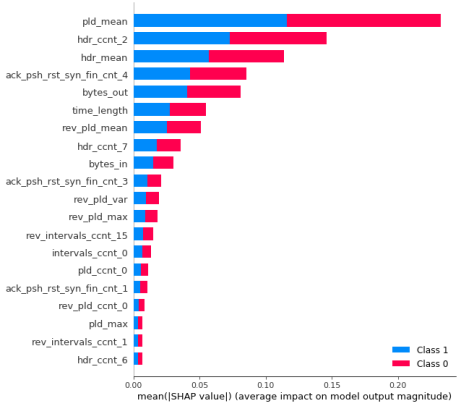
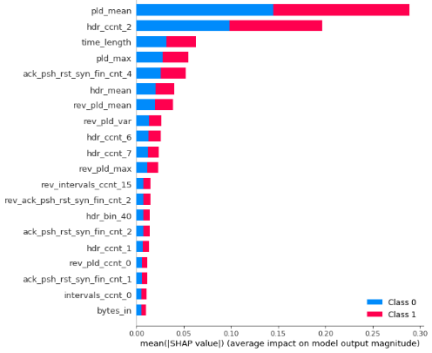
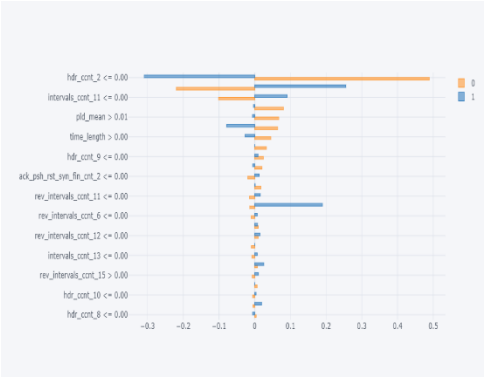
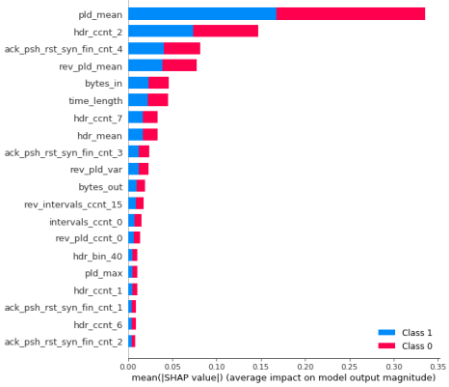
Training Metrics – with SMOTE	**Average K-Fold (5 Folds) AUC & ROC**	AUC: 99.90% 	AUC: 99.76% 	AUC: 99.68% 
	Optimal Probability Cut-off	0.19 	0.12 	0.08 
	Confusion Matrix	[[192402 1548] [1860 192090]]	[[191680 2270] [2249 191701]]	[[190886 3064] [2296 191654]]
Test Metrics – with SMOTE	**AUC & ROC**	AUC: 99.83% 	AUC: 99.69% 	AUC: 99.67% 
	Confusion Matrix	[[39178 335] [534 48177]]	[[39080 433] [581 48130]]	[[39108 405] [591 48120]]
	F1-Score	0.9890 , 0.9892	0.9871 , 0.9895	0.9874 , 0.9897

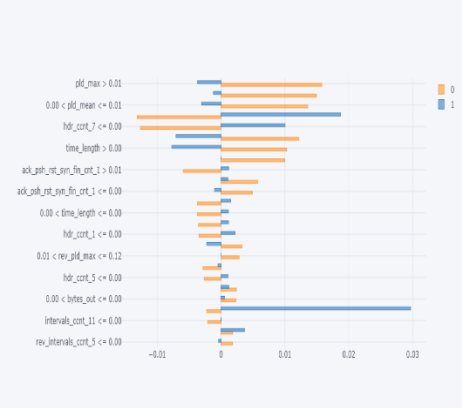
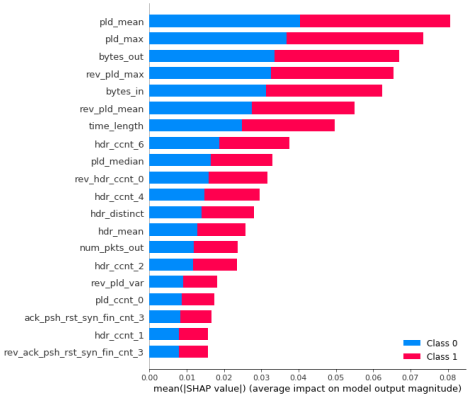
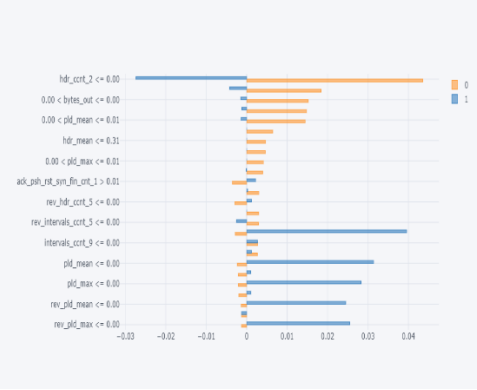
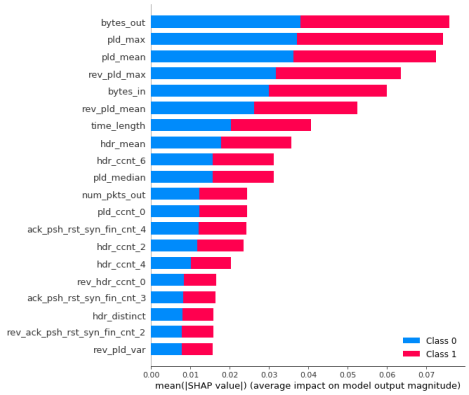
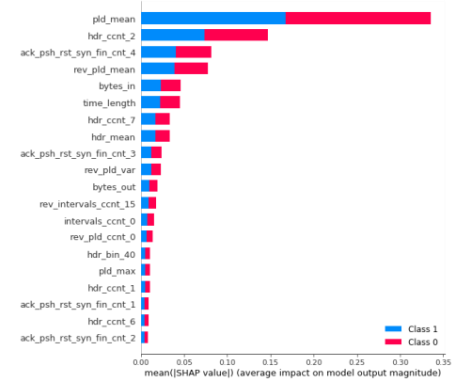
5.3.4. CICIDS Dataset – Explainable AI

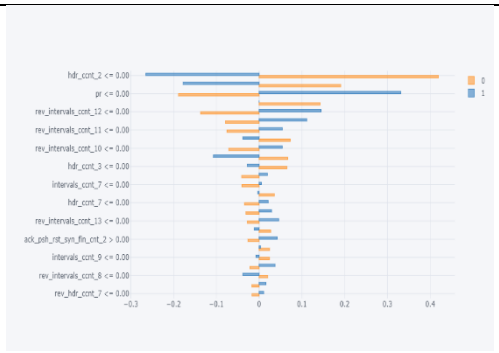
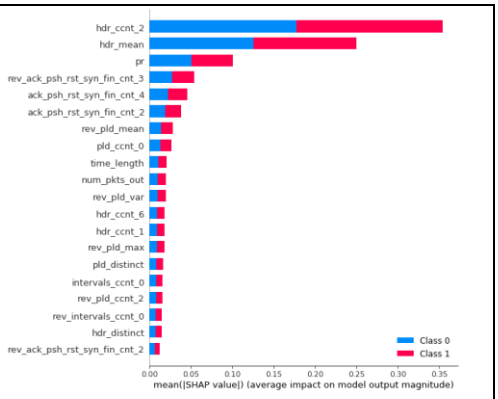
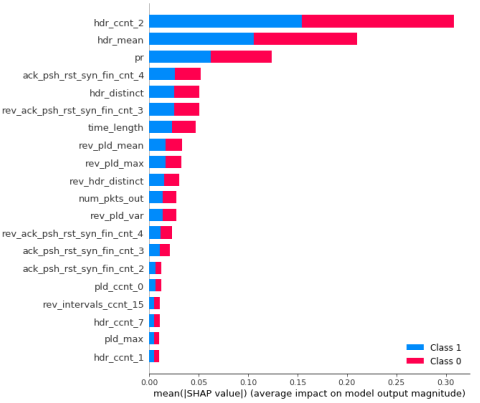
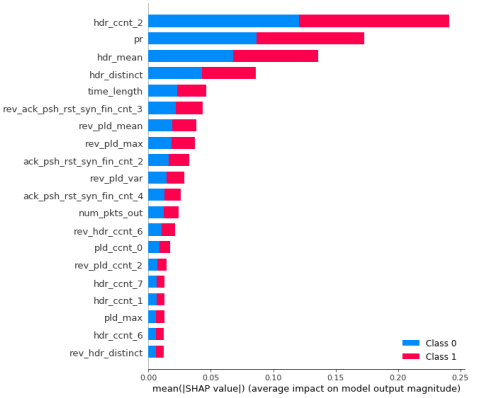
LIME and SHAP different they also overlap to a moderate extent and they also differ in the order of importance. Explanations for the models built from the original data and the datasets balanced with ADASYN and SMOTE are compared.

There is significant overlap between these explanations, though it can be said concluded the explanations for models when using default and SMOTE balanced datasets are closer when compared to the ADASYN. In these explanations, the key features that contribute to the classification of Malware are hdr_ccnt_2, pr, hdr_mean, hdr_distinct, time_length, rev_ack_psh_rst_syn_fin_cnt_3, rev_pld_mean, bytes_in, bytes_out.

Table 5 : Model Explanations for CICIDS Dataset

Model	LIME Explanations	SHAP Explanations
XGBoost Default parameters		
XGBoost with ADASYN		
XGBoost with SMOTE		

Random Forest Default Params		
Random Forest - ADASYN		
Random Forest - SMOTE		

Neural Networks Default		
Neural Networks with ADASYN		
Neural Networks with SMOTE		

5.3.5. Non-VPN Dataset – Machine Learning Models

For the non-VPN dataset from the below results, it can be concluded that XGBoost with default parameters without any class-balancing of the training data performs better than Random Forest and Neural Networks. XGBoost model with default parameters achieved reasonably high F1-Score for

class 0(P2P) and class 1(Audio). The F1-Score for class 6(Video) score is just above fifty percent which is not reasonably good for classification. For other classes, class 2 (Chat), class 3(Email), class 4(File transfer), and class 5(TOR) the F1-score is very low and is not suitable for classification. Achieving a reasonably good F1- score for these classes might need more feature engineering, which is needed for use-cases where high sensitivity and specificity of the model are important. This though falls short on the F1-Score for five of the seven classes, it has achieved good precision for 6 out of 7 classes. Models that have been built not only from the original dataset and models also have been class balanced using ADASYN and SMOTE. It is observed that the ADASYN and SMOTE-based models' Accuracy/F1-Score has decreased when compared to models that are built without any class balancing. For the nature of this dataset, where the top significant feature values are overlapping significantly, classifications from models based on Random Forest and Neural Networks are not up to mark. The F1-Score for the SMOTE-based models is lesser than the F1-Score for the original dataset. That way, these results for Network Traffic Classification are different from the Malware Detection.

For evaluating the classification of the model, three different metrics *Accuracy*, *Macro F1-Score*, and *Weighted Macro F1-Score* are considered. For this, a multi-class confusion matrix is generated from the test results. This is also called *OVR(one over the rest)* classification. The accuracy of the model just tells how many observations from test data have been predicted accurately. Macro F1-Score is the average of the F1-Score for each of the classes. Weighted Macro F1-Score gives more weight to the class based on the number of observations in that class. Based on these metrics, XGBoost with default parameters without any class balancing resulted in better metrics than all the approaches considered. Usage of GridSearchCV to find out better optimization and regularization parameters did not yield any better results.

Table 6 : Model Results for Non-VPN Dataset

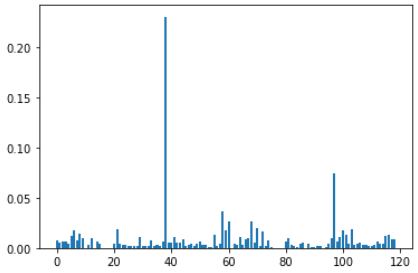
Description		**XGBoost**	**RandomForest**	**Neural Network**
Original Dataset without Class balancing				
Training Metrics – Original dataset	**Accuracy**	Accuracy: 72.36% (Stratified K Folds, 5 folds)	Accuracy: 70.91% (Stratified K Folds, 5 folds)	Accuracy: 70.98% (50 epochs)
	Confusion Matrix for classes 0 to 6	[[[104153, 35], [142, 522]], [[5360, 27042], [265, 72185]], [[99331, 435], [3697, 1389]],	[[[104156 32] [133 531]] [[8880 23522] [173 72277]] [[99344 422] [3564 1522]]	[[[103991 197] [324 340]] [[2761 29641] [378 72072]] [[99588 178] [4476 610]]

		[[101478, 47], [2807, 520]], [[84194, 26], [19857, 775]], [[104733, 14], [38, 67]], [[101973, 291], [1084, 1504]]]	[[101479 46] [2758 569]] [[84202 18] [16519 4113]] [[104729 18] [38 67]] [[102100 164] [1037 1551]]]	[[101518 7] [3089 238]] [[84101 119] [20264 368]] [[104745 2] [105 0]] [[101990 274] [1782 806]]]
Test Metrics - Original dataset	**Accuracy**	Accuracy: 72.50%	Accuracy: 22.91%	Accuracy: 70.81%
	Confusion Matrix for classes 0 to 6	[[[26027 18] [61 107]] [[1148 6851] [140 18074]] [[24852 152] [899 310]] [[25361 20] [737 95]] [[21010 33] [5030 140]] [[26190 6] [15 2]] [[25482 128] [326 277]]]	[[[25865 180] [20 148]] [[7472 527] [16086 2128]] [[18180 6824] [648 561]] [[20604 4777] [402 430]] [[14461 6582] [2974 2196]] [[26140 56] [8 9]] [[24350 1260] [68 535]]]	[[[25986 59] [102 66]] [[590 7409] [131 18083]] [[24956 48] [1073 136]] [[25380 1] [788 44]] [[21010 33] [5103 67]] [[26195 1] [17 0]] [[25511 99] [436 167]]]
	F1-Score	**class f1-score** 0 0.73 1 0.84 2 0.37 3 0.20 4 0.05 5 0.16 6 0.55 Macro Avg: 0.41 Weighted Avg: 0.63	**class f1-score** 0 0.60 1 0.18 2 0.13 3 0.14 4 0.32 5 0.22 6 0.45 Macro Avg: 0.29 Weighted Avg: 0.23	**class f1-score** 0 0.45 1 0.83 2 0.20 3 0.10 4 0.03 5 0.00 6 0.38 Macro Avg: 0.28 Weighted Avg: 0.60
Class balancing using ADASYN				
Training Metrics - with ADASYN	**Accuracy**	Accuracy: 60.22% (Stratified K Folds, 5 folds)	Accuracy: 57.36% (Stratified K Folds, 5 folds)	Accuracy: 53.73% (50 epochs)
	Confusion Matrix for classes 0 to 6	[[[435493 1610] [2270 70155]] [[428469 8609] [63164 9286]] [[371021 65657] [34637 38213]] [[390507 46067] [35958 36996]] [[386613 48880] [41316 32719]] [[434022 3053] [484 71969]] [[429789 7378] [3425 68936]]]	[[[435858 1245] [2093 70332]] [[436964 114] [58640 13810]] [[409576 27102] [54682 18168]] [[285181 151393] [10700 62254]] [[434947 546] [59938 14097]] [[434294 2781] [456 71997]] [[430621 6546] [3218 69143]]]	[[[421353 15750] [4494 67931]] [[434854 2224] [68333 4117]] [[259114 177564] [8758 64092]] [[430537 6037] [63653 9301]] [[425371 10122] [63429 10606]] [[429531 7544] [19636 52817]] [[420685 16482] [7420 64941]]]
Test Metrics - with ADASYN	**Accuracy**	Accuracy: 22.91%	Accuracy: 17.21%	Accuracy: 13.87%
	Confusion Matrix for classes 0 to 6	[[[25865 180] [20 148]] [[7472 527] [16086 2128]] [[18180 6824] [648 561]] [[20604 4777] [402 430]] [[14461 6582] [2974 2196]]	[[[25892 153] [19 149]] [[7442 557] [15981 2233]] [[22270 2734] [889 320]] [[9631 15750] [117 715]] [[19854 1189] [4611 559]]	[[[25792 253] [25 143]] [[7826 173] [17145 1069]] [[6772 18232] [245 964]] [[25108 273] [675 157]] [[18951 2092] [4385 785]]

		[[26140 56] [8 9]] [[24350 1260] [68 535]]]	[[26140 56] [9 8]] [[24349 1261] [74 529]]]	[[26010 186] [4 13]] [[24244 1366] [96 507]]]
	F1-Score	**Class f1-score** 0 0.60 1 0.20 2 0.13 3 0.14 4 0.31 5 0.22 6 0.45 Macro Avg: 0.29 Weighted Avg: 0.23	**Class f1-score** 0 0.63 1 0.21 2 0.15 3 0.08 4 0.16 5 0.20 6 0.44 Macro Avg: 0.27 Weighted Avg: 0.20	**Class f1-score** 0 0.51 1 0.11 2 0.09 3 0.25 4 0.20 5 0.12 6 0.41 Macro Avg: 0.24 Weighted Avg: 0.14
Class balancing using SMOTE				
Training Metrics – with SMOTE	**Accuracy**	Accuracy: 65.87% (Stratified K Folds, 5 folds)	Accuracy: 63.81% (Stratified K Folds, 5 folds)	Accuracy: 59.49% (50 epochs)
	Confusion Matrix for classes 0 to 6	[[[433479 1221] [908 71542]] [[415131 19569] [58082 14368]] [[382113 52587] [31028 41422]] [[403849 30851] [41649 30801]] [[379039 55661] [34678 37772]] [[432179 2521] [356 72094]] [[427529 7171] [2880 69570]]]	[[[433685 1015] [834 71616]] [[434593 107] [58661 13789]] [[402063 32637] [41307 31143]] [[433977 723] [59272 13178]] [[305704 128996] [9195 63255]] [[432382 2318] [324 72126]] [[428376 6324] [2527 69923]]]	[[[430624 4076] [1905 70545]] [[293562 141138] [16552 55898]] [[424600 10100] [53642 18808]] [[407112 27588] [53270 19180]] [[430528 4172] [68074 4376]] [[429280 5420] [4920 67530]] [[421778 12922] [7053 65397]]]
Test Metrics – with SMOTE	**Accuracy**	Accuracy: 28.28%	Accuracy: 28.43%	Accuracy: 58.94%
	Confusion Matrix for classes 0 to 6	[[[25886 159] [20 148]] [[6985 1014] [14847 3367]] [[20029 4975] [592 617]] [[21713 3668] [556 276]] [[13371 7672] [2707 2463]] [[26147 49] [9 8]] [[24349 1261] [67 536]]]	[[[25895 150] [19 149]] [[7416 583] [15949 2265]] [[22152 2852] [731 478]] [[25282 99] [726 106]] [[7264 13779] [1249 3921]] [[26145 51] [9 8]] [[24364 1246] [77 526]]]	[[[25739 306] [13 155]] [[2715 5284] [4218 13996]] [[24727 277] [927 282]] [[22629 2752] [642 190]] [[20303 740] [4865 305]] [[26084 112] [5 12]] [[24319 1291] [92 511]]]
	F1-Score	**Class f1-score** 0 0.62 1 0.30 2 0.18 3 0.12 4 0.32 5 0.22 6 0.45 Macro Avg: 0.31 Weighted Avg: 0.30	**Class f1-score** 0 0.64 1 0.22 2 0.21 3 0.20 4 0.34 5 0.21 6 0.44 Macro Avg: 0.32 Weighted Avg: 0.25	**Class f1-score** 0 0.49 1 0.75 2 0.32 3 0.10 4 0.10 5 0.17 6 0.42 Macro Avg: 0.34 Weighted Avg: 0.57

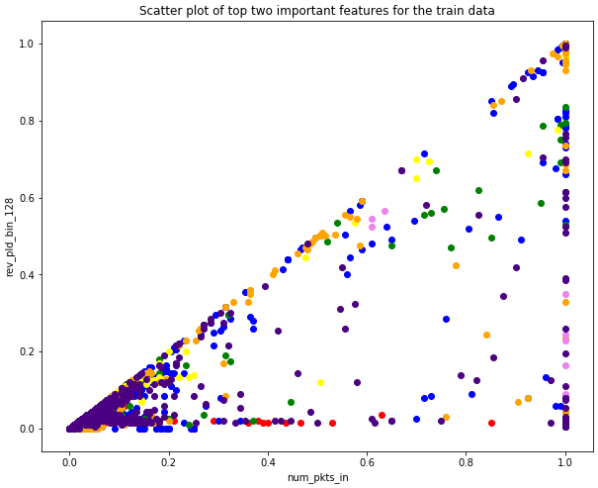
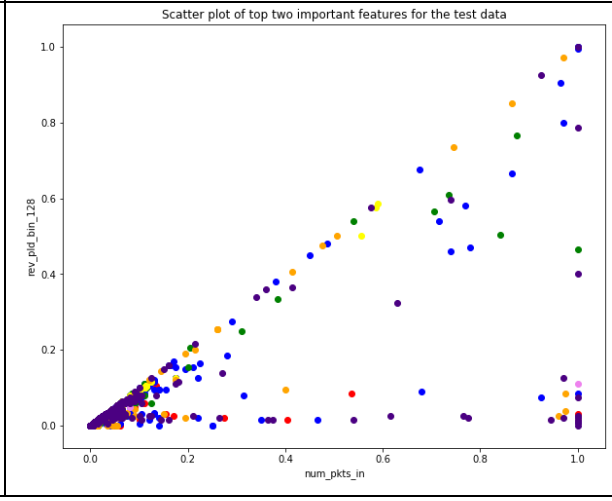
Now the next step looks at the reason for the low F1-Score for 5 of the 7 classes. Out of all the models, the XGBoost with default parameters without any class balancing performs better. The two

top important features from the XGBoost model are num_pkts_in and rev_pld_bin_128 as shown in the diagram.



By drawing a scatter plot between these two features, we can observe that there is a significant overlap between these features for most of the classes. Additional experiments may be needed to overcome this problem. Additional feature engineering may be required to overcome the problem of low training and accuracy/metrics and low performance with test data.

Table 7: XGBoost Important Features - Scatter Plot

Scatter Plot for top two features on training data	**Scatter Plot for top two features on test data**
	

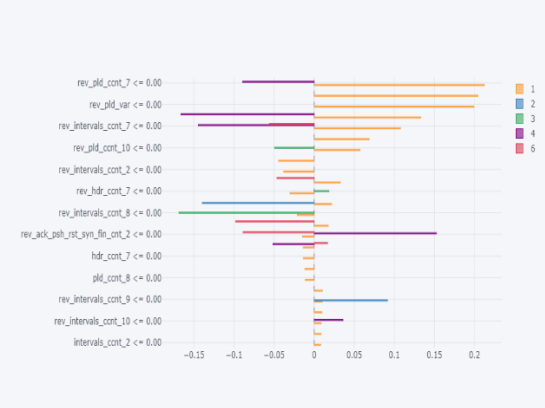
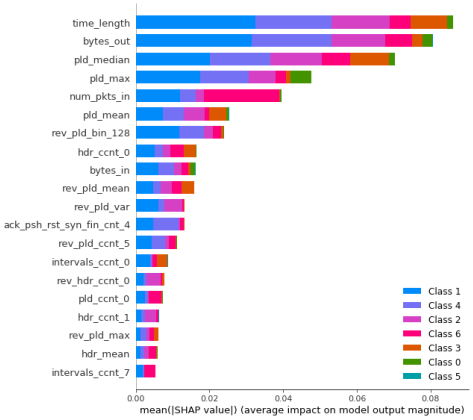
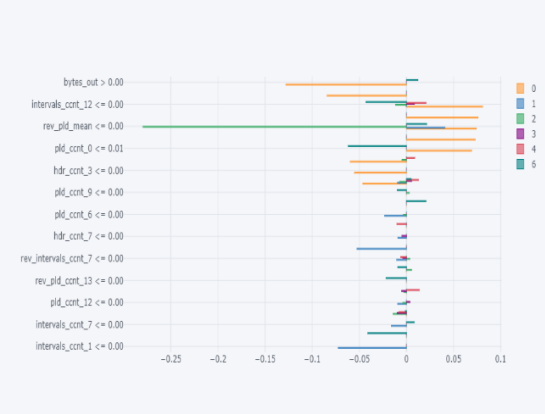
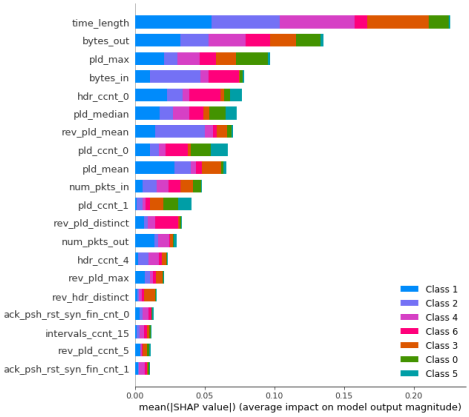
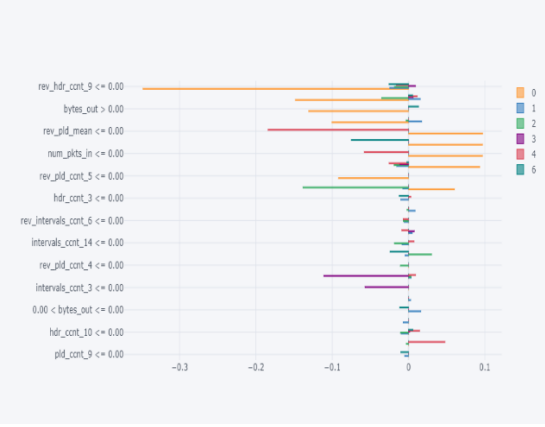
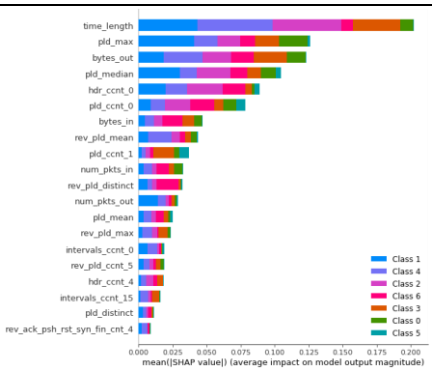
Color Coding Legend
Red – Class 0 - P2P
Blue – Class 1 - Audio
Green - Class 2 - Chat
Yellow – Class 3 - Email
Orange - Class 4 – File transfer
Violet – Class 5 - Tor
Indigo – Class 6 – Video

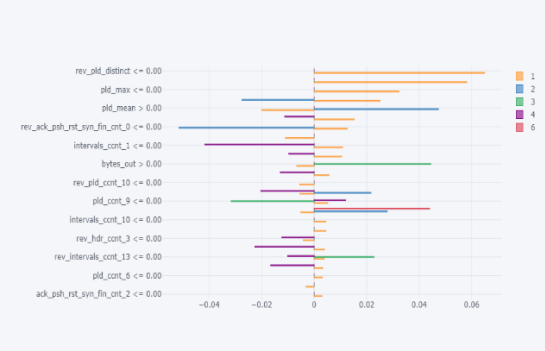
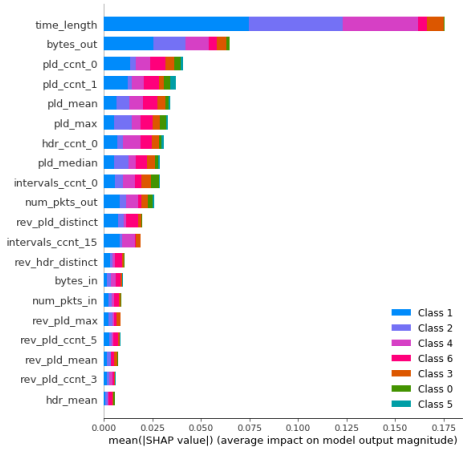
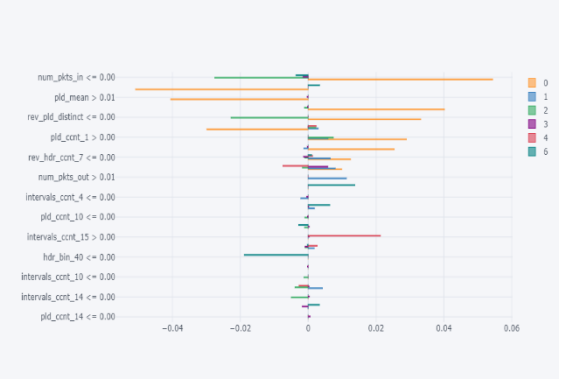
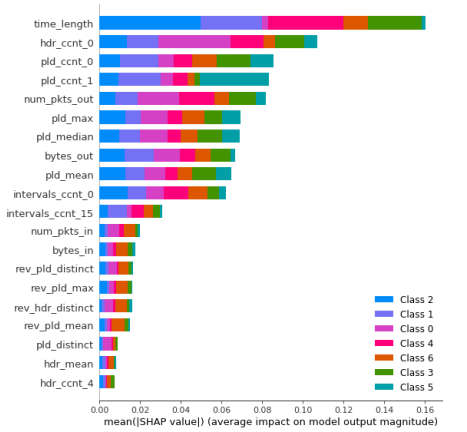
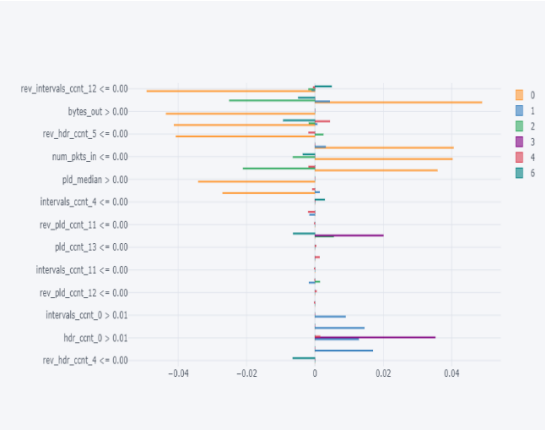
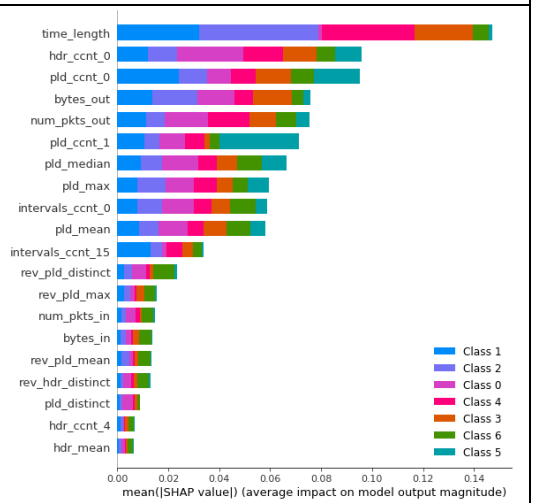
5.3.6. Non-VPN Dataset – Explainable AI Methods

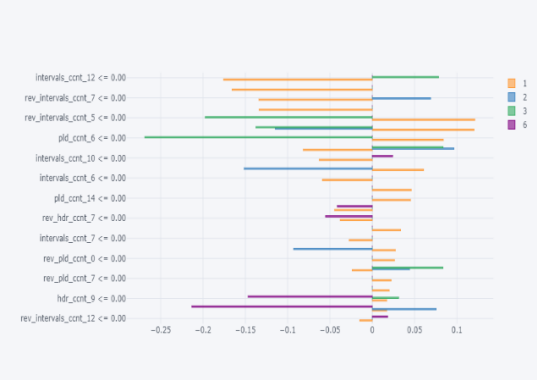
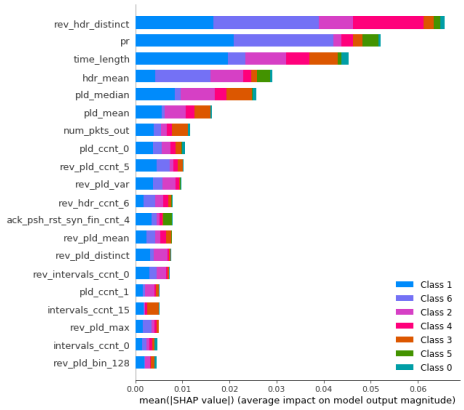
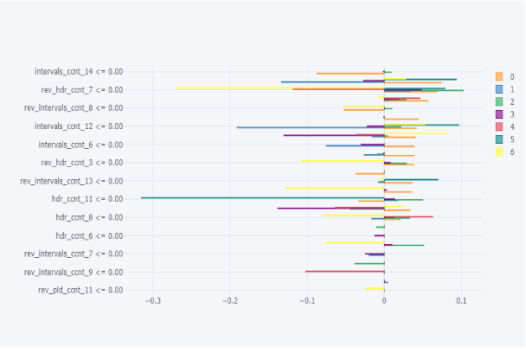
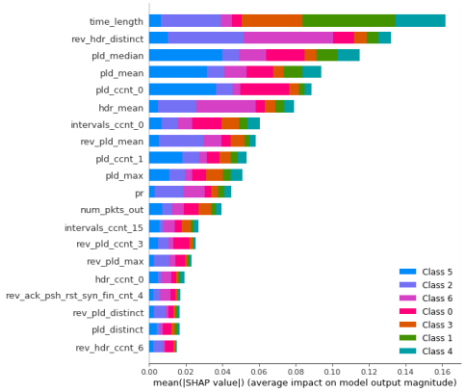
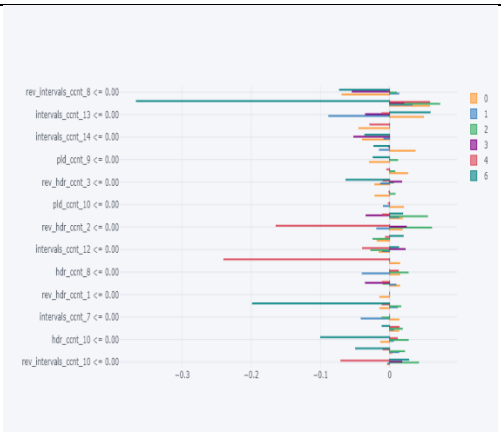
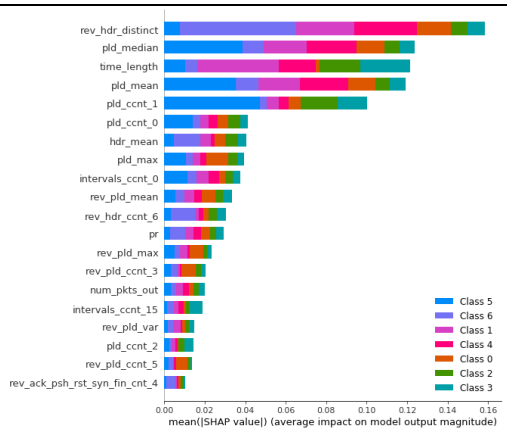
LIME and SHAP explanations significantly differ and SHAP explanations are more understandable as it includes important features like time_length, bytes_out, pld_median, pld_max. Similar to other models built with default datasets and that is SMOTE based offer similar explanations and look more realistic. Compared with the two other datasets, in this dataset, the ADASYN based model explanations are a bit closer to the default dataset and SMOTE-based models. But, from this, any concrete conclusions could not be drawn as the Accuracy of these models needs further improvement. Given that the XGBoost without any class balancing performed better on accuracy only these explanations can be considered. It has to be noted that the important features as shown by XGBoost

differ from the importance of the features inferred from SHAP, for the small set of observations for which explanations are being examined.

Table 8 : Model Explanations for Non-VPN Dataset

Model	**LIME Explanations**	**SHAP Explanations**
XGBoost Default parameters		
XGBoost with ADASYN		
XGBoost with SMOTE		

Random Forest Default parameters		
Random Forest with ADASYN		
Random Forest with SMOTE		

Neural Networks Default parameters		
Neural Networks with ADASYN		
Neural Networks with SMOTE		

5.4. Summary

After looking at the suitability of algorithms to build the optimal model with high AUC/Accuracy, for Malware Detection, it can be concluded XGBoost with the original dataset and with a class-balanced dataset using SMOTE gives us the best results. Random Forest is next best, while Neural Networks with 8 dense layers came next. The XGBoost default parameters were sufficient to arrive at a very good accuracy for malware detection which is a binary classification problem. Usage of GridSearchCV has resulted in insignificant improvement.

For Network Traffic classification, which is a multiclass classification problem, has achieved some good F1-Score only for two out of seven classes while decently good precision has been achieved for five out of seven classes. For calculations, OVR(one-vs-rest) has been adopted. The accuracy for the other classes needs further improvement. This might require more feature engineering which is not covered as part of this paper. XGBoost with just default parameters without any class balancing performed better than all the other models.

Explainable AI methods LIME and SHAP were used to generate explanations for all the three models that were built using XGBoost, Random Forest, and Neural Networks. These explanations for the models built without class balancing and with SMOTE-based datasets are mostly similar while the explanations for models based on ADASYN are not so good in line with the decreased accuracy of these models.

On a whole, XGBoost with default parameters without any class balancing has performed better on all three datasets. Class balancing did not bring any additional improvement in the model metrics. Further Feature engineering may be needed to improve the metrics of the XGBoost model for Network Traffic classification. SHAP explanations provided better insights while the explanation for a given small set of observations.

6. CONCLUSIONS AND RECOMMENDATIONS

6.1. Introduction

In this paper, various possible options that are available to classify network traffic have been reviewed. This included the various formats of the dataset in which the network packet data is captured from the TCP traffic. For the scope of this paper, we have considered 3 different datasets which are NetML, CICIDS for Malware classifications, and Non-VPN dataset for network traffic type classification. All three different datasets are in JSON format extracted using an accelerated feature extraction library. It computes the features of a network flow based on the first 200 packets in both directions. For the scope of this paper, only the top-level classifications have been considered using XGBoost, Random Forest, and Neural Networks. Explainable AI methods LIME and SHAP have been applied on models and a comparison that better explains these models.

6.2. Discussion and Conclusion

6.2.1. Models

As its first objective, many types of models were considered to find the most appropriate algorithms for modeling appropriate for this use case. XGBoost, Random Forest and Neural Networks algorithms have been shortlisted for building the models for classification. The performance of these models is compared against the AUC and Accuracy. They were ranked in the following order XGBoost, RandomForest, and Neural Networks as a whole. XGBoost has achieved the best results by using the default parameters. Usage of GridSearchCV did not lead to any significant improvement in the model. Usage of a very low learning rate has led to a very negligible improvement in the ROC value. The Neural Network model was built with 8 dense hidden layers for both binary classification and multiclass classification problems. The accuracy / AUC achieved with neural networks is lesser than both the XGBoost and RandomForest. Features of All three datasets that are under consideration are not normally distributed and tree-based algorithms like XGBoost has have handled it better than the Neural Networks model for these classification tasks.

For Malware detection at the top level which is a binary classification problem to arrive at better accuracy, it is better to use an optimal probability cut-off based on the sensitivity, specificity & accuracy. Usage of ADAYN and SMOTE for class balancing the data has been tested. Given the nature of the dataset which is not normally distributed usage of ADASYN resulted in reduced

AUC/Accuracy as ADASYN uses weighted distribution for generating the random samples for class balancing. Results using SMOTE resulted in almost the same AUC/Accuracy when compared to the original dataset, though mostly using SMOTE resulted in a very small or insignificant improvement. The conclusion from this is that for datasets where observations are extracted from the summary of network flows, which are inherently not normally distributed, usage of SMOTE may result in a minor improvement in AUC/Accuracy which might be needed when the goal is to achieve high accuracy.

For Network Traffic classification, which is a multiclass classification problem, has achieved some good F1-Score only for two out of seven classes, as there is a significant overlap of top significant features values for different classes of the model. Where high sensitivity and specificity are needed this model can be applied to identify these two classes only. For some use-cases where good precision is sufficient, this can be applied to five out of seven classes. For metrics calculations, OVR(one-vs-rest) has been adopted. The F1 metrics for the other five classes need further improvement. This might require more feature engineering which is not covered as part of this paper.

6.2.2. Explainable AI methods

The second objective of this whole study is to explain all the three models considered for our study primarily using LIME and SHAP Explainable AI methods. The Explanations provided by both these explainable AI methods are different though there is a small overlap. The feature scoring from these methods not only differs in the top features identified but also in the order of priority. LIME generates faster explanations and SHAP explanations are based on Shapley values and it takes a significantly longer time to generate explanations. At the outset, it looked that SHAP explanations are more realistic and more reliable when compared to LIME explanations.

Comparing the explanations of models based on default datasets with SMOTE and ADASYN based models, the explanations for ADASYN based models are significantly different. The explanations for models based on default datasets and SMOTE-based models are broadly similar with about 80% overlap approximately, through the scoring of the top features differ to some extent. Though it was difficult to conclude with concrete evidence that SMOTE-based models are better, they are not any lesser than the models that are based on the dataset without any class balancing. For all practical reasons, for Malware detection, which is a binary classification problem, it can be concluded that XGBoost models that are built by class balancing using SMOTE or without any class balancing offered better explanations using SHAP. For Network Traffic classification, which is a multi-class

classification, XGBoost works better with just the default/original dataset without any class balancing and SHAP offers better model explanations.

6.3. Contribution to Knowledge

The paper contributes to the body of knowledge in the following ways:

- While building the models for classifying the data SMOTE is a better class balancing technique for Malware detection, given that the observations are not normally distributed. Despite using SMOTE the incremental AUC/Accuracy achieved is very small. In the case of Network Traffic Classification, which is a multi-class classification problem, usage of class balancing dragged down the accuracy of the model.
- XGBoost with just the default parameters can achieve close to the best accuracy when working with datasets where each observation is a set of features from the network flows.
- This is one of the very few papers to explore the application of Explainable AI on the Network Traffic Data Classification. This helps to identify the most important features that lead us to identify malware in Network Traffic and classification Network Traffic types. These features are important when implementing various types of use cases and notifications for network applications.
- When compared between LIME and SHAP, SHAP provides betters explanations.

6.4. Future Recommendations

Below are a few recommendations for further study.

- Check if the accuracy of the Neural Networks model can be improved with further model engineering. Check if converting the observations into images and then applying Convolutional Neural Networks can improve the accuracy of the model.
- Explore the application of IntegratedGradients and DeepLIFT after improving the accuracy of the model.
- Check if the multi-class classification of Network traffic can be improved using better feature engineering.

References

Ahmad, I., Basheri, M., Iqbal, M.J. and Rahim, A., (2018) Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection. *IEEE Access*, 6, pp.33789–33795.

Ancona, M., Ceolini, E., Öztireli, C. and Gross, M., (2017) Towards better understanding of gradient-based attribution methods for deep neural networks. *arXiv*, pp.1–16.

Anderson, B., Paul, S. and McGrew, D., (2018) Deciphering malware's use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques*, 143, pp.195–211.

Apley, D.W. and Zhu, J., (2020) Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 824, pp.1059–1086.

Barut, O., (n.d.) NetML : A Challenge for Network Traffic Analytics.

Calleja, A., Martín, A., Menéndez, H.D., Tapiador, J. and Clark, D., (2018) Picking on the family: Disrupting android malware triage by forcing misclassification. *Expert Systems with Applications*, [online] 95, pp.113–126. Available at: https://doi.org/10.1016/j.eswa.2017.11.032.

Dhaliwal, S.S., Nahid, A. Al and Abbas, R., (2018) Effective intrusion detection system using XGBoost. *Information (Switzerland)*, 97.

Doshi-Velez, F. and Kim, B., (2017) Towards A Rigorous Science of Interpretable Machine Learning. [online] Ml, pp.1–13. Available at: http://arxiv.org/abs/1702.08608.

Freund, Y. and Schapire, R.E., (1995) A decision-theoretic generalization of on-line learning and an application to boosting. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9041, pp.23–37.

Hatwell, J., Gaber, M.M. and Azad, R.M.A., (2020) *CHIRPS: Explaining random forest classification*. [online] *Artificial Intelligence Review*, Springer Netherlands. Available at: https://doi.org/10.1007/s10462-020-09833-6.

He, H., Bai, Y., Garcia, E., & Li, S., (2008) ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In IEEE International Joint Conference on Neural Networks, 2008. *IJCNN 2008.(IEEE World Congress on Computational Intelligence) (pp. 1322– 1328)*, 3, pp.1322– 1328.

Inouye, D.I., Leqi, L., Kim, J.S., Aragam, B. and Ravikumar, P., (2019) Automated Dependence Plots. [online] 124. Available at: http://arxiv.org/abs/1912.01108.

Kovács, B., Tinya, F., Németh, C. and Ódor, P., (2020) SMOTE: Synthetic Minority Over-sampling Technique Nitesh. *Ecological Applications*, 302, pp.321–357.

Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R.J. and Wasserman, L., (2018) Distribution-Free Predictive Inference for Regression. *Journal of the American Statistical Association*, 113523, pp.1094–1111.

Lundberg, S.M. and Lee, S.I., (2017) A unified approach to interpreting model predictions. *arXiv*, Section 2, pp.1–10.

Melis, M., Maiorca, D., Biggio, B., Giacinto, G. and Roli, F., (2018) Explaining black-box android malware detection. *European Signal Processing Conference*, 2018-Septe, pp.524–528.

Miguel Marín Freire, G. and Sprechmann Google DeepMind Pere Barlet-Ros UPC Barcelona Tech Director Académico, P., (2019) Deep Learning for the Analysis of Network Traffic Measurements. [online] Available at: http://iie.fing.edu.uy/.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W. and Müller, K.R., (2017) Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, [online] 65May 2016, pp.211–222. Available at: http://dx.doi.org/10.1016/j.patcog.2016.11.008.

Moreo, A., Esuli, A. and Sebastiani, F., (2016) Distributional random oversampling for imbalanced text classification. *SIGIR 2016 - Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.805–808.

Morichetta, A., Casas, P. and Mellia, M., (2019) Explain-IT: Towards explainable AI for unsupervised network traffic analysis. *Big-DAMA 2019 - Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks, Part of CoNEXT 2019*, pp.22–28.

Pavlov, Y.L., (2019) Random forests. *Random Forests*, pp.1–122.

Ribeiro, M.T., Singh, S. and Guestrin, C., (2016) 'Why should i trust you?' Explaining the predictions of any classifier. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-Augu, pp.1135–1144.

Robnik-Šikonja, M. and Bohanec, M., (2018) *Perturbation-Based Explanations of Prediction Models*. [online] Springer International Publishing. Available at: http://dx.doi.org/10.1007/978-3-319-90403-0_9.

Shrikumar, A., Greenside, P. and Kundaje, A., (2017) Learning important features through propagating activation differences. *34th International Conference on Machine Learning, ICML 2017*, 7, pp.4844–4866.

Sundararajan, M. and Taly, A., (2018) A note about: Local explanation methods for deep neural networks lack sensitivity to parameter values. *arXiv*, pp.1–10.

Sundararajan, M., Taly, A. and Yan, Q., (2017) Axiomatic attribution for deep networks. *arXiv*.

De Veaux, R.D. and Ungar, L.H., (1994) Multicollinearity: A tale of two nonparametric regressions. pp.393–402.

Visani, G., Bagli, E., Chesani, F., Poluzzi, A. and Capuzzo, D., (2020) Statistical stability indices for LIME: obtaining reliable explanations for Machine Learning models. *arXiv*.

Vuppala, S.K., Behera, M., Jack, H. and Bussa, N., (2020) Explainable Deep Learning Methods for Medical Imaging Applications. pp.334–339.

Online References:

Shum, (2020), Explaining AI, [online] Available at: https://a16z.com/2020/01/16/biases-and-black-boxes-a-call-for-ai-transparency/ [Accessed 10 Mar 2021].

Molnar, (2021), Interpretable Machine Learning., [online] Available at: https://christophm.github.io/interpretable-ml-book/ [Accessed 10 Mar 2021].

Alberto Fernández, (2018) , Learning from Imbalanced Data Sets, [online] Available at:: https://link.springer.com/book/10.1007%2F978-3-319-98074-4 [Accessed 10 Mar 2021]

Zehori, (2020) Learning from Imbalanced Datasets , [online] Available at: https://towardsdatascience.com/learning-from-imbalanced-datasets-b601a1f1e154 [Accessed 10 Mar 2021]

Jaadi, 2019, [online] Available at: https://builtin.com/data-science/when-and-why-standardize-your-data [Accessed 10 Mar 2021]

Stöttner, 2019, [online] Available at: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d [Accessed 10 Mar 2021]

Freund and Schapire, 1995, [online] Available at: https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5 [Accessed 10 Mar 2021]

APPENDIX A: Research Proposal

Research Proposal

Explainable AI for Network Traffic Analytics

Srinivas Pokuri (Student ID: 939948)

Liverpool John Moore's University, Liverpool, UK

Research Supervisor : Snehansu Sekhar Sahu

Abstract

Classifying networks traffic is important pre-requisite for Network applications. Though extensive research has been done in this area , the progress has a been moderate owing to the availability of real representative datasets during various security concerns. (Barut, n.d.) makes us available a new novel dataset and a curated datasets of Network flows with all features aggregated for each of various classes of network flows. There are multiple sets of datasets which cater to malware detection at multiple levels and non-vpn network traffic details specifying the kind of network traffic it is. Here each observation in the dataset is a summary of first 200 network packets from each of the network session, where each network session is from one pcap file. In this Thesis we try to provide explanations to these Deep learning models and compare these explanations to the Shallow models like RandomForest/XGBoost, SVM. Model agnostic explainable AI methods LIME, SHAP will be applied to understand which features lead to detecting a malware/network classification. Additionally, for Deep learning model Integrated Gradients and DeepLIFT also will be applied to see if these methods helps us to draw additional inferences related to Network Traffic Analysis over the ones provided by LIME and SHAP. Standard Python libraries like Keras, Sklearn will be used for building the models and libraries like ELI5, LIME, SHAP, DeepLIFT will be used for explaining the models. The Explainability study will be confined to high level malware detection and network classification owing to the huge number scenarios that will be possible. The contribution of this paper is identifying the best model for malware detection and network classification and a comparative study to explain the models with multiple explainable AI methods.

Table of Contents

Table of Figures

Table of Tables

1. Background

Classifying networks traffic is important pre-requisite for Network applications. Though extensive research has been done in this area , the progress has a been moderate owing to the availability of real representative datasets during various security concerns. (Barut, n.d.) makes available a new novel dataset and a curated datasets of Network flows with all features aggregated for each of various classes of network flows.

Several advances have been made in different domains using Neural networks, Convolutional neural works that have achieved pathbreaking accuracies in multi-class classifications, where the models automatically identify the features and learns from those features. The main drawback going with Deep neural networks is that there pre-dominantly backbox models and is hard for humans how the model arrives at those decisions. Understanding how these decisions are made is important to understand to build future proof systems that are resilient, reliable and adaptive.

In this Thesis we try to provide explanations to these Deep learning models and compare these explanations to the Shallow models and see how well these explanations compare and if these makes sense to humans. From these we try to find a balance for achieving improved accuracy of the model while still it being reasonable explainable.

2. Related Research

Network Traffic Analysis

Use of TLS causes a challenge in network threat detection as traditional pattern matching techniques can no longer be applied here. In (Anderson et al., 2018), shows that the TLS features can be used to detect and understand malware communication. The paper extracts the TLS network flow features into a 5-feature tuple like sequence of packet lengths, interval arrival times, byte encryption and unencrypted TLS handshake information. SVM and L1-Logistic regression have successfully classified 18 classes of malwares with 90% accuracy.

Advances in Machine learning has provided a way to leverage it for Network Traffic Analysis (Barut, n.d.). The important methods that have been used in this paper are the SVM , Random forest and Extreme learning are neural nets with single hidden layer. This paper revels that SVM

in accuracy with smaller datasets , while Random Forest is slow in prediction in real-time. ELMs perform better in speed and accuracy (Ahmad et al., 2018).

Off late, Deep learning is being applied in multiple areas and its application to Network Traffic Analysis is no exception. Deep learning for Analysis of Network Traffic Measurements (Miguel Marín Freire and Sprechmann Google DeepMind Pere Barlet-Ros UPC Barcelona Tech Director Académico, 2019), showcases an exhaustive research that has been carried out in this area. This papers results show that Deep learning achieves great accuracy in Malware detection and Malware classification. This also focuses exhaustively on using two variant datasets, one based on the Raw packets, where each instance represents one packet and using Raw flows where each flow is made up of multiple raw packets assembled as a Tensor.

In most of the classification problem, the general issue is the minority classes will much smaller in number and are subsumed by the majority classes. It is important to get the prediction of these minority classes right , as in most cases the use cases will be get the predictions around the minority predictions right. SMOTE (Synthetic minority Over-Sampling Technique) along with majority class under sampling has given better results. In this approach over sampling is generated synthetic samples are generated by taking the difference between the feature vector under consideration and its nearest neighbor and multiplying this difference between 0 and 1 and adding it feature vector under consideration. SMOTE forces focused learning and introduces a bias towards the minority class (Kovács et al., 2020).

2.1. Explainable AI

2.1.1. Models classification (Black box/ White box)

From the point of interpretability machine learning models can be a white box or black box.

Whitebox models: Whitebox models are transparent and logic on how they work is easily understood from their linearity , monotonicity where the relation between the input and output features can be easily understood by the humans. Linear regression, logistic regression and decision trees are some examples where the model predictions can be understood by humans easily.

Blackbox Models: In case of Black box models the internal working of the model is hard to understand for the humans and they functional relationships between the input features and models output. Deep Neural networks, Gradient boosting, Random forest, Support Vector machines are examples of Black-box models.

In general, the black-box models are help in generating more accurate predictions as they use more complex logic to arrive at the results and this significantly reduces their interpretability. It is not for the explainable AI methods a balance must be stuck between the model interpretability and the model accuracy.

2.1.2. Local vs Global explanations

Explanation of AI can either be Local or global. For local explanations, the individual relevance of each of the feature in effecting the output (Melis et al., 2018). To globally explain the model a better way is to take a sample set and then average the relevance of the features (Melis et al., 2018). While this approach works for differential models, for non-differential models like random forest we take a help of surrogate models ,where predictions from a non-differential algorithm are used to train non-linear models like SVM (Melis et al., 2018). It is important for global explanation of the model to pick instances/observations such that components/features are not redundant (Ribeiro et al., 2016).

2.1.3. Model agnostic methods for Explainable AI / Attribution and Perturbation methods

Perturbation and Attribution explainable AI methods are slightly related in a way. Perturbation methods work by changing value of inputs in a controlled way to see how it is impacting the outputs. Attribution methods work by masking the inputs and see if that has any impact on the outputs.

With the success of black-box models like Deep Neural Networks (DNN) , Conventional Neural Networks (CNN), Gradient Boosting, Random Forest, Support Vector Machines (SVM) there is need for explainable AI methods for black-box models. Various explainable AI methods to explain the black-box models are: Partial Dependent Plot(PDP), Feature importance, Surrogate Model, LIME, SHAP, Anchor (Harry, 2020).

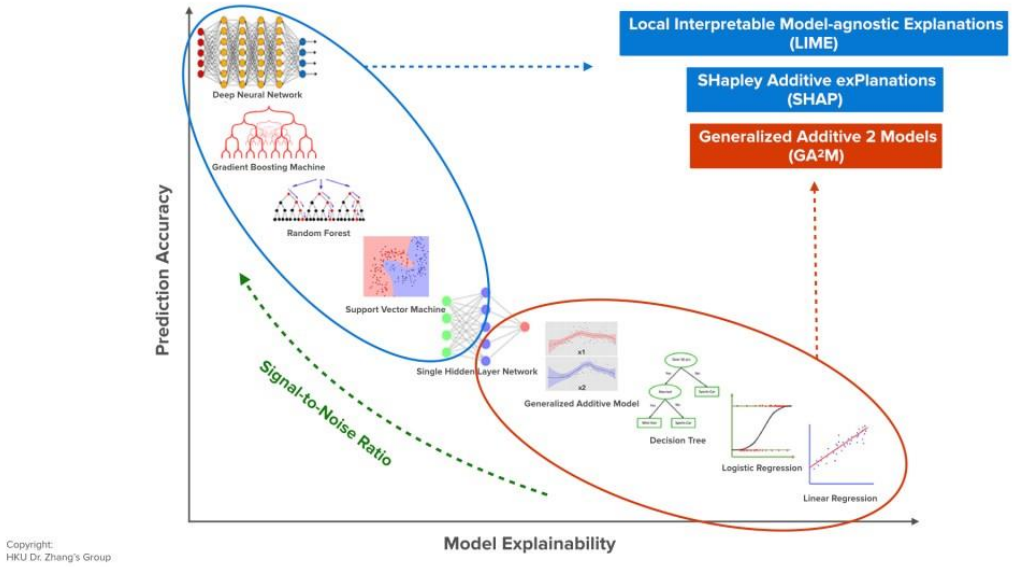


Figure 1: ML Models vs Explainable Models *(Shum, 2020)*

- **Partial Dependent Plots (PDP):** PDP plots average model output changes when the feature or a set of features are marginalized over other features. Which meaning the a few select feature values while keeping the other features unchanged. Since these plots average one response over other to give a global view impact of features on model response a variation of this method called Automated Dependent Plots (ADP) (Inouye et al., 2019).
- **Individual Conditional Expectation(ICE):** This shows one line per instance when a feature changes. A PDP is an average of ICE plot. ICE curves displays one curve meaningfully and so feature correlation is not captured (Molnar, 2021).
- **Accumulated Local Effects:** These are unbiased and faster compared to PDP and can work well even when the features are correlated. This can provide better visualization plots (Apley and Zhu, 2020).
- **Permutation Feature Importance:** It permutates feature values and change in the prediction error is measured. This captures the relationship between input features and model's targets (Molnar, 2021).

- **Global Surrogate:** For non-differential models like random forest we take a help of surrogate models ,where predictions from a non-differential algorithm are used to train non-linear models like SVM (Melis et al., 2018)
- **Local Interpretable Model-agnostic Explanations (LIME):** This method explains predictions of any classifiers by learning the model locally around individual model's prediction. This can be applied to all types of models including classification of images. This can be done by randomly picking a few instances to explain the model outputs (Ribeiro et al., 2016).
- **Anchors:** This method uses a perturbation approach using if-then-else rules when inputs are complex within neighborhood of an instance.
- **Shapley Values:** Tells how to fairly distribute prediction amount the features. This is derived from game theory. This is computationally expensive (Molnar, 2021) .
- **SHapley Additive exPlanations (SHAP)**: This is derived from about 6 other existing explainable methods such as LIME, LOCO, Shapley values (Molnar, 2021) . It addresses important features of explainability like Local accuracy, Missingness and Consistency across different observations(Lundberg and Lee, 2017).
- **Leave one Covariate out (LOCO):** One variable is removed from inputs and the variable that has largest impact on the model output is the most important one. This deteriorates in accuracy when the when complex non-linear models exists in accuracy (Lei et al., 2018).

2.1.4. Model specific Explainable AI methods - for Deep Neural Networks

Most of the model specific explainable methods are to explain the Deep convolutional networks and these can be categorized as below.

- **Backpropagation-Based Methods**:
  - **Gradient method**: $\frac{\partial S}{\partial x}$ ie., quantifies for each unit change in input dimension, what would the change in prediction $S(x)$ (Sundararajan and Taly, 2018)**.**

- **Integrated gradients:** (x-$\bar{x}$) $*$ $\int_0^1 \frac{\partial S}{\partial x}(\bar{x} + \mu(x - \overline{\boldsymbol{x}})/\partial$x , where $\bar{\boldsymbol{x}}$ represents baseline input that represents absence of feature in original input x (Sundararajan and Taly, 2018).
- **Layer-wise Relevance Propagation (LRP):** The algorithm starts at output layer proceeds layer by layer backward and redistributes the relevance of the output across all layers of neural network till the input layer (Ancona et al., 2017).
- **DeepLIFT:** Each feature is assigned an attribution corresponding to relative effect of the feature on the network compared to activation at some reference input. The baseline can be chosen as zero (Ancona et al., 2017).
- **Saliency Maps:** Constructs attributions by taking the absolute value of the partial derivative of the target output with respect to the input features (Ancona et al., 2017)**.**
- **Deep Taylor Decomposition:** Each neuron is calculated as a function that is decomposed on its input variables. The decompositions of multiple neurons are then aggregated or propagated backwards (Montavon et al., 2017). This method produces sparser explanations, assumes no negative evidence in the input and produces only positive attribution maps (Ancona et al., 2017).
- The other methods like **Deconvolutional Network**, **Guided back propagation, Guided GradCAM, SmoothGrad, VarGrad** are primarily used for explaining models that classify the images. Since in our case we are do not dealing with the input images , these methods are not considered for further study.

- **Forward propagation methods:**
  - Activation Maps: All called channels are widely used with Convolutional Neural networks. Here the features are activated in the forward pass of the network (Vuppala et al., 2020).
  - All model agnostic Perturbation / Attribution methods work here as well . In contrast to DL backpropagation methods, forward propagation transfers the changes in input to the scores that are computed at the very last layers (Salehi, 2019).

2.1.5. Model specific Explainable AI methods - for RandomForest

Explainable AI methods that specifically address the random forest visualizations exploit the observations from the validation set to identify the important features and similarity between the outputs can be measured (Robnik-Šikonja and Bohanec, 2018). Recently a explainable AI method CRIPS has been proposed that can extracts the decision path in each of the tree that contributes to majority classification and then uses frequent pattern mining to identify the most commonly occurring split conditions (Hatwell et al., 2020).

3. Research Questions

The specific questions that this this thesis tries to find answer are:

- Is deep learning the best method to classify the network traffic with high accuracy when using the aggregated NetML flows ?
- Can Explainable AI methods convincingly explain the above Networks Traffic classification by Neural networks ?

4. Aim and Objectives

The aim of this thesis to be able to better explain the network traffic analysis for malware detection and network traffic classification.

4.1. Aim

Find an explainable AI method that best suits to explain the model/models that are built to classify the network traffic for malware detection and classifying the general network traffic.

4.2. Objective

Network Traffic Analytics for Malware detection and traffic classification is generally a tough problem to solve considering that there are very few real datasets available for model training. The network flows dataset is that is made available for the NetML challenge will be leveraged for the

purpose. This is made up of a novel NetML and curated CICIDS2017 dataset for malware detection and non-vpn2016 dataset for network classification(Barut, n.d.).

In this thesis we try to classify the network flows for malware detection also classifying the non-vpn network traffic and explain the model using the explainable AI methods.

Network Traffic Classification:

1. Classify the Malwares top level (malware/benign) on NetML dataset.
2. Classify the Malware top level (malware/benign) on the CICIDS2017 curated dataset.
3. Classify the non-vpn network traffic at top level on the on the curated non-vpn2016 dataset.

For all the above three use cases, class imbalances should be handled. Classification will be done using various models suitable for Network traffic analysis like SVM , XGBoost/Random Forest and Deep learning and the appropriate model needed for the classification will be recommended.

Model explainability:

1. Explain the above models using various model agnostic explainable AI methods like LIME , SHAP etc.
2. In case of DL model, additional explain it using the Deep LIFT explainable Method, and verify this explanation is as good as the natively interpretable models like Random Forest/XGBoost.

5. Significance of Study

Compared to other areas, Explainable AI has not been applied so exhaustively in the area of Network Traffic Analysis. Many critical network applications classification network traffic but we should be able to understand why such a classification is being made (Melis et al., 2018; Morichetta et al., 2019). For e.g., being able to track and understand if and why there is any malware attack/alarm or tracing the network bandwidth bottlenecks because of a specific application running on the network.

In this thesis the planned contribution is three-fold:

- Use a relatively new/novel dataset and curated datasets made available by the Challenge for building a more accurate model for malware detection and traffic classification (Barut, n.d.).
- To increase the human confidence in the model using explainable AI methods.
- Since the black box models have when applied to computer security can be easily evaded by small changes in the content (Calleja et al., 2018). Identify potential vulnerabilities of linear/non-linear models against adversarial manipulations (Melis et al., 2018).

6. Scope of Study

The study is confined to the following:

- The dataset will be leveraged as is. It is assumed the dataset provided is curated.
- The scope of study will be confined to a maximum of three models SVM , Random Forest/XGBoost and Neural Networks based classification.
- While doing the SVM, XGBoost /RF modelling all provided features will be considered. No upfront manual selection of the features based on the domain knowledge will be considered.
- A maximum of four Explainable AI methods (LIME, SHAP, Integrated Gradients, DeepLift) will be considered to explain the explanations for the model.
- For each model , explain the local predictions for a few random observations and one global explanation i.e., ALE.
- The ML models and Explanation of the models will be confined to top-level malware detection and top-level non-vpn network classification owing to number of possible scenarios and time.

7. Research Methodology

7.1. Introduction

The dataset made available by for the challenge is Novel NetML dataset and has curated datasets from CICIDS2017 and non-vpn2016 datasets. The idea here is we try to do high-level malware detection using first two datasets. High-level Network traffic classification will be done on non-vpn2016 dataset. As stated the focus is achieving high test accuracy in malware and network classifications using multiple approaches SVM, RF and Deep Learning. Use explainable AI would be considered to explain the results from each of the model. This provides more insights how a particular observation will be how to avert the avoid the malware detections (Melis et al., 2018).

7.2. Dataset Description

As mentioned earlier, all three different datasets are in json format. Using accelerated feature extraction library . “It computes the features of a flow up to 200 packets in both directions. Given a raw traffic capture (pcap) file as input to the feature extraction tool, flow features are extracted in JSON format, and each flow sample is listed line by line in the output file. Given a raw traffic capture file as input to the feature extraction tool, flow features are extracted in JSON format, and each flow sample is listed line by line in the output file. Metadata features are extracted for each flow sample and TLS, DNS and HTTP features are extracted only if the flow sample contains packets for the given protocols.” (Barut, n.d.) . Note here in these datasets the source and destination IPs are masked and start and end time of packet capture in pcap is replaced with duration. All the datasets are split into 3 sets. 80% training set, 10% test set and another 10% as challenge/validation set.

These datasets are explained in brief below.

NetML dataset (For malware detection and classification) (Barut, n.d.):

Table 1: Data Set Fields Description

Metadata Features
sa: source address
da: destination address
pr: protocol (6 or 17)
src_port: source port
dst_port: destination port
bytes_out: total bytes out
num_pkts_out: total packets out
bytes_in: total bytes in
num_pkts_in: total packets in
time_start: time stamp of first packet
time_end: time stamp of last packet
intervals_ccnt[]: compact histogram of pkt arriving intervals
ack_psh_rst_syn_fin_cnt[]: histogram of tcp flag counting
hdr_distinct: number of distinct values of header lengths
hdr_ccnt[]: compact histogram of header lengths
pld_distinct: number of distinct values of payload length
pld_ccnt[]: compact histogram of payload lengths
hdr_mean: mean value of header lengths
hdr_bin_40: # of pkts with header lengths between 28 and 40
pld_bin_128: # of pkts whose payload lengths are below 128
pld_bin_inf: # of pkts whose payload lengths are above 1024
pld_max: max value of payload length
pld_mean: mean value of payload length
pld_medium: medium value of payload length
pld_var: variance value of payload length
rev_...: flow features of the reverse flow

TLS Features
tls_cnt: number of tls packets
tls_len[]: array of tls payload length
tls_cs_cnt: number of ciphersuits
tls_cs[]: array of ciphersuits value
tls_ext_cnt: number of tls extensions
tls_ext_types[]: array of tls extensions
tls_key_exchange_len: length of tls key exchange
tls_svr_...: tls features advertised by the server
DNS Features
dns_query_cnt: number of dns query records
dns_query_name_len[]: array of dns query name lengths
dns_query_name[]: array of dns query names
dns_query_type[]: array of dns query types
dns_query_class[]: array of dns query classes
dns_answer_cnt: number of dns answer records
dns_answer_ttl[]: array of dns answer âĂIJtime to liveâĂİ values
dns_answer_ip[]: array of dns answer ips
HTTP Features
http_method: http request method
http_uri: http request uri
http_host: http request hostname
http_code: http code
http_content_type: http content_type
http_content_len: http content length

For each of observations with the above features they classification at top level is either benign/malware and at fine grained level the categorization specifies the type of malware.

7.3. Data pre-processing

No specific additional data preprocessing is needed/ or identified so far the dataset is already curated, except for encoding of the categorical variables and applying standard scaling on the input variables.

7.4. Models

Below are three models that will be used. To start with all the features will be used while building the models.

- Balance the dataset using the SMOTE algorithm as needed.
- ML Models:
    - SVM: Find the optimal parameters that can give high accuracy

- Random Forest and/or XGBoost: Find optimal parameters that can give high accuracy.
- Deep Learning: Use Neural networks with atleast 4-8 hidden layers.

- Explanation Methods:
  - LIME : For local interpretable models for model agnostic methods
  - SHAP: Good at local explanations, more suitable for Tree base algorithms.
  - DeepLift : Decomposes the prediction of a neural network by backpropagating the contributions of all neurons in different layers in the network to every feature of the input.
  - Integrated gradients: where we
  - ELI5 : For local and global interpretable models

7.5. Model Evaluation

Model evaluation is based on multiple metrics like accuracy, confusion matrix, precision, recall, Area under curve (AUC) and Receiving Operating Curve(ROC). All these metrics are leveraged in evaluating various aspects of classification accuracy.

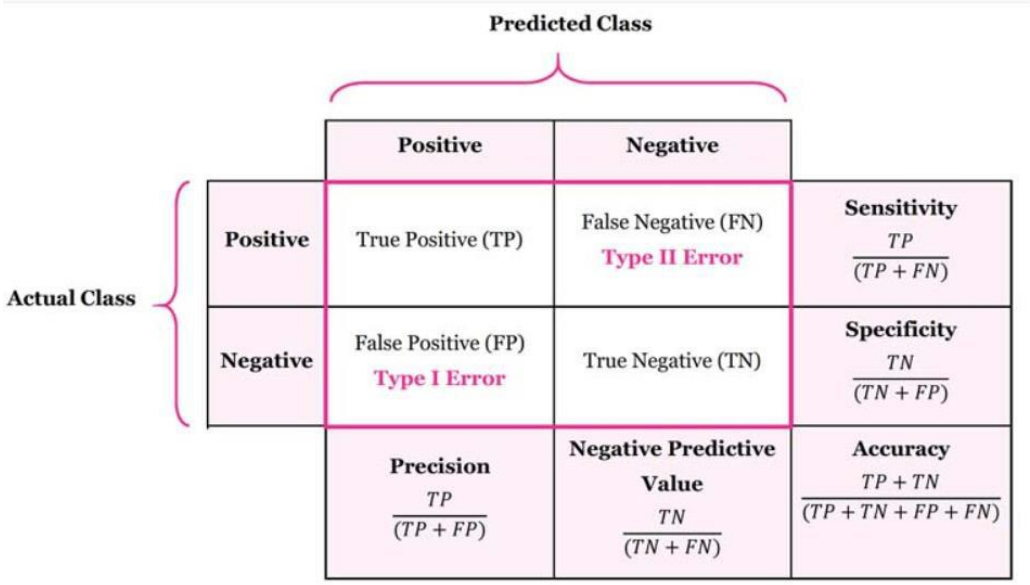


Figure 2: Confusion Matrix *(Science, 2019)*

- Accuracy: Ratio between the correct predictions and the total observations. This will be used for both malware detections/ classifications and network traffic classification,
- Confusion matrix: This a matrix representation of actual vs predicted in case of binary classification. This will be applied to high level classification of malware detection based on the TCP flow.
- ROC (Receiving Operating Curve) shows the trade-off between True positive rate and False positive rate. AUC (Area under the ROC Curve) gives the overall performance of a binary classifier.
- Precision & Recall and Sensitivity & Specificity are two widely used binary classification metrics, based on the use case.
- F1-Score is harmonic mean that combines the recall and precision into one single metric. This score has high importance when high precision and high recall values are needed at the same time. The formula for F1 Score is $\frac{2*(Precision*recall)}{(Precision+recall)}$

7.6. Explaining AI

7.6.1. Model agnostic explainable AI methods

Below explainable AI will be considered for the scope of this work, based on the wide use and their applicability to Network Traffic Analysis use cases.

- **Local Interpretable Model-agnostic Explanations (LIME)**: This method explains predictions of any classifiers by learning the model locally around individual model's prediction. This can be applied to all types of models including classification of images. This can be done by randomly picking a few instances to explain the model outputs (Ribeiro et al., 2016). LIME is a perturbation method which checks the significance of a feature in the outcome that has been predicted. The explanations change based on the inputs and cannot be considered for general explanation of the model.
- **SHapley Additive exPlanations (SHAP)** : This is derived from about 6 other existing explainable methods such as LIME, LOCO, Shapley values. It addresses important features of explainability like Local accuracy, Missingness and Consistency across different observations(Lundberg and Lee, 2017).

7.6.2. Explainable AI methods for deep learning models

Below Attribution methods for Neural networks will be considered as we have structured data and we do not deal with the images in our case study.

- **Integrated Gradients**: This is recommended for differential models and neural networks where there are large number of input features , as it can perform faster. This computer the partial derivative of the model output against corresponding input feature. Integrated Gradients calculates the mean gradient by varying the input linearly from $\bar{x}$ to x. The baseline can be chosen as zero. The attributions add up to the model output minus the model output evaluated at chosen baseline (Ancona et al., 2017).

- **DeepLIFT**: Each feature is assigned an attribution corresponding to relative effect of the feature on the network compared to activation at some reference input. The baseline can be chosen as zero (Ancona et al., 2017).

7.7. Comparative study

A Comparative study will be performed on the model metrics, for each of the model the Explainable AI methods would be applied to verify which of the methods explain the results well. There are no definite metrics for explainability the evaluation can be at application, human and functional level (Doshi-Velez and Kim, 2017) .

8. Required Resources

Below are software and hardware requirements.

8.1. Software Requirements

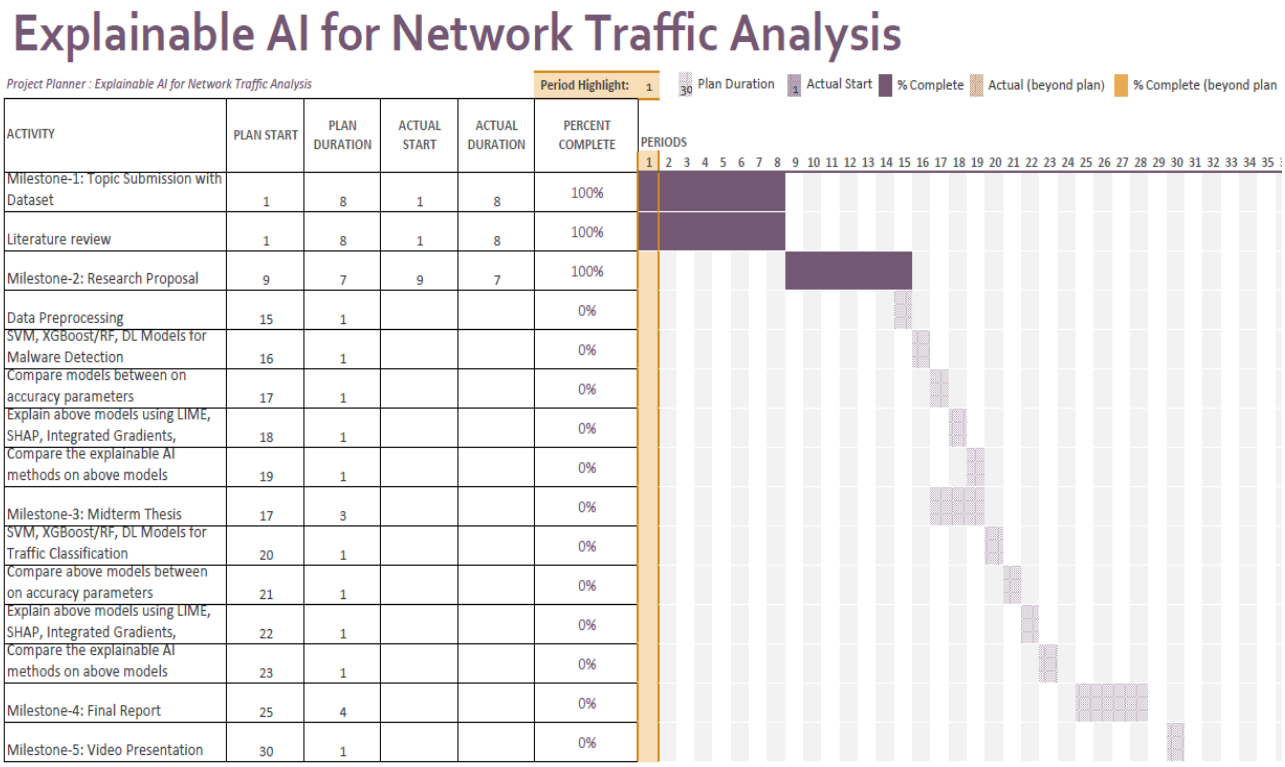
- Mendeley – For literature review and citations.
- Anaconda Navigator
- Jupyter notebook
- Python
- Python Libraries for modelling:
  - Pandas
  - Numpy
  - Matplotlib and Seaborn
  - Scikit-learn
  - Keras
  - Tensorflow
- Python libraries for interpretability methods:
  - ELI5
  - LIME
  - SHAP
  - DeepLIFT

8.2. Hardware Requirements

- Windows 10, 64 bit

- 16 GB RAM
- Intel CORE i5, 8th Gen
- GPUs.

9. Research Plan



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Milestone-1: Topic Submission with Dataset	1	8	1	8	100%
Literature review	1	8	1	8	100%
Milestone-2: Research Proposal	9	7	9	7	100%
Data Preprocessing	15	1			0%
SVM, XGBoost/RF, DL Models for Malware Detection	16	1			0%
Compare models between on accuracy parameters	17	1			0%
Explain above models using LIME, SHAP, Integrated Gradients,	18	1			0%
Compare the explainable AI methods on above models	19	1			0%
Milestone-3: Midterm Thesis	17	3			0%
SVM, XGBoost/RF, DL Models for Traffic Classification	20	1			0%
Compare above models between on accuracy parameters	21	1			0%
Explain above models using LIME, SHAP, Integrated Gradients,	22	1			0%
Compare the explainable AI methods on above models	23	1			0%
Milestone-4: Final Report	25	4			0%
Milestone-5: Video Presentation	30	1			0%

Figure 3 Research Plan

References

Ahmad, I., Basheri, M., Iqbal, M.J. and Rahim, A., (2018) Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection. *IEEE Access*, 6, pp.33789–33795.

Ancona, M., Ceolini, E., Öztireli, C. and Gross, M., (2017) Towards better understanding of gradient-based attribution methods for deep neural networks. *arXiv*, pp.1–16.

Anderson, B., Paul, S. and McGrew, D., (2018) Deciphering malware's use of TLS (without decryption). *Journal of Computer Virology and Hacking Techniques*, 143, pp.195–211.

Apley, D.W. and Zhu, J., (2020) Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 824, pp.1059–1086.

Barut, O., (n.d.) NetML : A Challenge for Network Traffic Analytics.

Calleja, A., Martín, A., Menéndez, H.D., Tapiador, J. and Clark, D., (2018) Picking on the family: Disrupting android malware triage by forcing misclassification. *Expert Systems with Applications*, [online] 95, pp.113–126. Available at: https://doi.org/10.1016/j.eswa.2017.11.032.

Dhaliwal, S.S., Nahid, A. Al and Abbas, R., (2018) Effective intrusion detection system using XGBoost. *Information (Switzerland)*, 97.

Doshi-Velez, F. and Kim, B., (2017) Towards A Rigorous Science of Interpretable Machine Learning. [online] Ml, pp.1–13. Available at: http://arxiv.org/abs/1702.08608.

Freund, Y. and Schapire, R.E., (1995) A decision-theoretic generalization of on-line learning and an application to boosting. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9041, pp.23–37.

Hatwell, J., Gaber, M.M. and Azad, R.M.A., (2020) *CHIRPS: Explaining random forest classification*. [online] *Artificial Intelligence Review*, Springer Netherlands. Available at: https://doi.org/10.1007/s10462-020-09833-6.

He, H., Bai, Y., Garcia, E., & Li, S., (2008) ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In IEEE International Joint Conference on Neural Networks, 2008. *IJCNN 2008.(IEEE World Congress on Computational Intelligence) (pp. 1322– 1328)*, 3, pp.1322– 1328.

Inouye, D.I., Leqi, L., Kim, J.S., Aragam, B. and Ravikumar, P., (2019) Automated Dependence Plots. [online] 124. Available at: http://arxiv.org/abs/1912.01108.

Kovács, B., Tinya, F., Németh, C. and Ódor, P., (2020) SMOTE: Synthetic Minority Over-sampling Technique Nitesh. *Ecological Applications*, 302, pp.321–357.

Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R.J. and Wasserman, L., (2018) Distribution-Free Predictive Inference for Regression. *Journal of the American Statistical Association*, 113523, pp.1094–1111.

Lundberg, S.M. and Lee, S.I., (2017) A unified approach to interpreting model predictions. *arXiv*, Section 2, pp.1–10.

Melis, M., Maiorca, D., Biggio, B., Giacinto, G. and Roli, F., (2018) Explaining black-box android malware detection. *European Signal Processing Conference*, 2018-Septe, pp.524–528.

Miguel Marín Freire, G. and Sprechmann Google DeepMind Pere Barlet-Ros UPC Barcelona Tech Director Académico, P., (2019) Deep Learning for the Analysis of Network Traffic Measurements. [online] Available at: http://iie.fing.edu.uy/.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W. and Müller, K.R., (2017) Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, [online] 65May 2016, pp.211–222. Available at: http://dx.doi.org/10.1016/j.patcog.2016.11.008.

Moreo, A., Esuli, A. and Sebastiani, F., (2016) Distributional random oversampling for imbalanced text classification. *SIGIR 2016 - Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.805–808.

Morichetta, A., Casas, P. and Mellia, M., (2019) Explain-IT: Towards explainable AI for unsupervised network traffic analysis. *Big-DAMA 2019 - Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks, Part of CoNEXT 2019*, pp.22–28.

Pavlov, Y.L., (2019) Random forests. *Random Forests*, pp.1–122.

Ribeiro, M.T., Singh, S. and Guestrin, C., (2016) 'Why should i trust you?' Explaining the predictions of any classifier. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-Augu, pp.1135–1144.

Robnik-Šikonja, M. and Bohanec, M., (2018) *Perturbation-Based Explanations of Prediction Models*. [online] Springer International Publishing. Available at: http://dx.doi.org/10.1007/978-3-319-90403-0_9.

Shrikumar, A., Greenside, P. and Kundaje, A., (2017) Learning important features through propagating activation differences. *34th International Conference on Machine Learning, ICML 2017*, 7, pp.4844–4866.

Sundararajan, M. and Taly, A., (2018) A note about: Local explanation methods for deep neural networks lack sensitivity to parameter values. *arXiv*, pp.1–10.

Sundararajan, M., Taly, A. and Yan, Q., (2017) Axiomatic attribution for deep networks. *arXiv*.

De Veaux, R.D. and Ungar, L.H., (1994) Multicollinearity: A tale of two nonparametric regressions. pp.393–402.

Visani, G., Bagli, E., Chesani, F., Poluzzi, A. and Capuzzo, D., (2020) Statistical stability indices for LIME: obtaining reliable explanations for Machine Learning models. *arXiv*.

Vuppala, S.K., Behera, M., Jack, H. and Bussa, N., (2020) Explainable Deep Learning Methods for Medical Imaging Applications. pp.334–339.

Research Proposal

Explainable AI for Network Traffic Analytics

Srinivas Pokuri (Student ID: 939948)

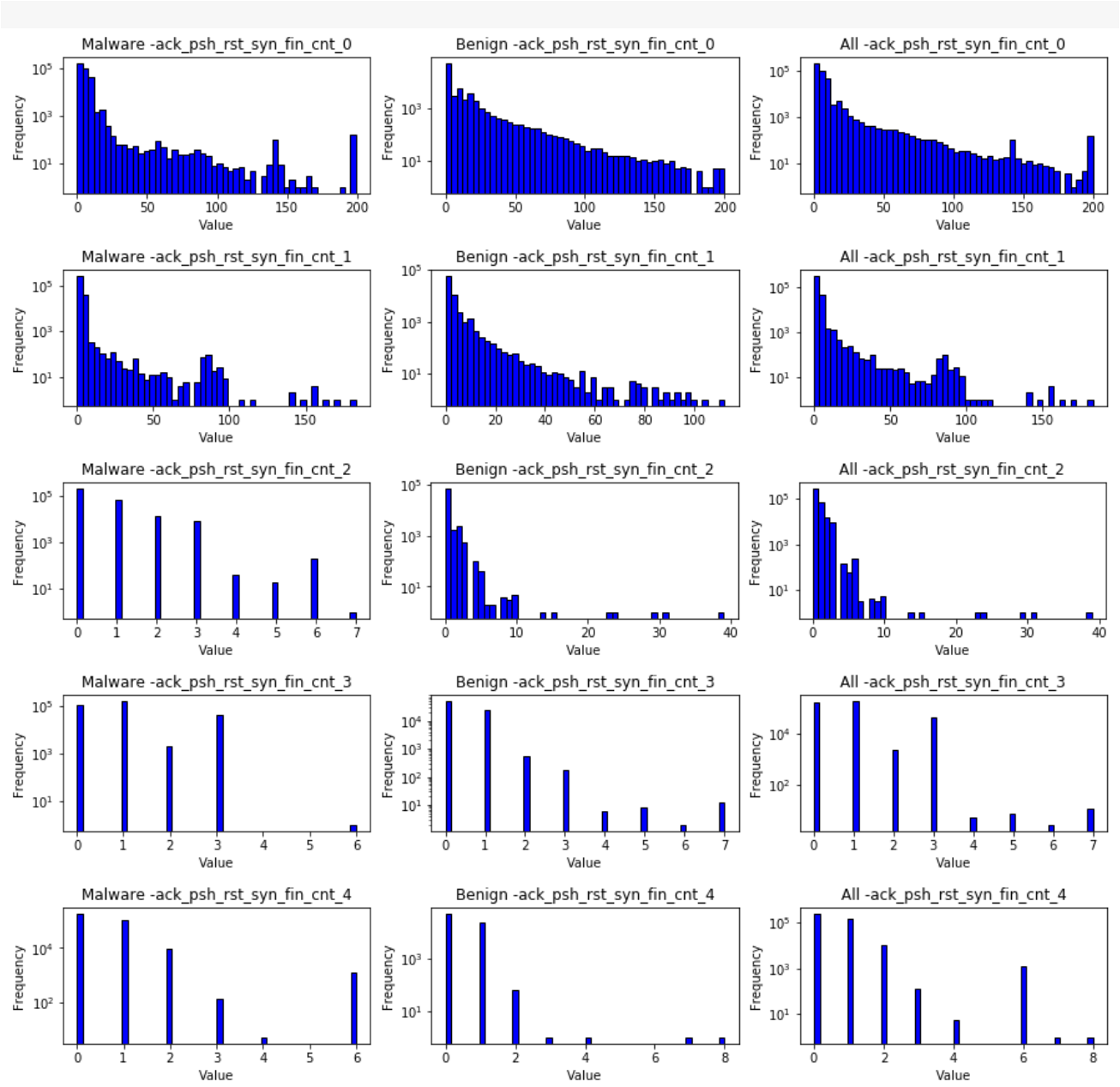
Liverpool John Moore's University, Liverpool, UK

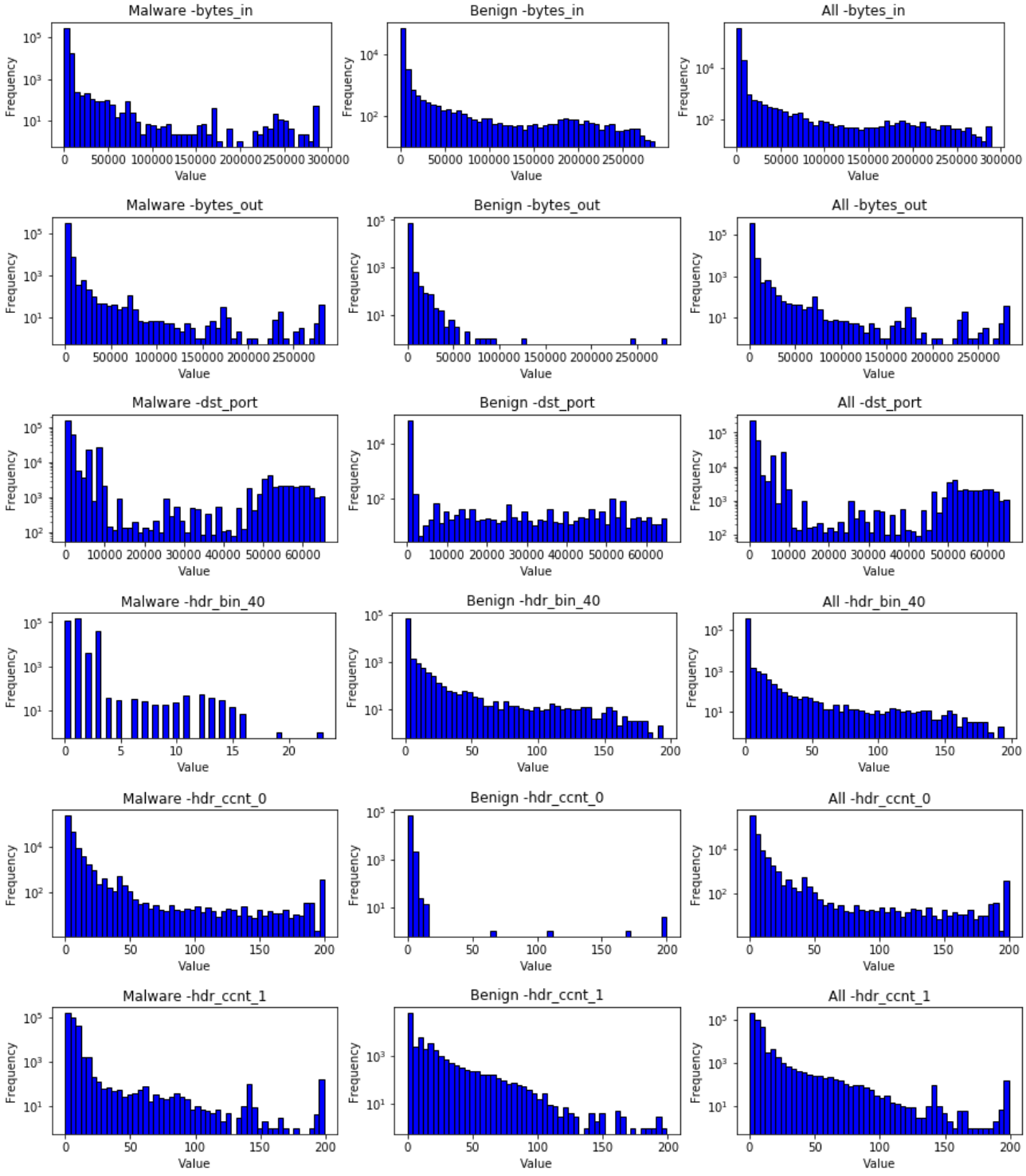
Research Supervisor : Snehansu Sekhar Sahu

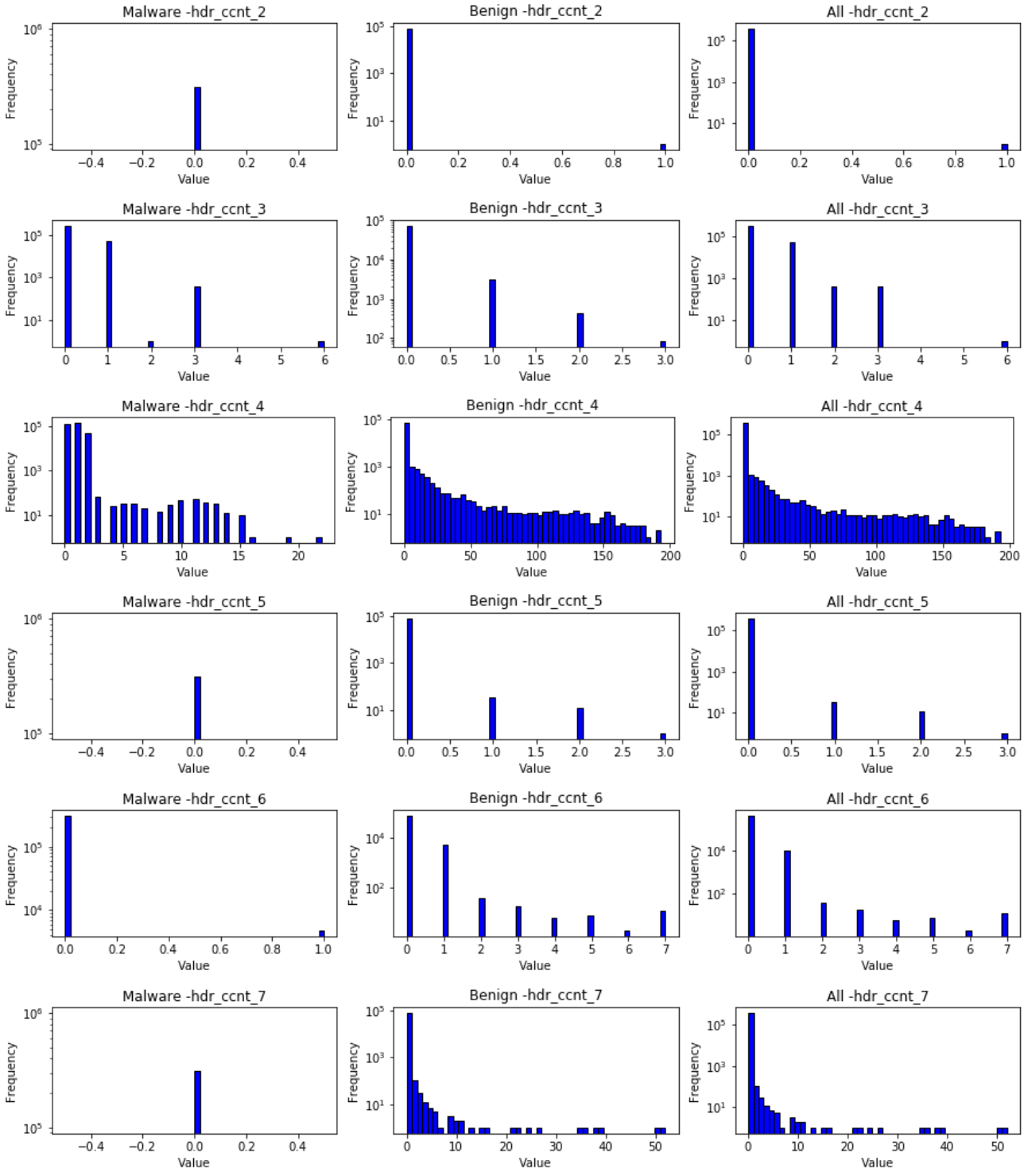
APPENDIX B: Exploratory Data Analysis

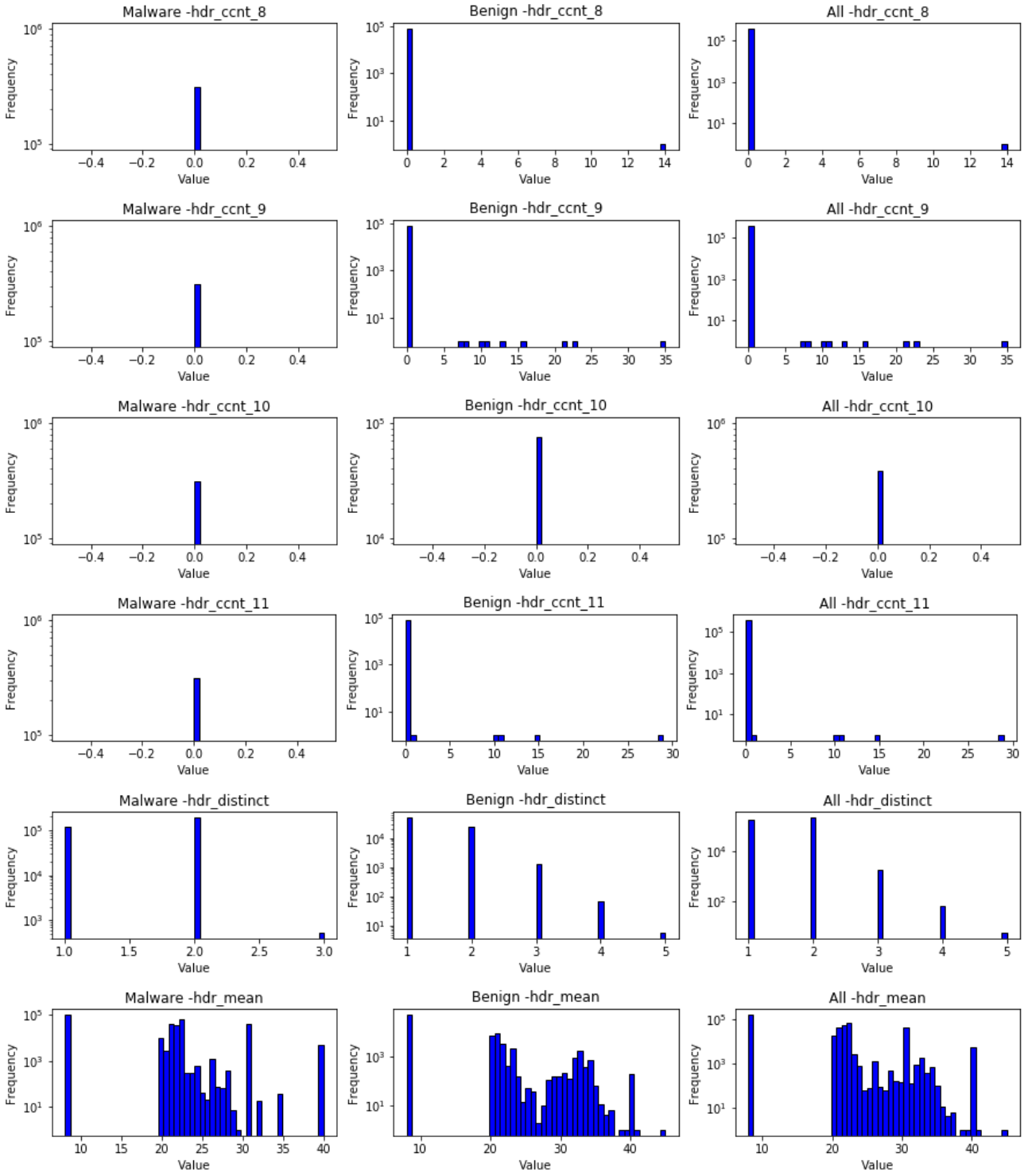
NetML Dataset

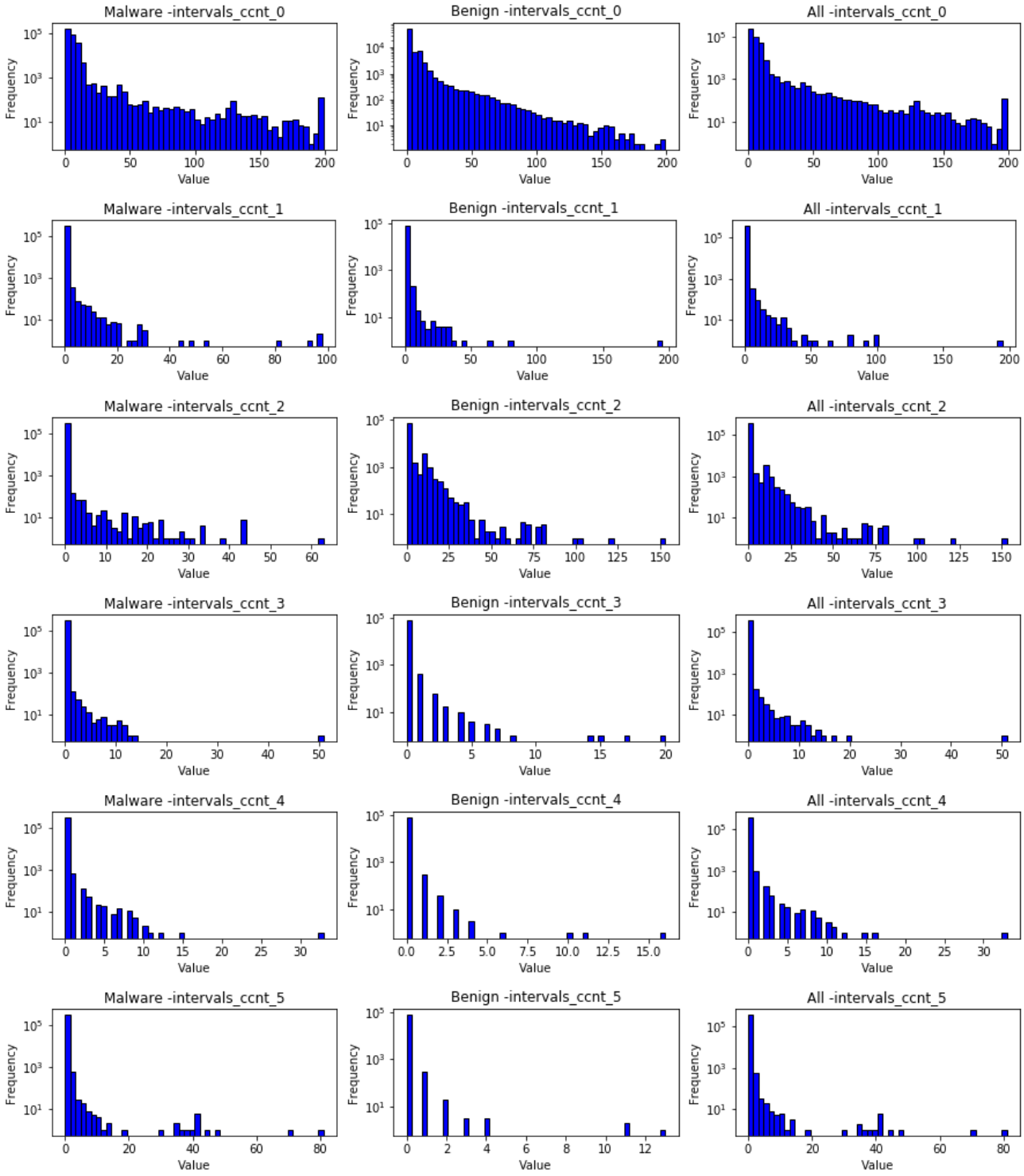
Univariate Analysis – Distribution Plots

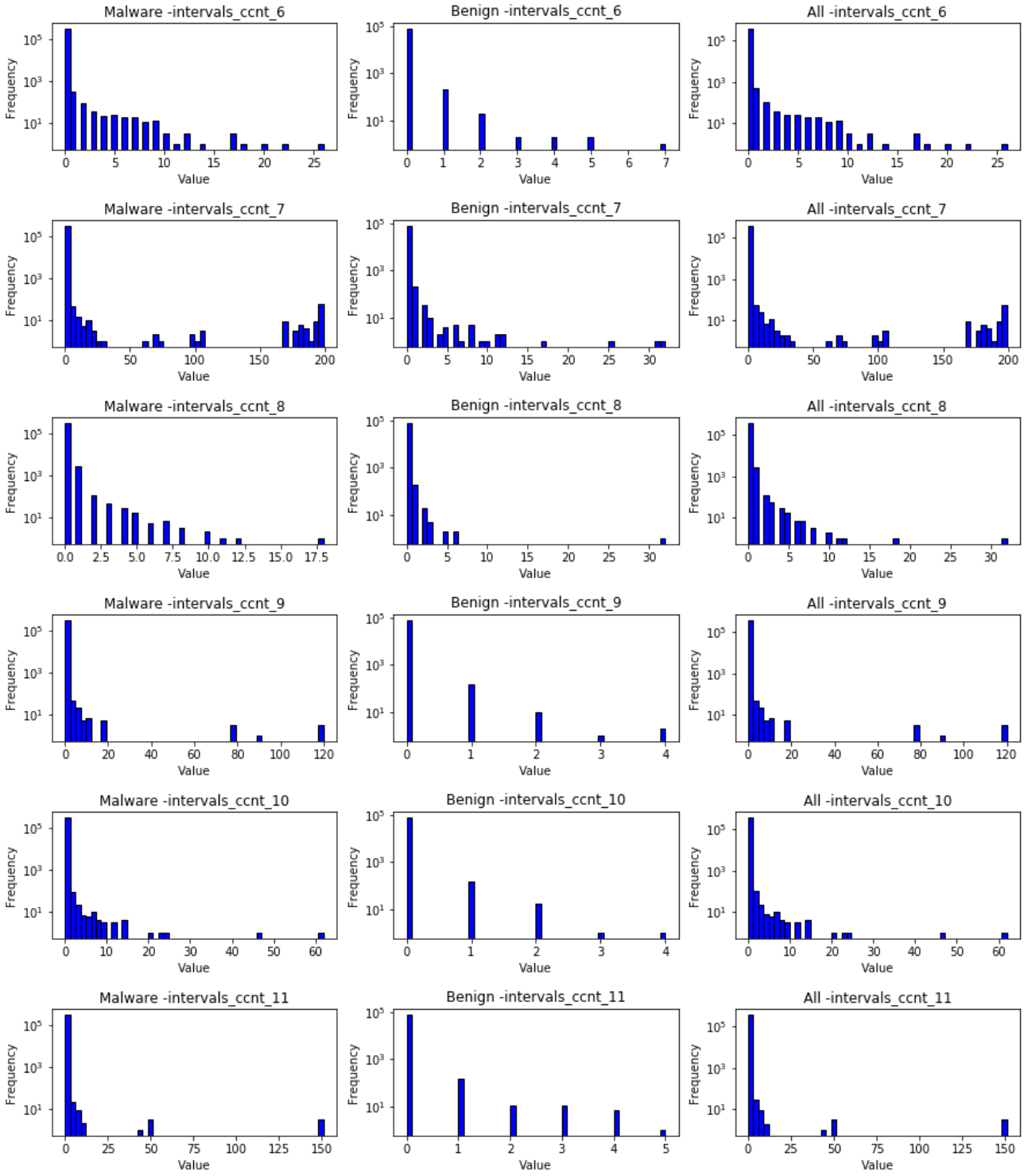


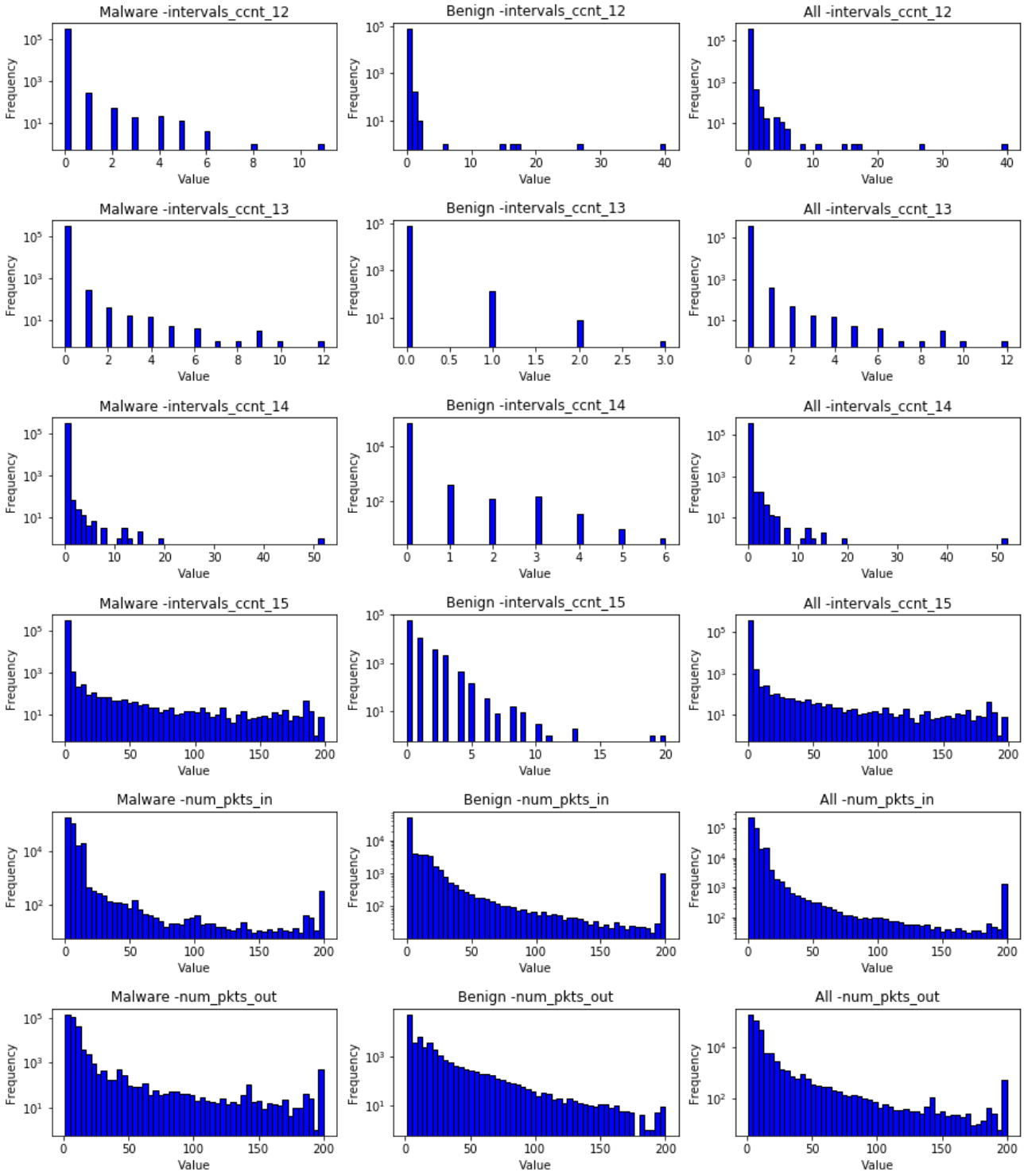

Malware -bytes_in
Benign -bytes_in
All -bytes_in
Malware -bytes_out
Benign -bytes_out
All -bytes_out
Malware -dst_port
Benign -dst_port
All -dst_port
Malware -hdr_bin_40
Benign -hdr_bin_40
All -hdr_bin_40
Malware -hdr_ccnt_0
Benign -hdr_ccnt_0
All -hdr_ccnt_0
Malware -hdr_ccnt_1
Benign -hdr_ccnt_1
All -hdr_ccnt_1
Frequency
Value

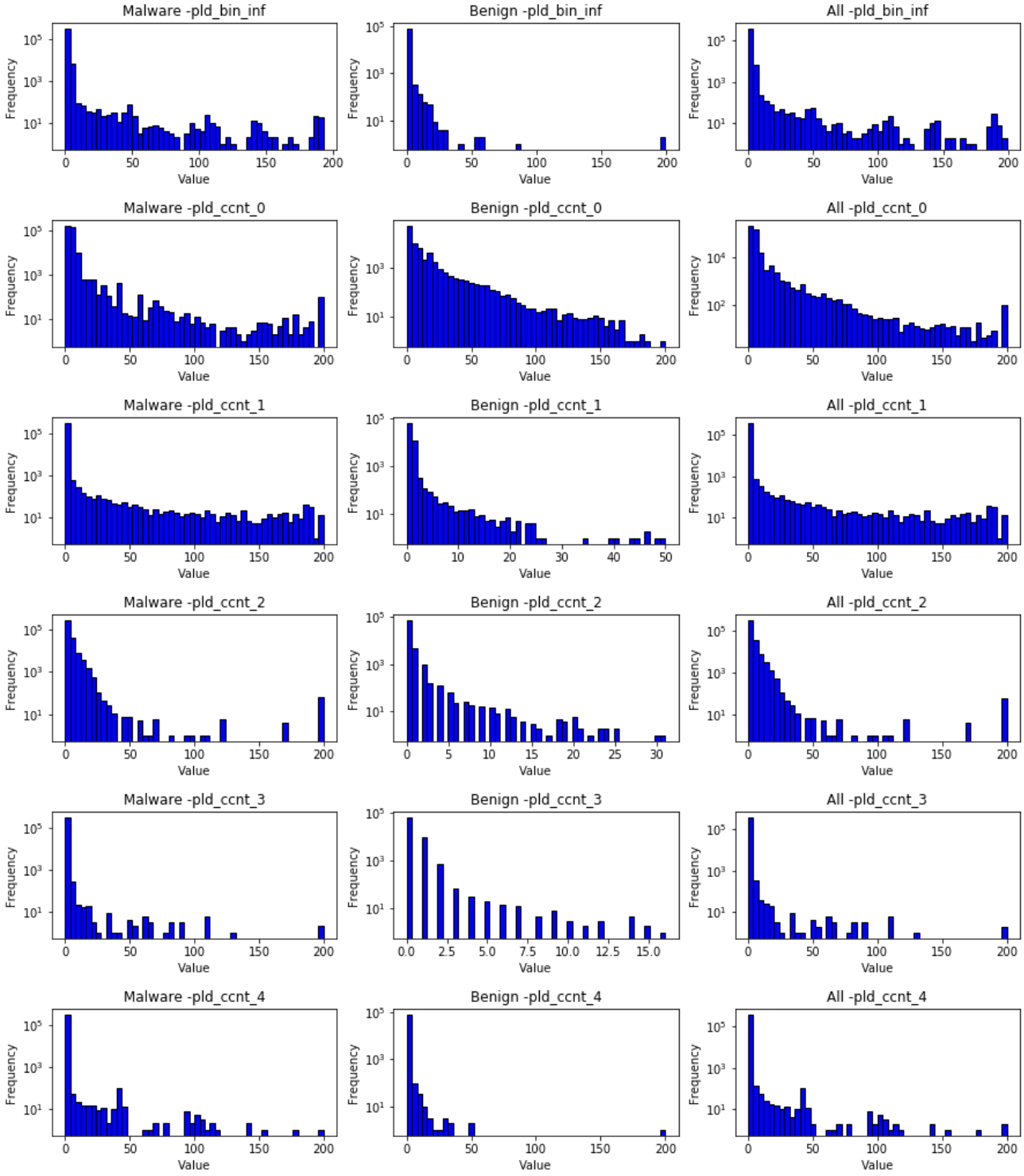

Malware -hdr_ccnt_2
Benign -hdr_ccnt_2
All -hdr_ccnt_2
Malware -hdr_ccnt_3
Benign -hdr_ccnt_3
All -hdr_ccnt_3
Malware -hdr_ccnt_4
Benign -hdr_ccnt_4
All -hdr_ccnt_4
Malware -hdr_ccnt_5
Benign -hdr_ccnt_5
All -hdr_ccnt_5
Malware -hdr_ccnt_6
Benign -hdr_ccnt_6
All -hdr_ccnt_6
Malware -hdr_ccnt_7
Benign -hdr_ccnt_7
All -hdr_ccnt_7
Frequency
Value

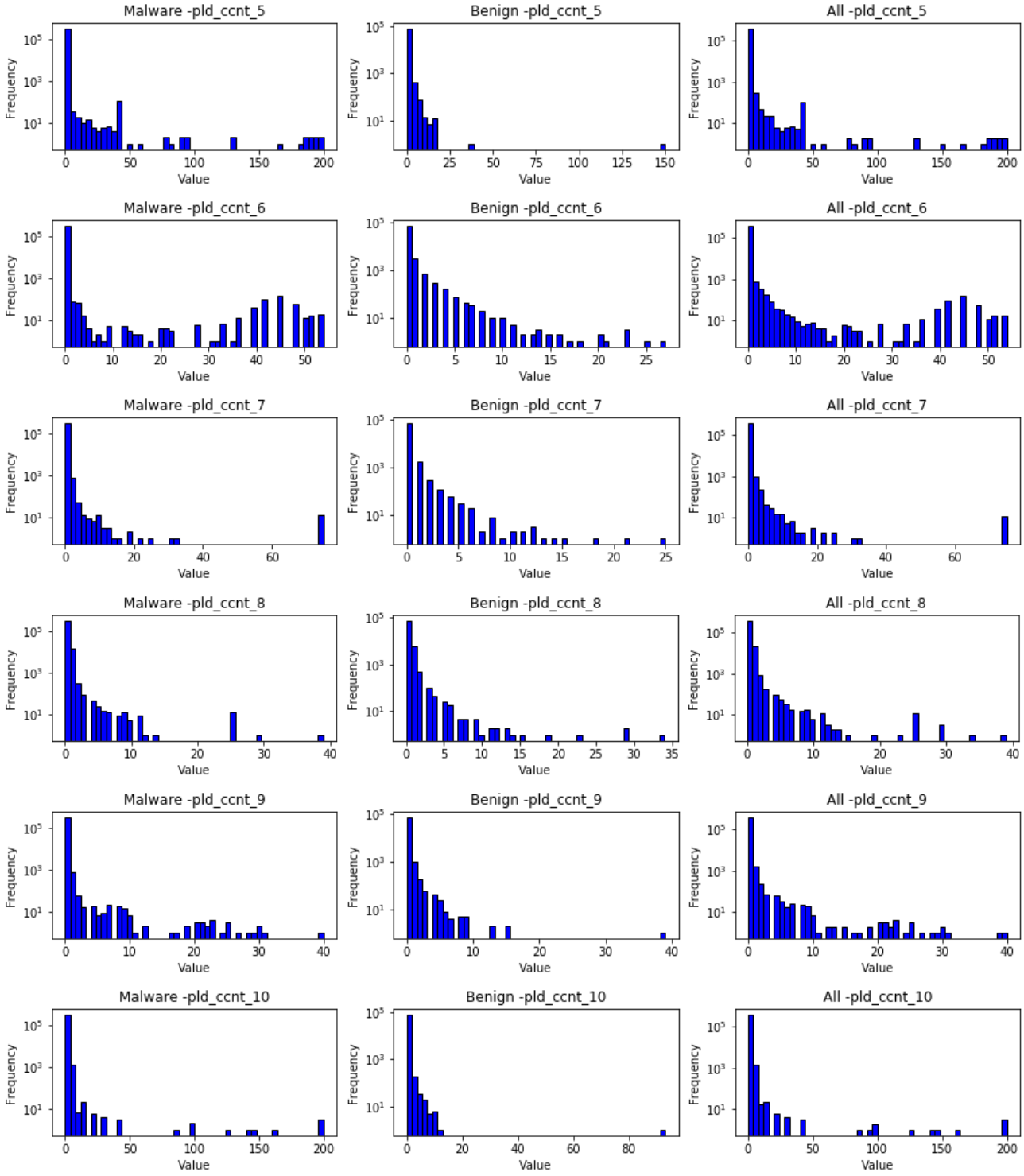

Malware -hdr_ccnt_8
Benign -hdr_ccnt_8
All -hdr_ccnt_8
Malware -hdr_ccnt_9
Benign -hdr_ccnt_9
All -hdr_ccnt_9
Malware -hdr_ccnt_10
Benign -hdr_ccnt_10
All -hdr_ccnt_10
Malware -hdr_ccnt_11
Benign -hdr_ccnt_11
All -hdr_ccnt_11
Malware -hdr_distinct
Benign -hdr_distinct
All -hdr_distinct
Malware -hdr_mean
Benign -hdr_mean
All -hdr_mean
Frequency
Value

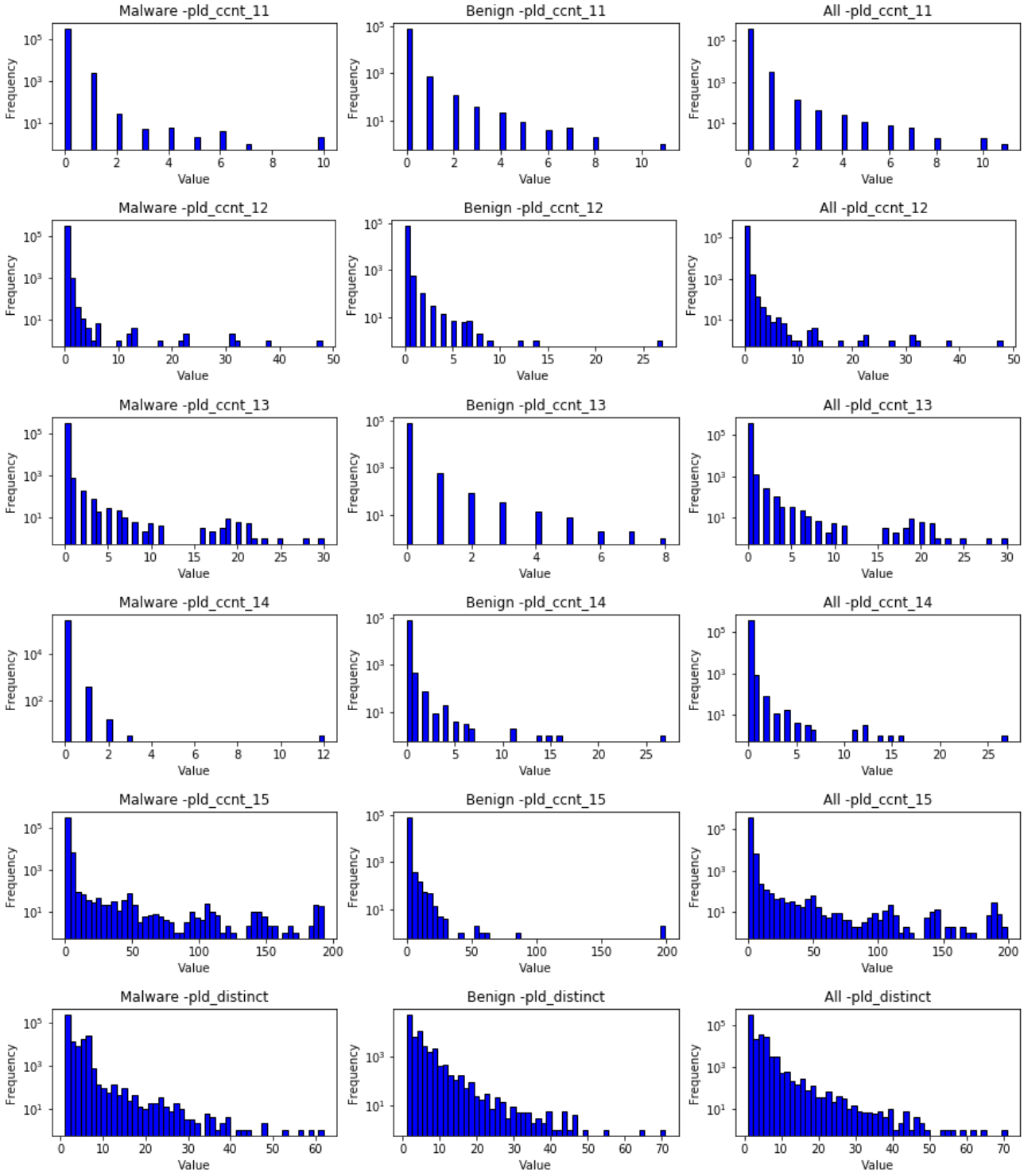

Malware -intervals_ccnt_0
Benign -intervals_ccnt_0
All -intervals_ccnt_0
Malware -intervals_ccnt_1
Benign -intervals_ccnt_1
All -intervals_ccnt_1
Malware -intervals_ccnt_2
Benign -intervals_ccnt_2
All -intervals_ccnt_2
Malware -intervals_ccnt_3
Benign -intervals_ccnt_3
All -intervals_ccnt_3
Malware -intervals_ccnt_4
Benign -intervals_ccnt_4
All -intervals_ccnt_4
Malware -intervals_ccnt_5
Benign -intervals_ccnt_5
All -intervals_ccnt_5
Frequency
Value

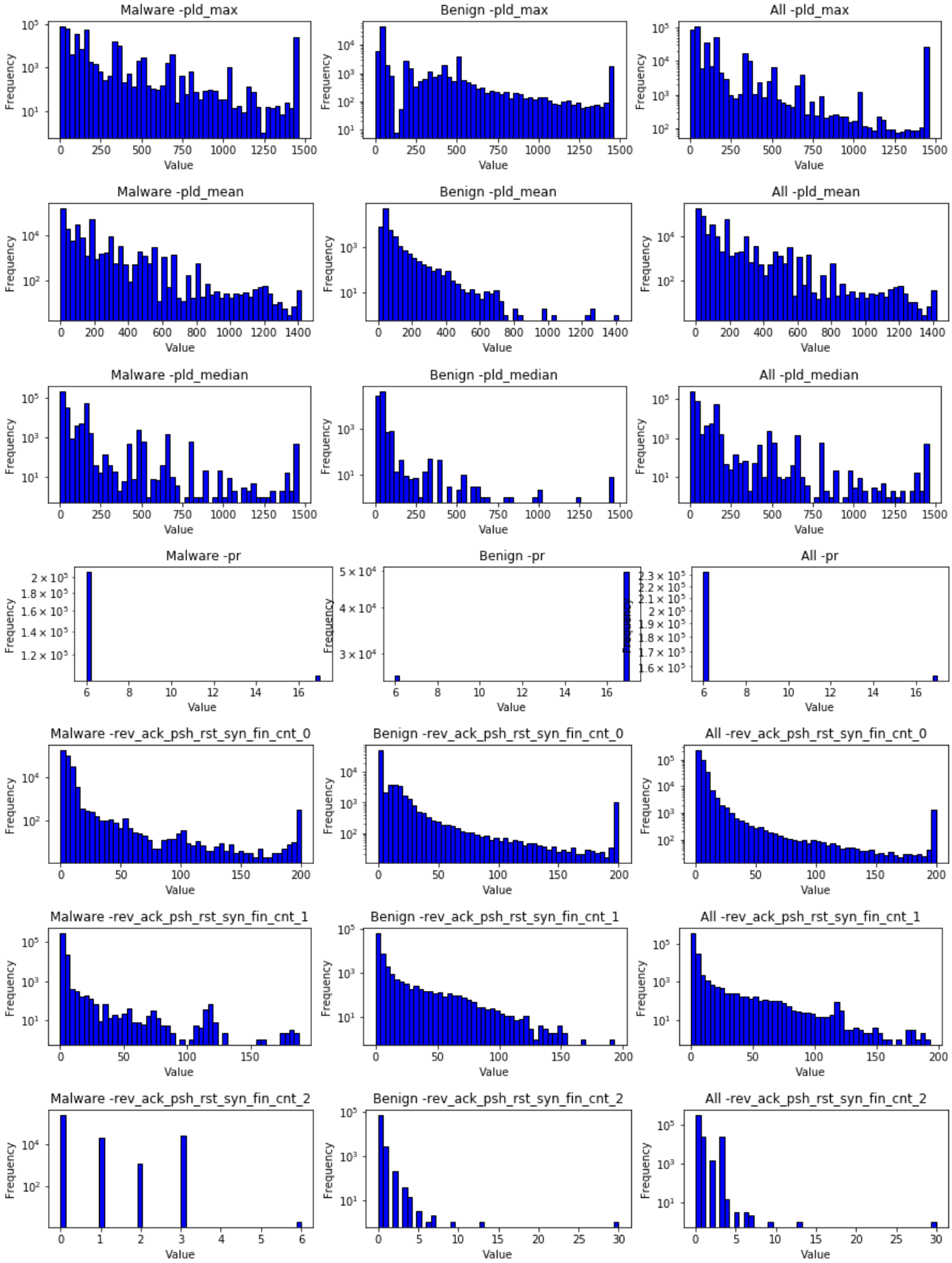

Malware -intervals_ccnt_6
Benign -intervals_ccnt_6
All -intervals_ccnt_6
Malware -intervals_ccnt_7
Benign -intervals_ccnt_7
All -intervals_ccnt_7
Malware -intervals_ccnt_8
Benign -intervals_ccnt_8
All -intervals_ccnt_8
Malware -intervals_ccnt_9
Benign -intervals_ccnt_9
All -intervals_ccnt_9
Malware -intervals_ccnt_10
Benign -intervals_ccnt_10
All -intervals_ccnt_10
Malware -intervals_ccnt_11
Benign -intervals_ccnt_11
All -intervals_ccnt_11
Frequency
Value

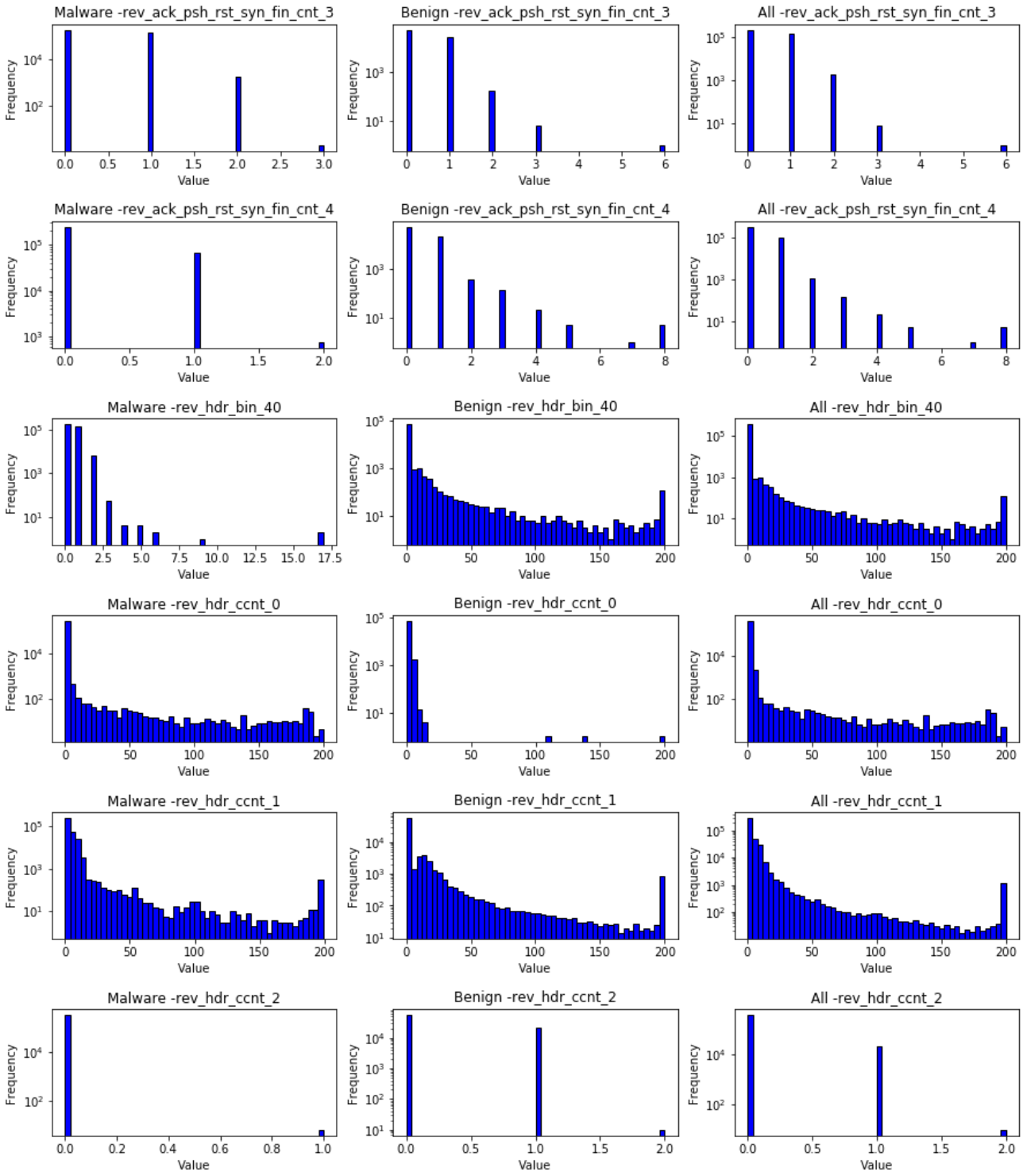

Malware -intervals_ccnt_12
Benign -intervals_ccnt_12
All -intervals_ccnt_12
Malware -intervals_ccnt_13
Benign -intervals_ccnt_13
All -intervals_ccnt_13
Malware -intervals_ccnt_14
Benign -intervals_ccnt_14
All -intervals_ccnt_14
Malware -intervals_ccnt_15
Benign -intervals_ccnt_15
All -intervals_ccnt_15
Malware -num_pkts_in
Benign -num_pkts_in
All -num_pkts_in
Malware -num_pkts_out
Benign -num_pkts_out
All -num_pkts_out
Frequency
Value

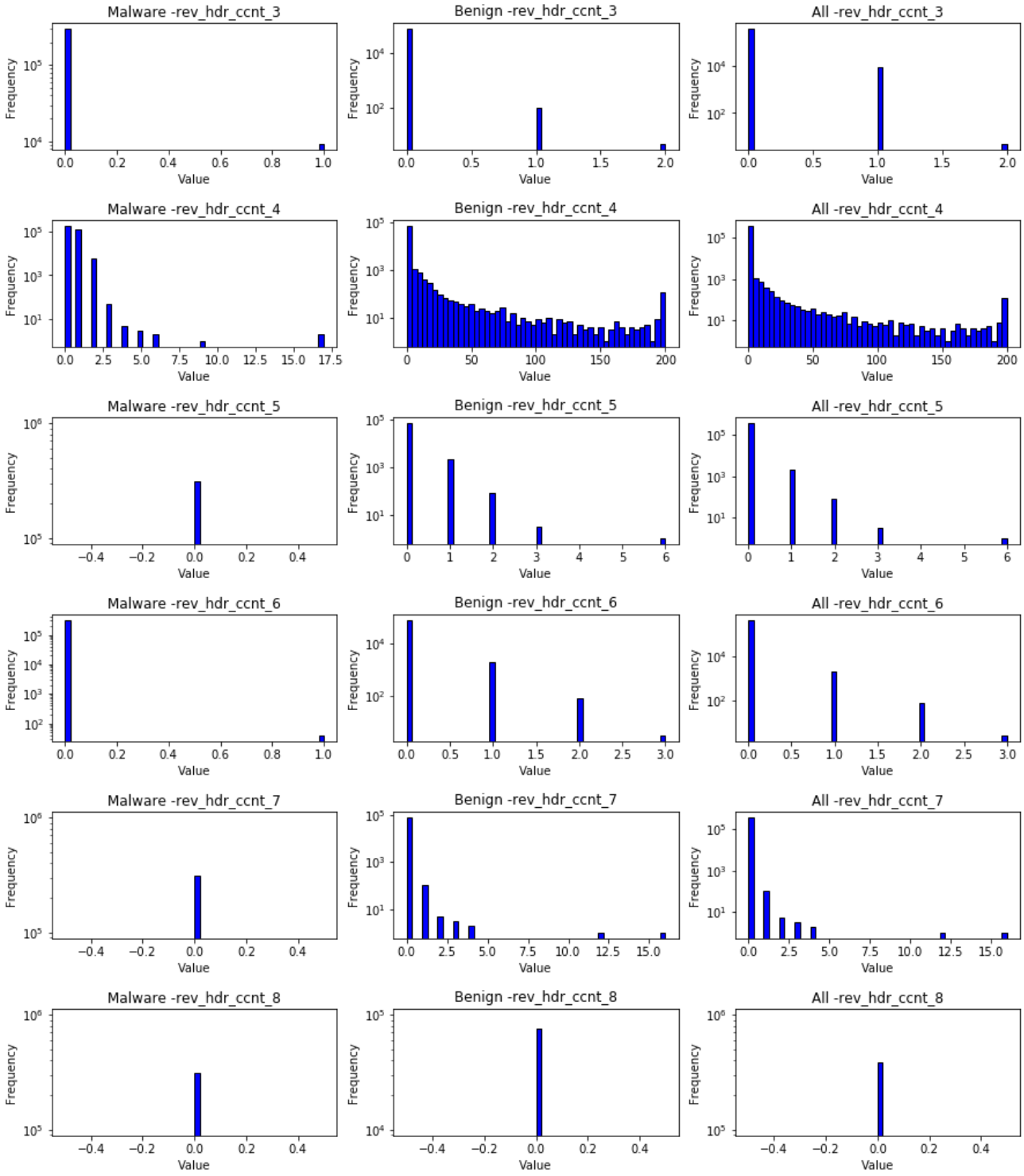

Malware -pld_bin_inf
Benign -pld_bin_inf
All -pld_bin_inf
Malware -pld_ccnt_0
Benign -pld_ccnt_0
All -pld_ccnt_0
Malware -pld_ccnt_1
Benign -pld_ccnt_1
All -pld_ccnt_1
Malware -pld_ccnt_2
Benign -pld_ccnt_2
All -pld_ccnt_2
Malware -pld_ccnt_3
Benign -pld_ccnt_3
All -pld_ccnt_3
Malware -pld_ccnt_4
Benign -pld_ccnt_4
All -pld_ccnt_4
Frequency
Value

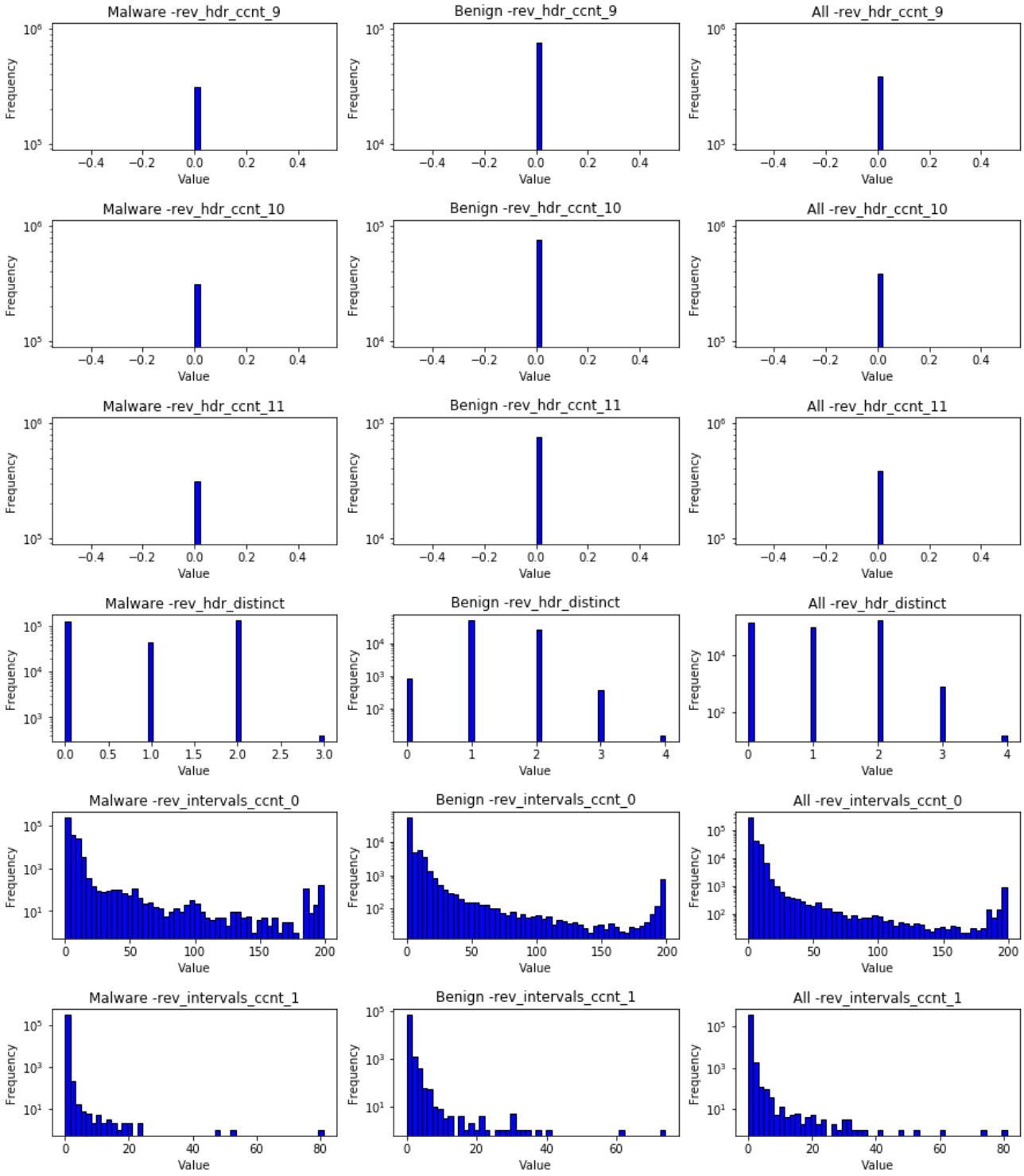

Malware -pld_ccnt_5
Benign -pld_ccnt_5
All -pld_ccnt_5
Malware -pld_ccnt_6
Benign -pld_ccnt_6
All -pld_ccnt_6
Malware -pld_ccnt_7
Benign -pld_ccnt_7
All -pld_ccnt_7
Malware -pld_ccnt_8
Benign -pld_ccnt_8
All -pld_ccnt_8
Malware -pld_ccnt_9
Benign -pld_ccnt_9
All -pld_ccnt_9
Malware -pld_ccnt_10
Benign -pld_ccnt_10
All -pld_ccnt_10
Frequency
Value

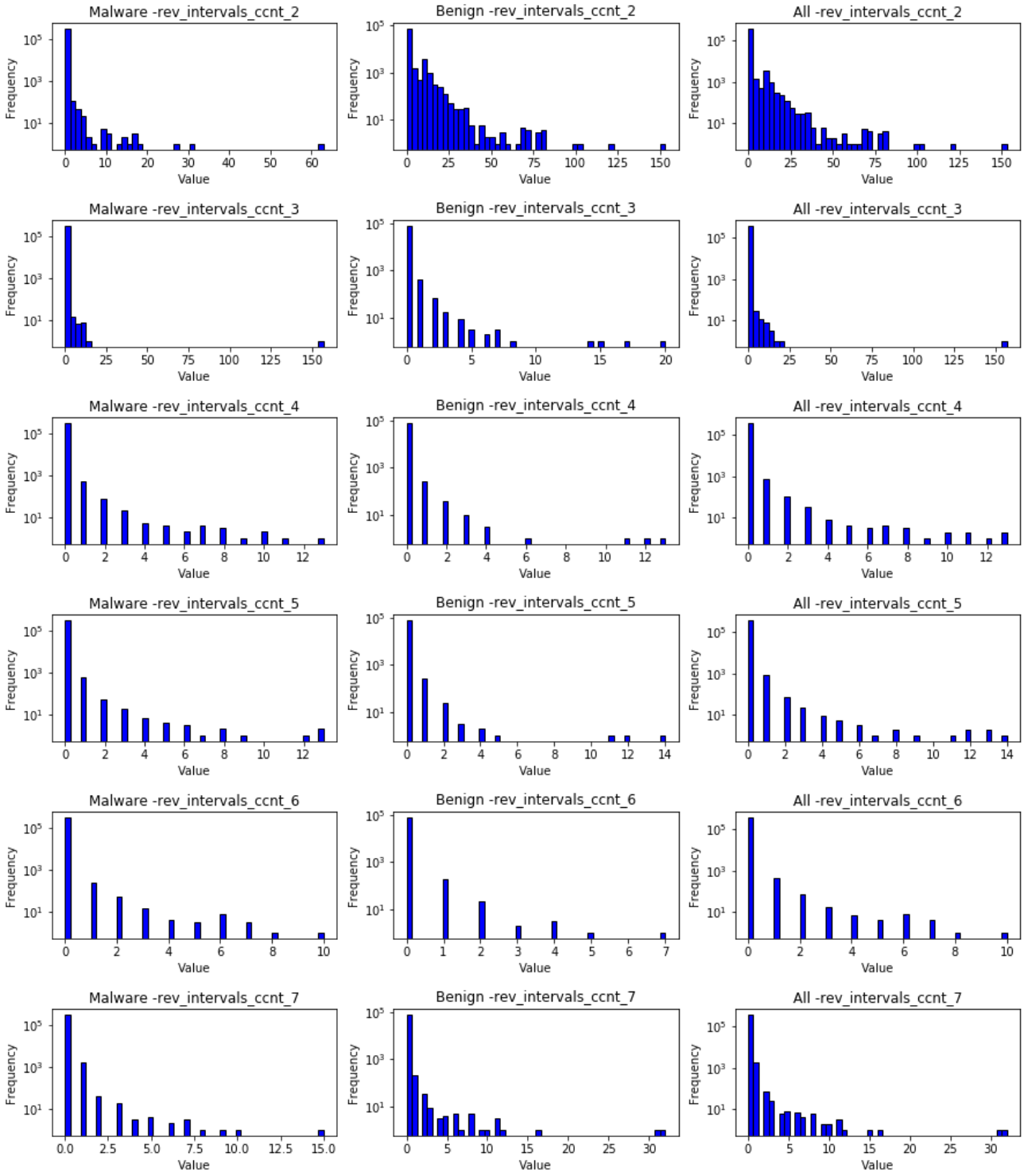

Malware -pld_ccnt_11
Benign -pld_ccnt_11
All -pld_ccnt_11
Malware -pld_ccnt_12
Benign -pld_ccnt_12
All -pld_ccnt_12
Malware -pld_ccnt_13
Benign -pld_ccnt_13
All -pld_ccnt_13
Malware -pld_ccnt_14
Benign -pld_ccnt_14
All -pld_ccnt_14
Malware -pld_ccnt_15
Benign -pld_ccnt_15
All -pld_ccnt_15
Malware -pld_distinct
Benign -pld_distinct
All -pld_distinct
Frequency
Value

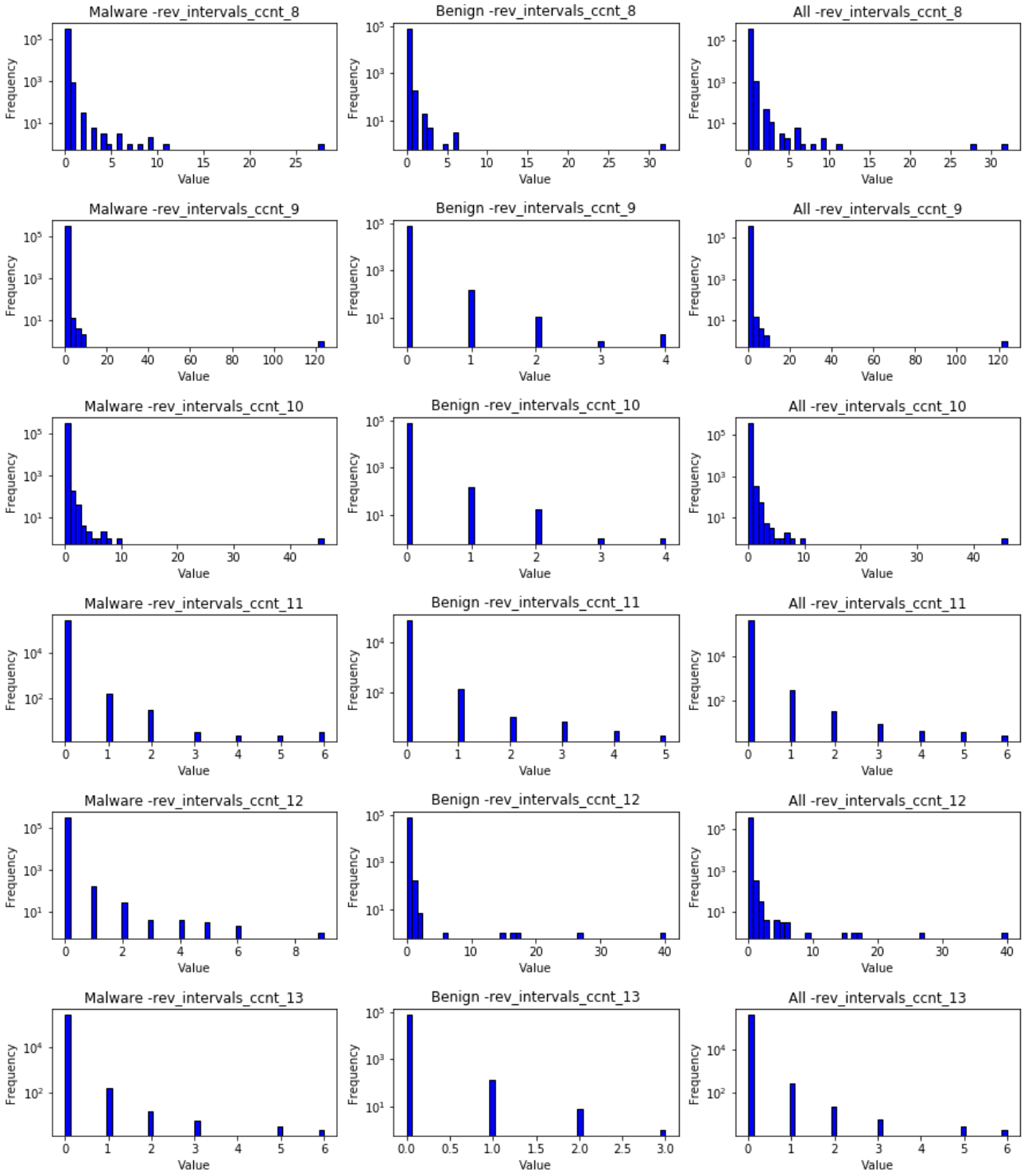

Malware -pld_max
Benign -pld_max
All -pld_max
Malware -pld_mean
Benign -pld_mean
All -pld_mean
Malware -pld_median
Benign -pld_median
All -pld_median
Malware -pr
Benign -pr
All -pr
Malware -rev_ack_psh_rst_syn_fin_cnt_0
Benign -rev_ack_psh_rst_syn_fin_cnt_0
All -rev_ack_psh_rst_syn_fin_cnt_0
Malware -rev_ack_psh_rst_syn_fin_cnt_1
Benign -rev_ack_psh_rst_syn_fin_cnt_1
All -rev_ack_psh_rst_syn_fin_cnt_1
Malware -rev_ack_psh_rst_syn_fin_cnt_2
Benign -rev_ack_psh_rst_syn_fin_cnt_2
All -rev_ack_psh_rst_syn_fin_cnt_2
Frequency
Value

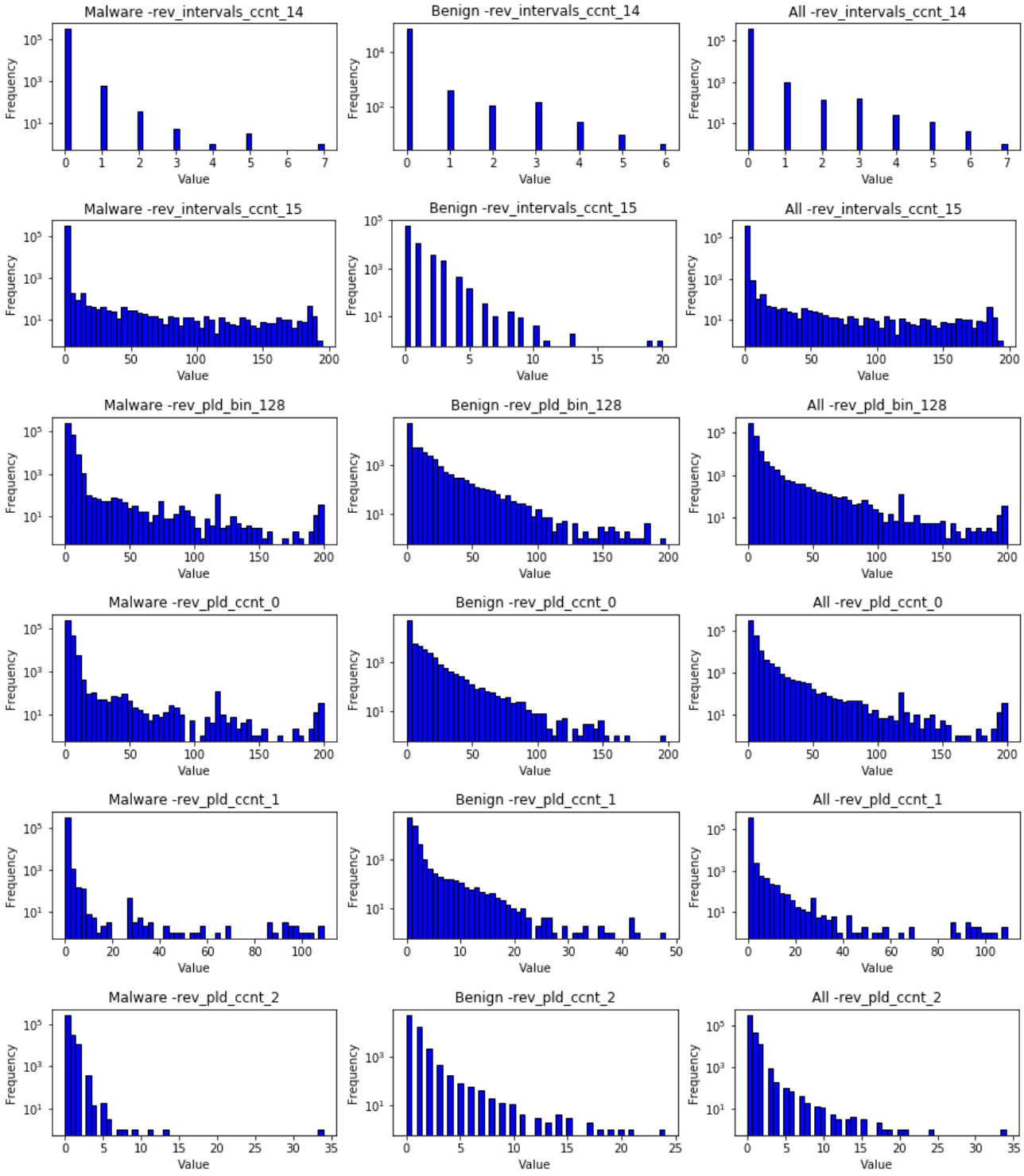

Malware -rev_ack_psh_rst_syn_fin_cnt_3
Benign -rev_ack_psh_rst_syn_fin_cnt_3
All -rev_ack_psh_rst_syn_fin_cnt_3
Malware -rev_ack_psh_rst_syn_fin_cnt_4
Benign -rev_ack_psh_rst_syn_fin_cnt_4
All -rev_ack_psh_rst_syn_fin_cnt_4
Malware -rev_hdr_bin_40
Benign -rev_hdr_bin_40
All -rev_hdr_bin_40
Malware -rev_hdr_ccnt_0
Benign -rev_hdr_ccnt_0
All -rev_hdr_ccnt_0
Malware -rev_hdr_ccnt_1
Benign -rev_hdr_ccnt_1
All -rev_hdr_ccnt_1
Malware -rev_hdr_ccnt_2
Benign -rev_hdr_ccnt_2
All -rev_hdr_ccnt_2
Frequency
Value

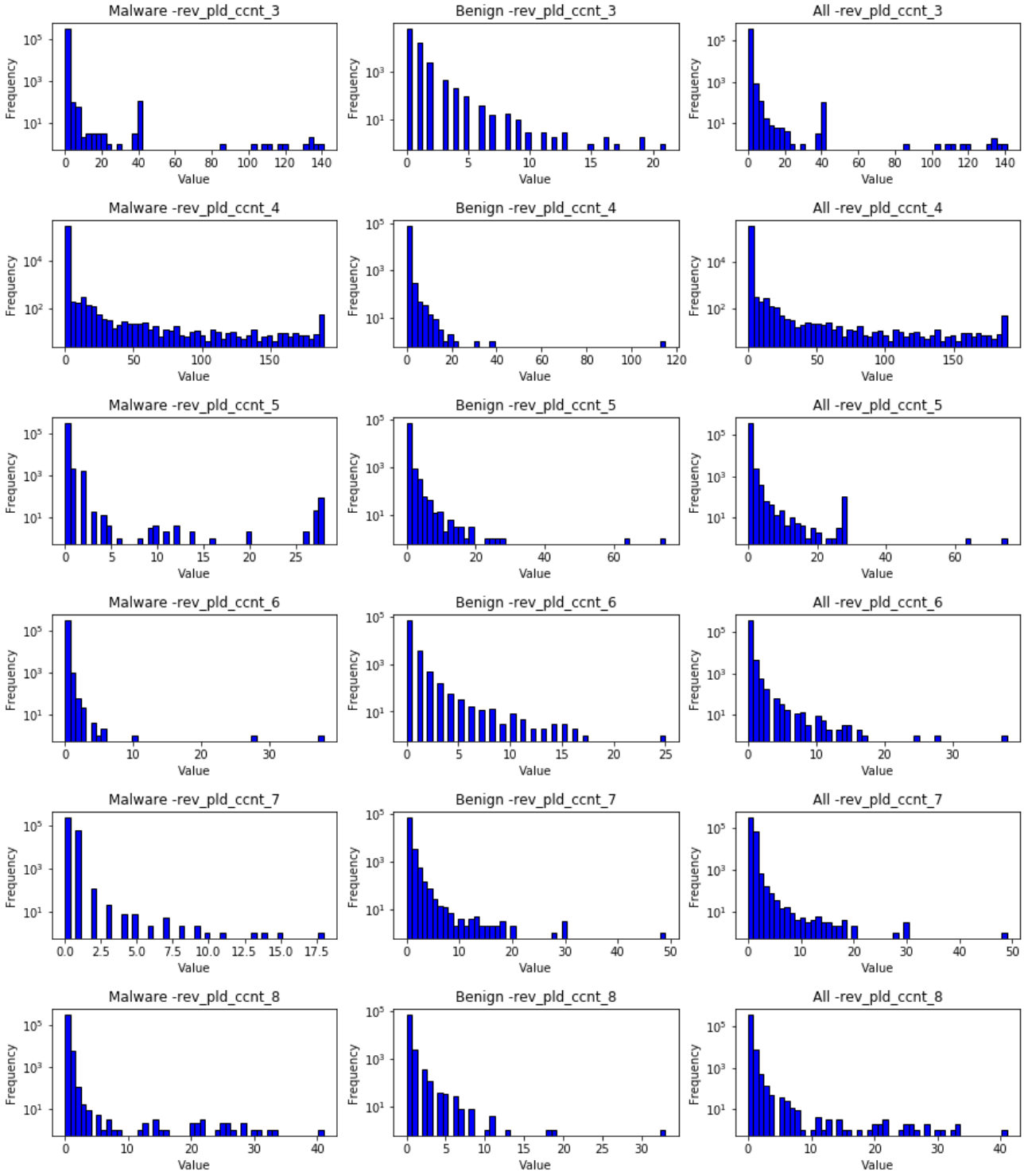

Malware -rev_hdr_ccnt_3
Benign -rev_hdr_ccnt_3
All -rev_hdr_ccnt_3
Malware -rev_hdr_ccnt_4
Benign -rev_hdr_ccnt_4
All -rev_hdr_ccnt_4
Malware -rev_hdr_ccnt_5
Benign -rev_hdr_ccnt_5
All -rev_hdr_ccnt_5
Malware -rev_hdr_ccnt_6
Benign -rev_hdr_ccnt_6
All -rev_hdr_ccnt_6
Malware -rev_hdr_ccnt_7
Benign -rev_hdr_ccnt_7
All -rev_hdr_ccnt_7
Malware -rev_hdr_ccnt_8
Benign -rev_hdr_ccnt_8
All -rev_hdr_ccnt_8
Frequency
Value

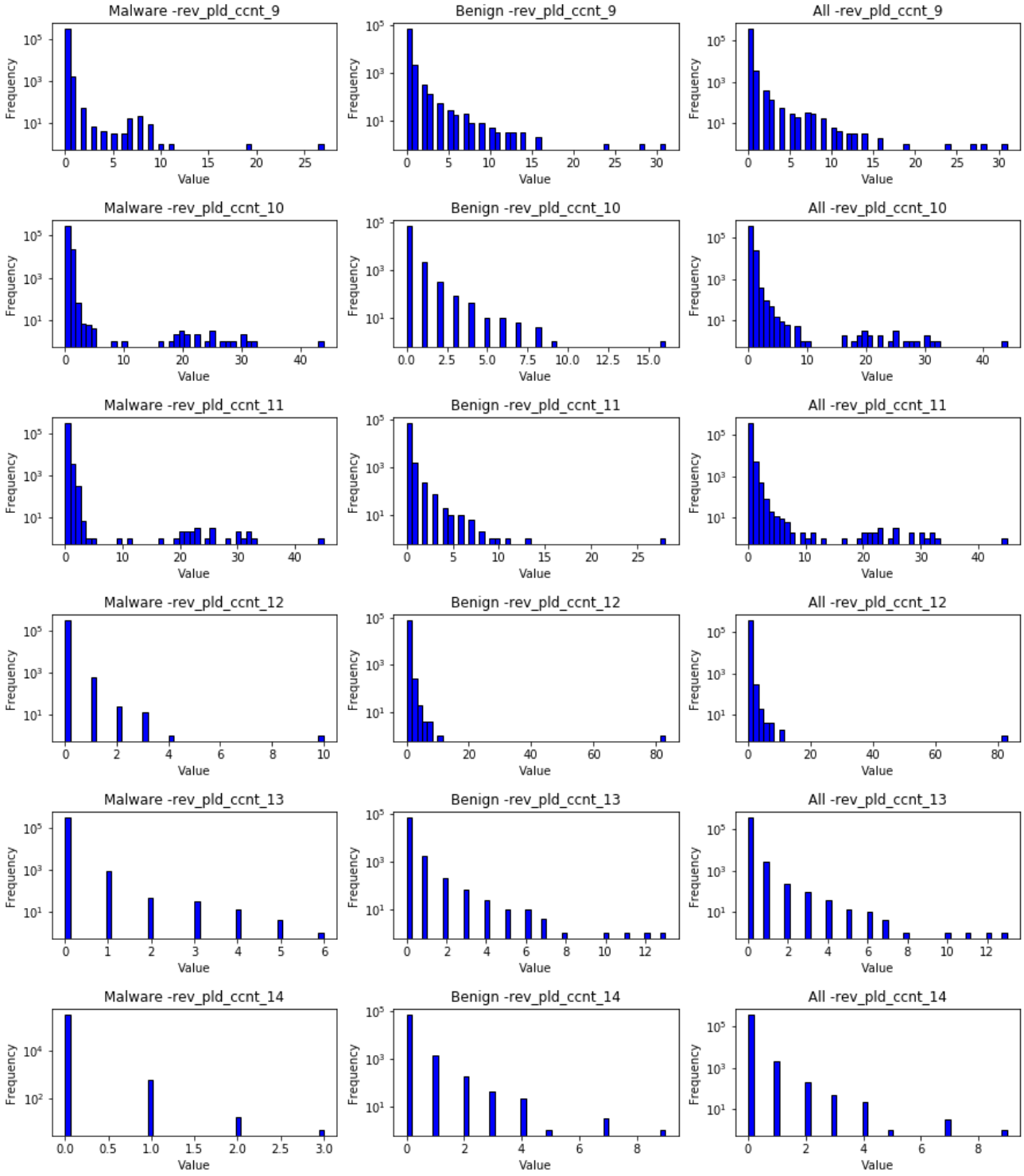

Malware -rev_hdr_ccnt_9
Benign -rev_hdr_ccnt_9
All -rev_hdr_ccnt_9
Malware -rev_hdr_ccnt_10
Benign -rev_hdr_ccnt_10
All -rev_hdr_ccnt_10
Malware -rev_hdr_ccnt_11
Benign -rev_hdr_ccnt_11
All -rev_hdr_ccnt_11
Malware -rev_hdr_distinct
Benign -rev_hdr_distinct
All -rev_hdr_distinct
Malware -rev_intervals_ccnt_0
Benign -rev_intervals_ccnt_0
All -rev_intervals_ccnt_0
Malware -rev_intervals_ccnt_1
Benign -rev_intervals_ccnt_1
All -rev_intervals_ccnt_1
Frequency
Value

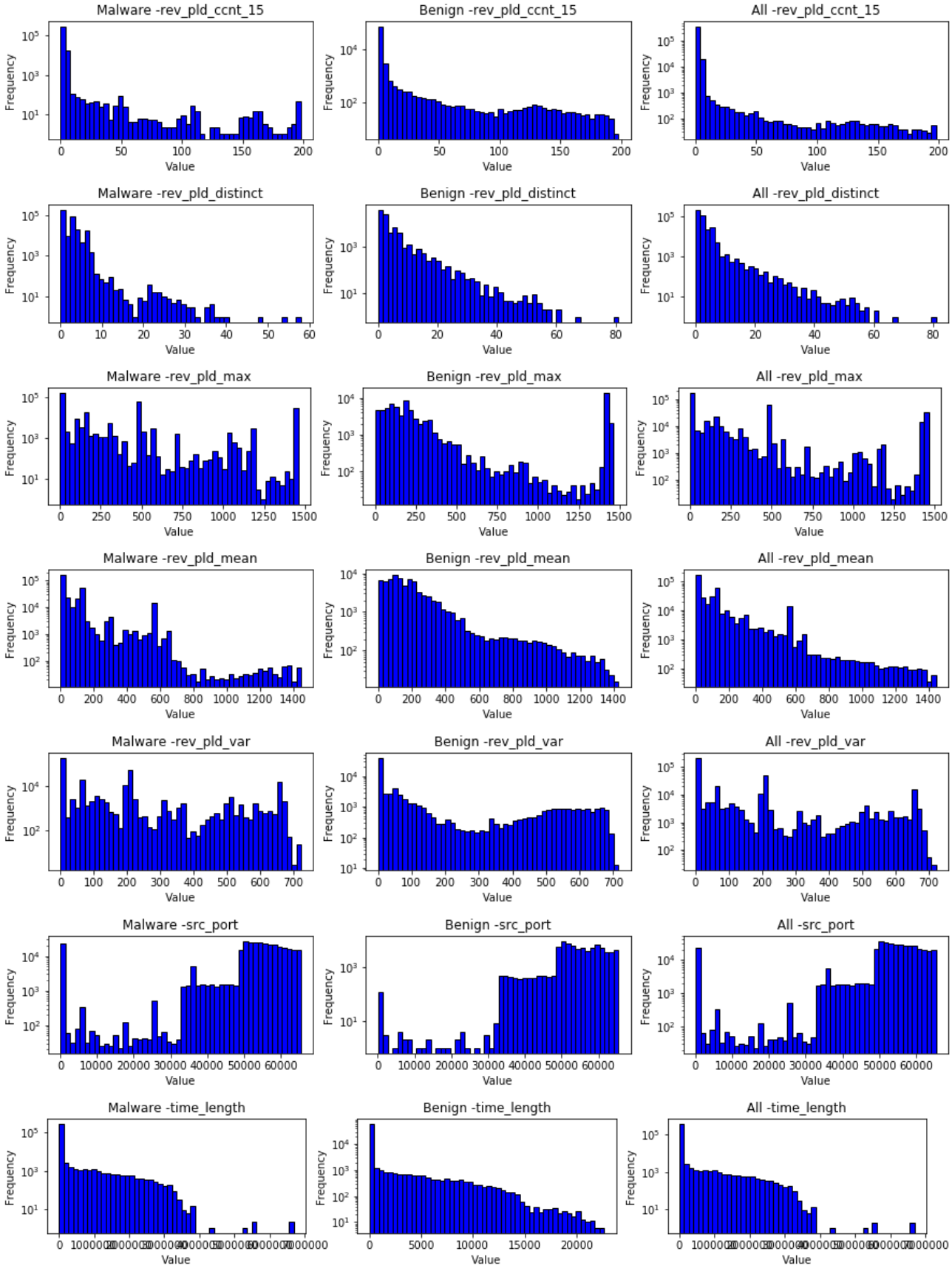

Malware -rev_intervals_ccnt_2
Benign -rev_intervals_ccnt_2
All -rev_intervals_ccnt_2
Malware -rev_intervals_ccnt_3
Benign -rev_intervals_ccnt_3
All -rev_intervals_ccnt_3
Malware -rev_intervals_ccnt_4
Benign -rev_intervals_ccnt_4
All -rev_intervals_ccnt_4
Malware -rev_intervals_ccnt_5
Benign -rev_intervals_ccnt_5
All -rev_intervals_ccnt_5
Malware -rev_intervals_ccnt_6
Benign -rev_intervals_ccnt_6
All -rev_intervals_ccnt_6
Malware -rev_intervals_ccnt_7
Benign -rev_intervals_ccnt_7
All -rev_intervals_ccnt_7
Frequency
Value

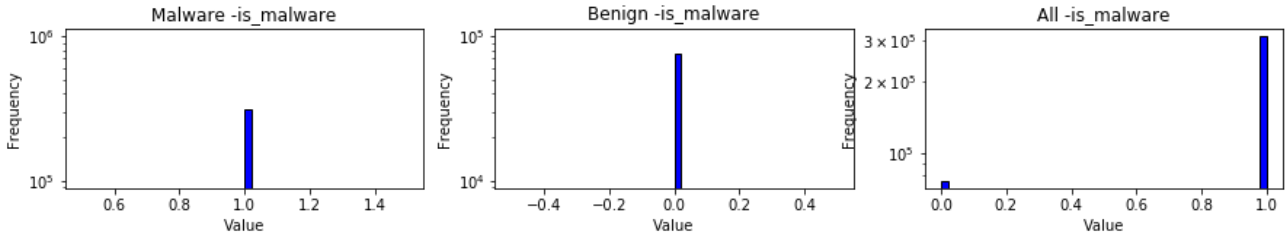

Malware -rev_intervals_ccnt_8
Benign -rev_intervals_ccnt_8
All -rev_intervals_ccnt_8
Malware -rev_intervals_ccnt_9
Benign -rev_intervals_ccnt_9
All -rev_intervals_ccnt_9
Malware -rev_intervals_ccnt_10
Benign -rev_intervals_ccnt_10
All -rev_intervals_ccnt_10
Malware -rev_intervals_ccnt_11
Benign -rev_intervals_ccnt_11
All -rev_intervals_ccnt_11
Malware -rev_intervals_ccnt_12
Benign -rev_intervals_ccnt_12
All -rev_intervals_ccnt_12
Malware -rev_intervals_ccnt_13
Benign -rev_intervals_ccnt_13
All -rev_intervals_ccnt_13
Frequency
Value


Malware -rev_intervals_ccnt_14
Benign -rev_intervals_ccnt_14
All -rev_intervals_ccnt_14
Malware -rev_intervals_ccnt_15
Benign -rev_intervals_ccnt_15
All -rev_intervals_ccnt_15
Malware -rev_pld_bin_128
Benign -rev_pld_bin_128
All -rev_pld_bin_128
Malware -rev_pld_ccnt_0
Benign -rev_pld_ccnt_0
All -rev_pld_ccnt_0
Malware -rev_pld_ccnt_1
Benign -rev_pld_ccnt_1
All -rev_pld_ccnt_1
Malware -rev_pld_ccnt_2
Benign -rev_pld_ccnt_2
All -rev_pld_ccnt_2
Frequency
Value

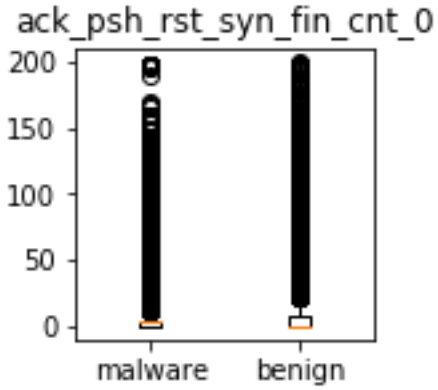

Malware -rev_pld_ccnt_3
Benign -rev_pld_ccnt_3
All -rev_pld_ccnt_3
Malware -rev_pld_ccnt_4
Benign -rev_pld_ccnt_4
All -rev_pld_ccnt_4
Malware -rev_pld_ccnt_5
Benign -rev_pld_ccnt_5
All -rev_pld_ccnt_5
Malware -rev_pld_ccnt_6
Benign -rev_pld_ccnt_6
All -rev_pld_ccnt_6
Malware -rev_pld_ccnt_7
Benign -rev_pld_ccnt_7
All -rev_pld_ccnt_7
Malware -rev_pld_ccnt_8
Benign -rev_pld_ccnt_8
All -rev_pld_ccnt_8
Frequency
Value

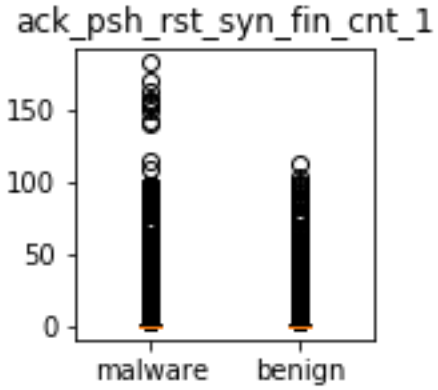

Malware -rev_pld_ccnt_9
Benign -rev_pld_ccnt_9
All -rev_pld_ccnt_9
Malware -rev_pld_ccnt_10
Benign -rev_pld_ccnt_10
All -rev_pld_ccnt_10
Malware -rev_pld_ccnt_11
Benign -rev_pld_ccnt_11
All -rev_pld_ccnt_11
Malware -rev_pld_ccnt_12
Benign -rev_pld_ccnt_12
All -rev_pld_ccnt_12
Malware -rev_pld_ccnt_13
Benign -rev_pld_ccnt_13
All -rev_pld_ccnt_13
Malware -rev_pld_ccnt_14
Benign -rev_pld_ccnt_14
All -rev_pld_ccnt_14
Frequency
Value

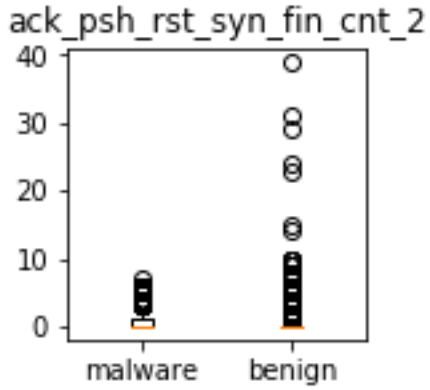

Malware -rev_pld_ccnt_15
Benign -rev_pld_ccnt_15
All -rev_pld_ccnt_15
Malware -rev_pld_distinct
Benign -rev_pld_distinct
All -rev_pld_distinct
Malware -rev_pld_max
Benign -rev_pld_max
All -rev_pld_max
Malware -rev_pld_mean
Benign -rev_pld_mean
All -rev_pld_mean
Malware -rev_pld_var
Benign -rev_pld_var
All -rev_pld_var
Malware -src_port
Benign -src_port
All -src_port
Malware -time_length
Benign -time_length
All -time_length
Frequency
Value

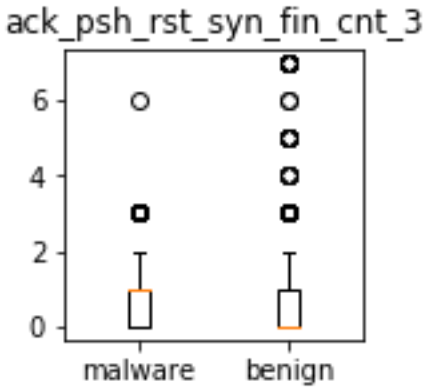


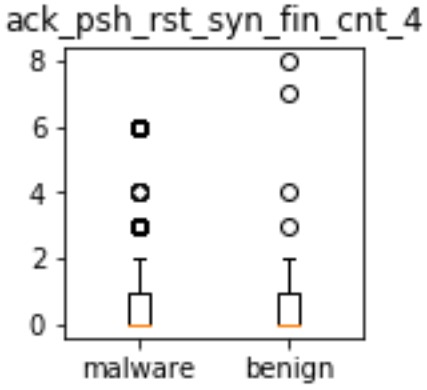
Univariate Analysis - Box Plots

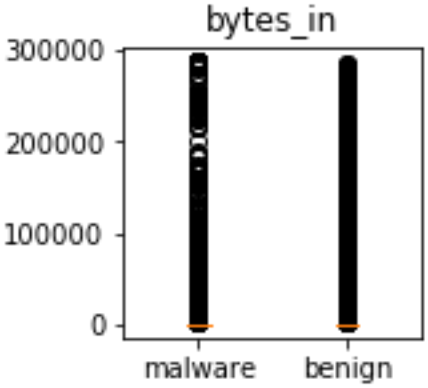


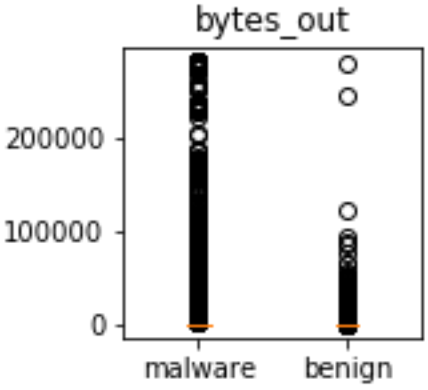


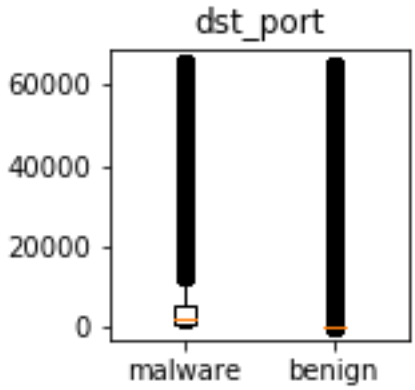


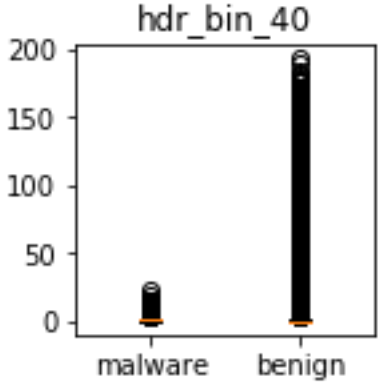

ack_psh_rst_syn_fin_cnt_3
6
4
2
0
malware
benign

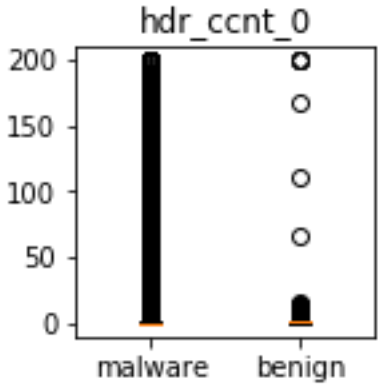

ack_psh_rst_syn_fin_cnt_4
8
6
4
2
0
malware
benign

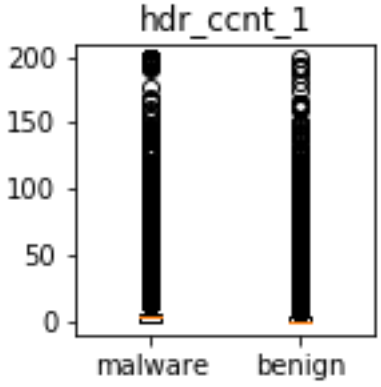

bytes_in
300000
200000
100000
0
malware
benign

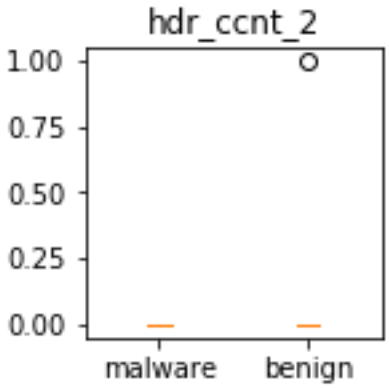

bytes_out
200000
100000
0
malware
benign

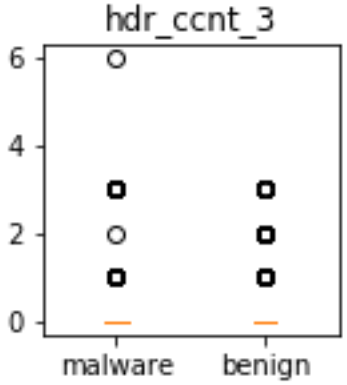

dst_port
60000
40000
20000
0
malware
benign

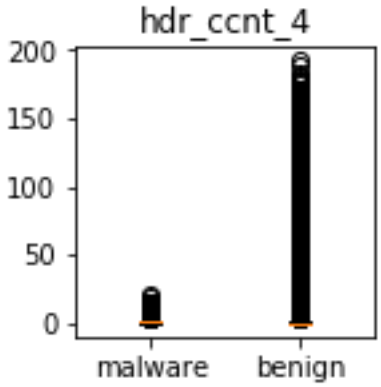

hdr_bin_40
200
150
100
50
0
malware
benign

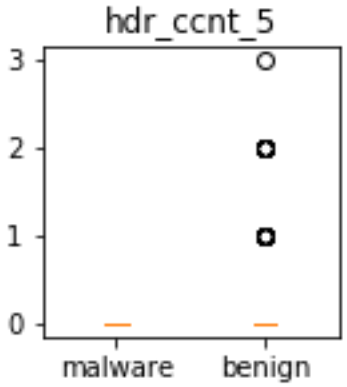

hdr_ccnt_0
200
150
100
50
0
malware
benign

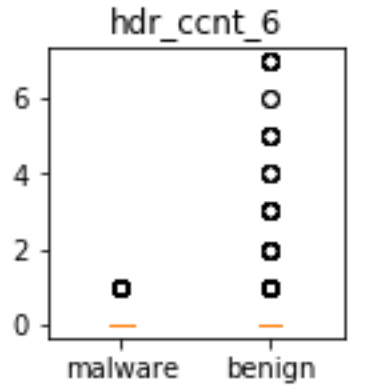

hdr_ccnt_1
200
150
100
50
0
malware
benign

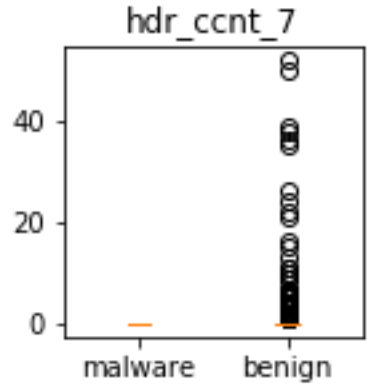

hdr_ccnt_2
1.00
0.75
0.50
0.25
0.00
malware
benign

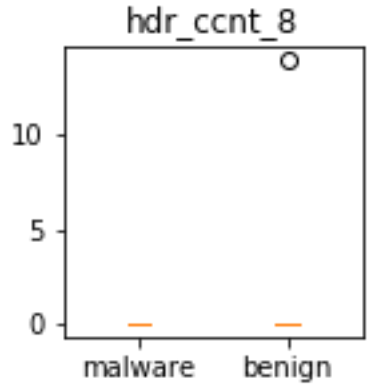

hdr_ccnt_3
6
4
2
0
malware
benign

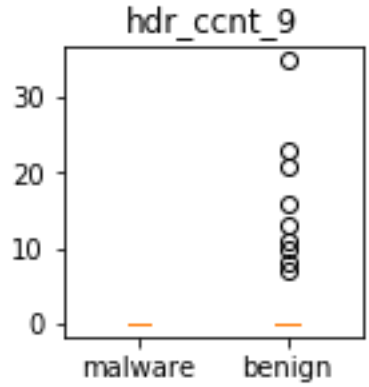

hdr_ccnt_4
200
150
100
50
0
malware
benign

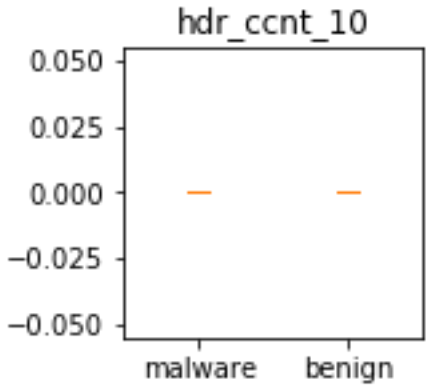

hdr_ccnt_5
3
2
1
0
malware
benign

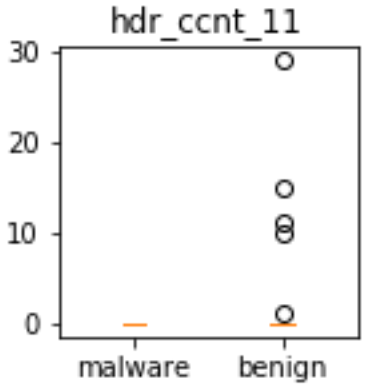

hdr_ccnt_6
6
4
2
0
malware
benign

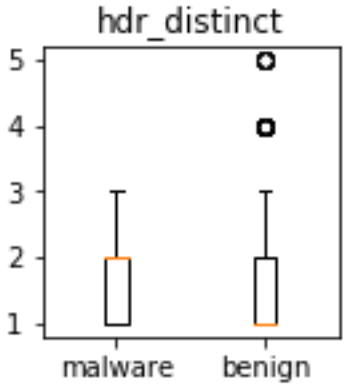

hdr_ccnt_7
40
20
0
malware
benign

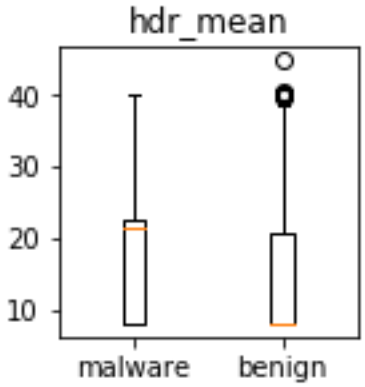

hdr_ccnt_8
10
5
0
malware
benign

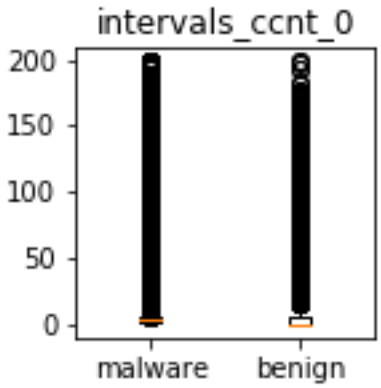

hdr_ccnt_9
30
20
10
0
malware
benign

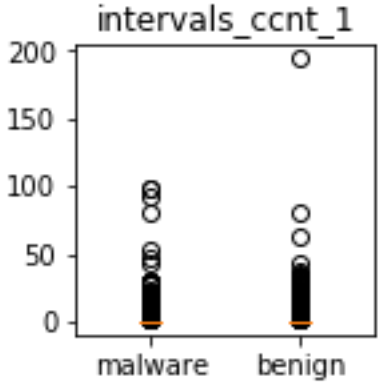

hdr_ccnt_10
0.050
0.025
0.000
−0.025
−0.050
malware
benign

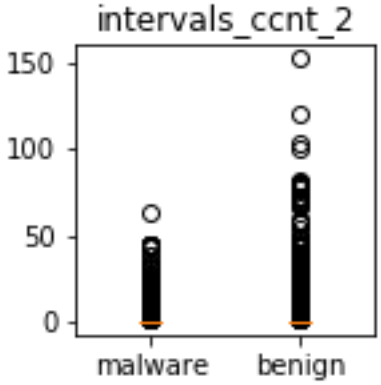

hdr_ccnt_11
30
20
10
0
malware
benign

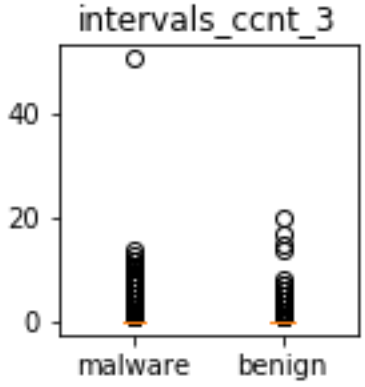

hdr_distinct
5
4
3
2
1
malware
benign

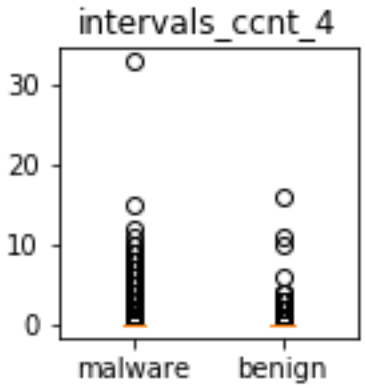

hdr_mean
40
30
20
10
malware
benign

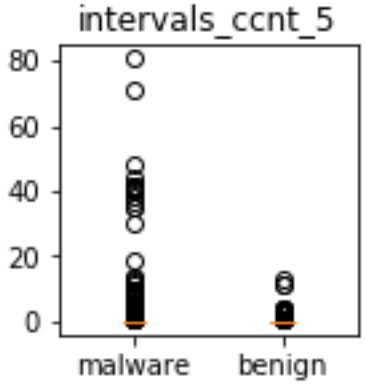

intervals_ccnt_0
200
150
100
50
0
malware
benign

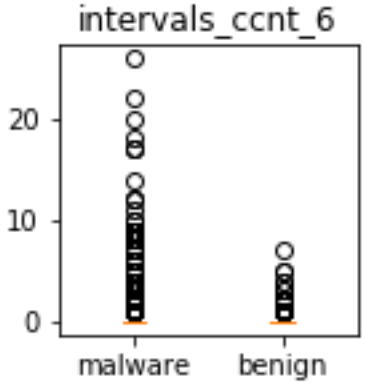

intervals_ccnt_1
200
150
100
50
0
malware
benign

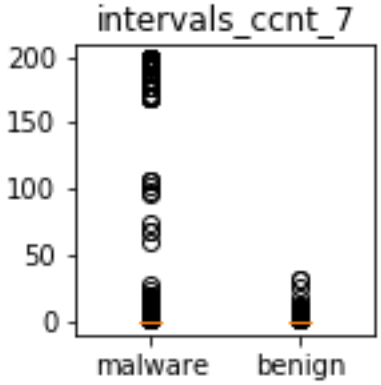

intervals_ccnt_2
150
100
50
0
malware
benign

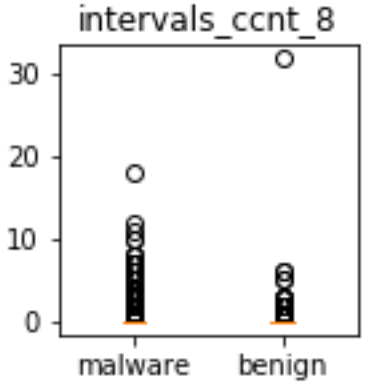

intervals_ccnt_3
40
20
0
malware
benign

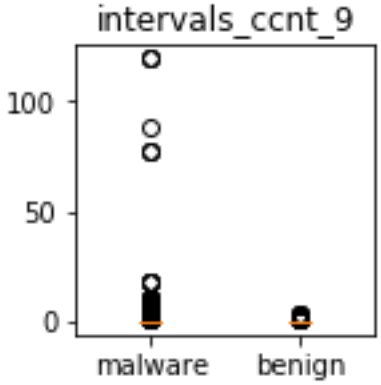

intervals_ccnt_4
30
20
10
0
malware
benign

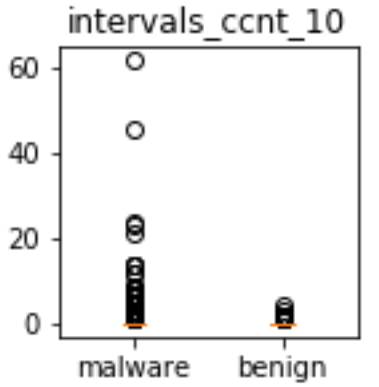

intervals_ccnt_5
80
60
40
20
0
malware
benign

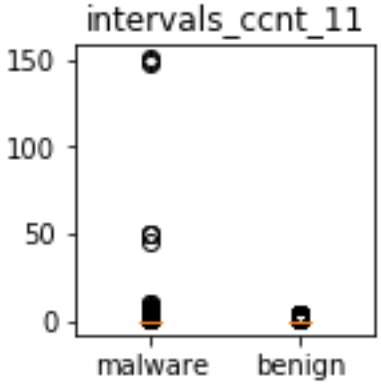

intervals_ccnt_6
20
10
0
malware
benign

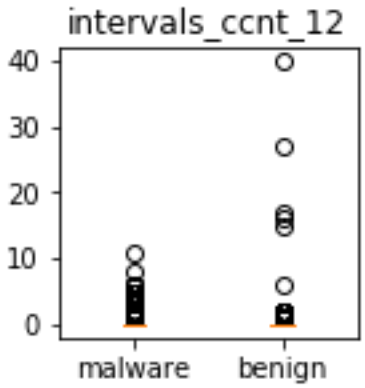

intervals_ccnt_7
200
150
100
50
0
malware
benign

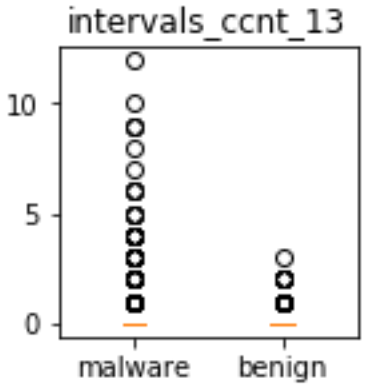

intervals_ccnt_8
30
20
10
0
malware
benign

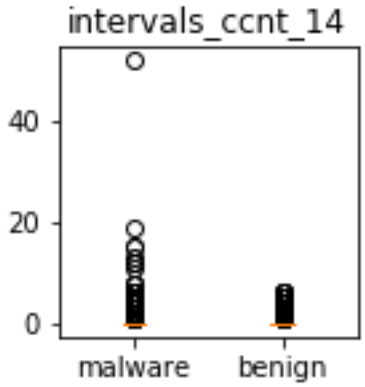

intervals_ccnt_9
100
50
0
malware
benign

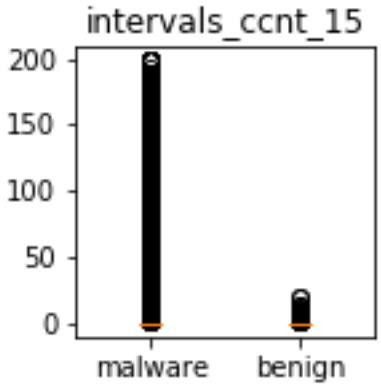

intervals_ccnt_10
60
40
20
0
malware
benign

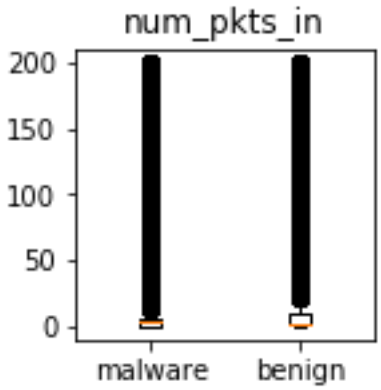

intervals_ccnt_11
150
100
50
0
malware
benign

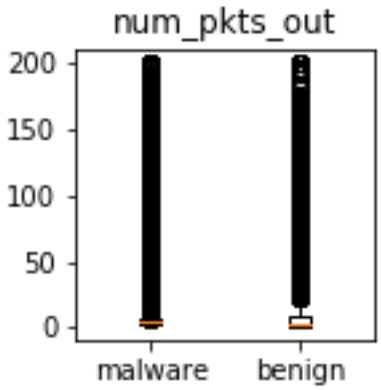

intervals_ccnt_12
40
30
20
10
0
malware
benign

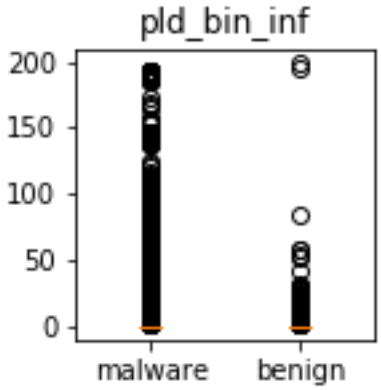

intervals_ccnt_13
10
5
0
malware
benign

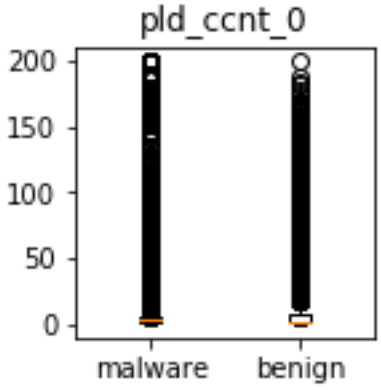

intervals_ccnt_14
40
20
0
malware
benign

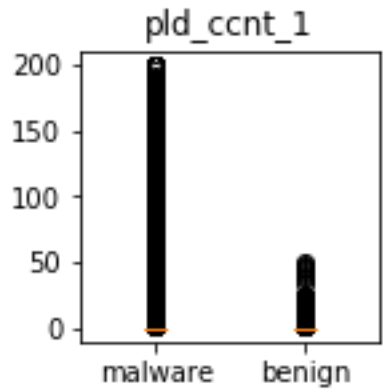

intervals_ccnt_15
200
150
100
50
0
malware
benign

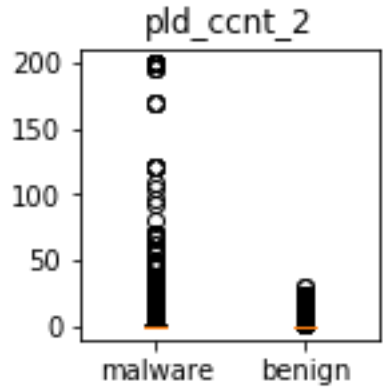

num_pkts_in
200
150
100
50
0
malware
benign

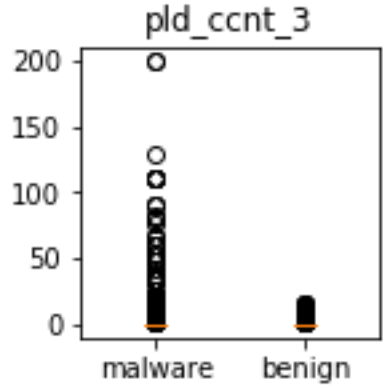

num_pkts_out
200
150
100
50
0
malware
benign

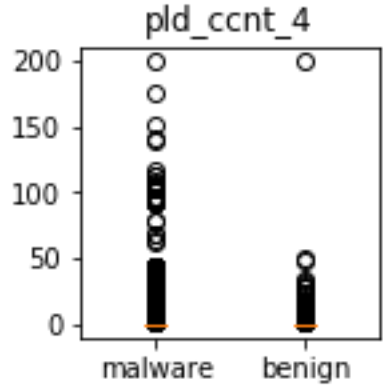

pld_bin_inf
200
150
100
50
0
malware
benign

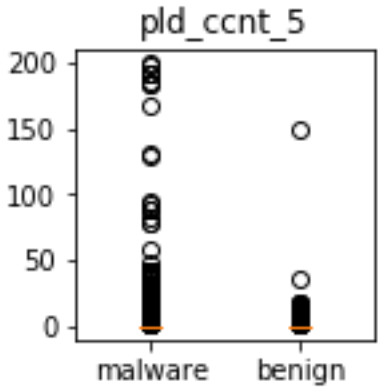

pld_ccnt_0
200
150
100
50
0
malware
benign

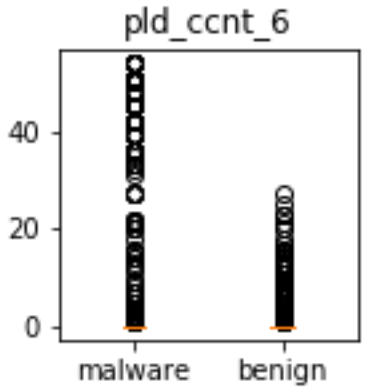

pld_ccnt_1
200
150
100
50
0
malware
benign

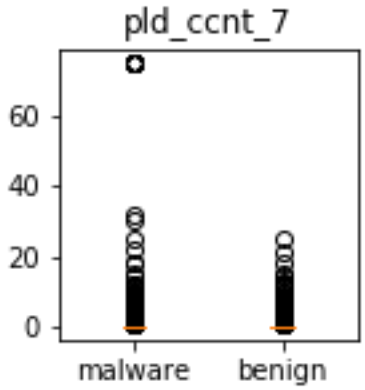

pld_ccnt_2
200
150
100
50
0
malware
benign

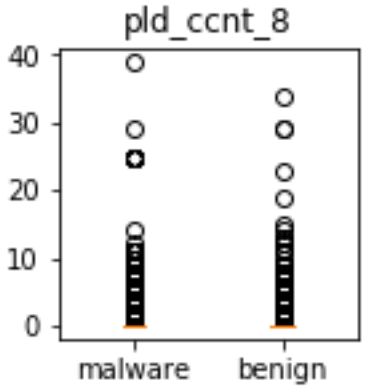

pld_ccnt_3
200
150
100
50
0
malware
benign

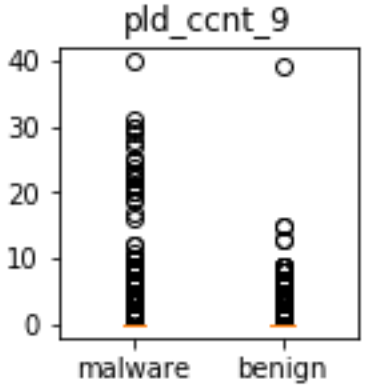

pld_ccnt_4
200
150
100
50
0
malware
benign

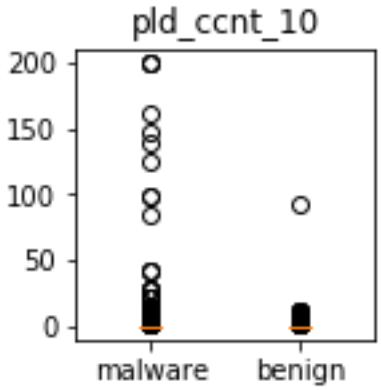

pld_ccnt_5
200
150
100
50
0
malware
benign

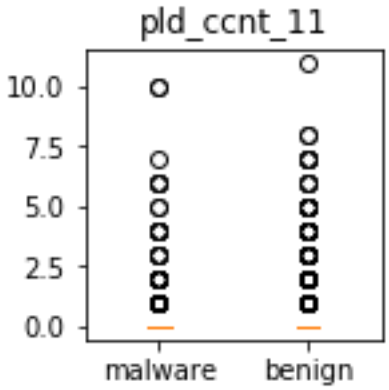

pld_ccnt_6
40
20
0
malware
benign

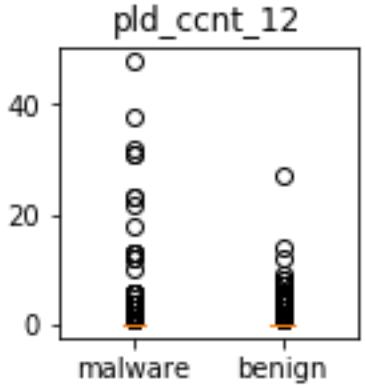

pld_ccnt_7
60
40
20
0
malware
benign

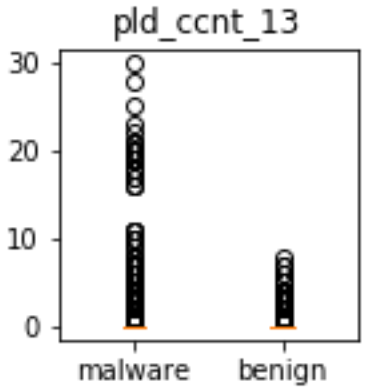

pld_ccnt_8
40
30
20
10
0
malware
benign

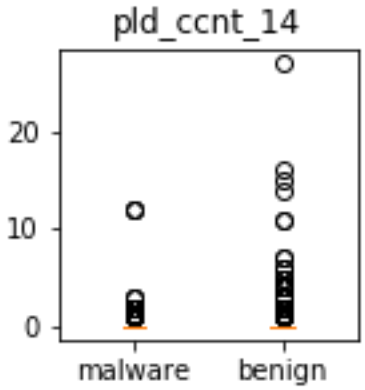

pld_ccnt_9
40
30
20
10
0
malware
benign

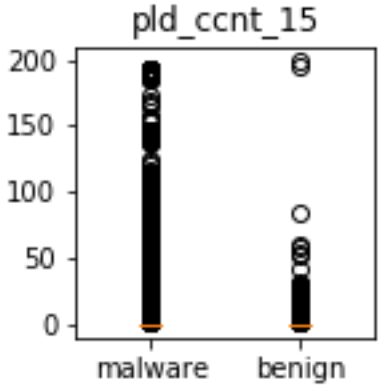

pld_ccnt_10
200
150
100
50
0
malware
benign

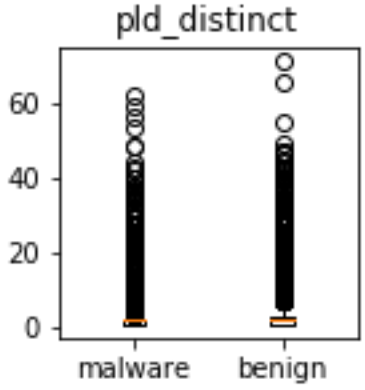

pld_ccnt_11
10.0
7.5
5.0
2.5
0.0
malware
benign

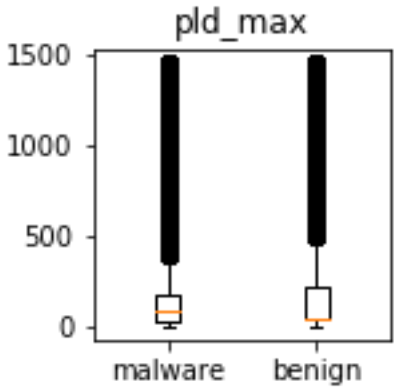

pld_ccnt_12
40
20
0
malware
benign

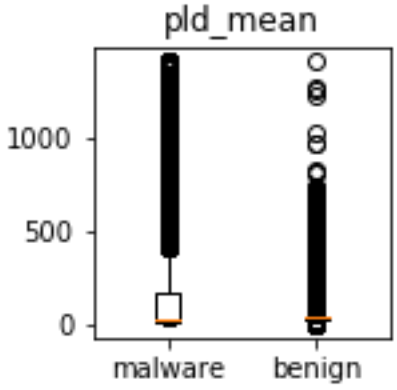

pld_ccnt_13
30
20
10
0
malware
benign

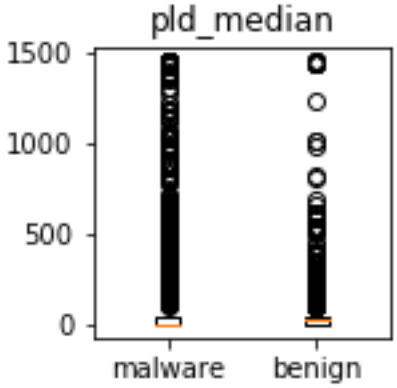

pld_ccnt_14
20
10
0
malware
benign

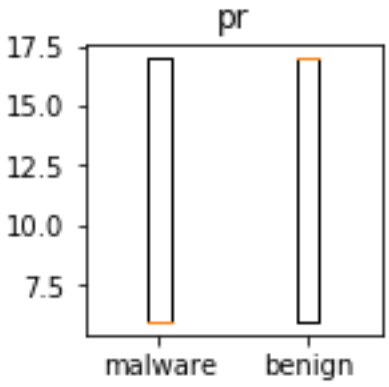

pld_ccnt_15
200
150
100
50
0
malware
benign


pld_distinct
60
40
20
0
malware
benign

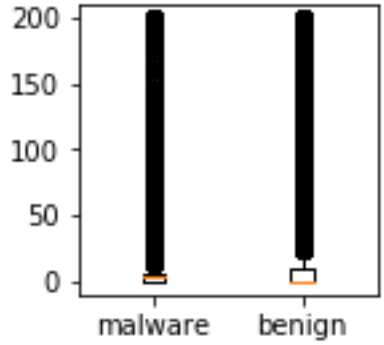

pld_max
1500
1000
500
0
malware
benign


pld_mean
1000
500
0
malware
benign

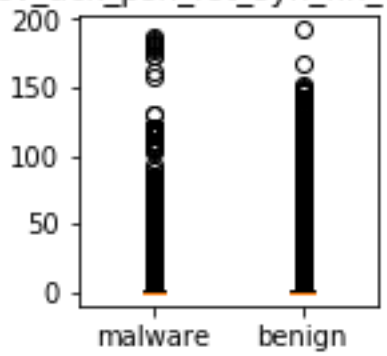

pld_median
1500
1000
500
0
malware
benign


pr
17.5
15.0
12.5
10.0
7.5
malware
benign

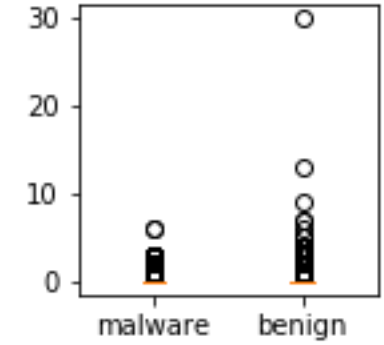
rev_ack_psh_rst_syn_fin_cnt_0



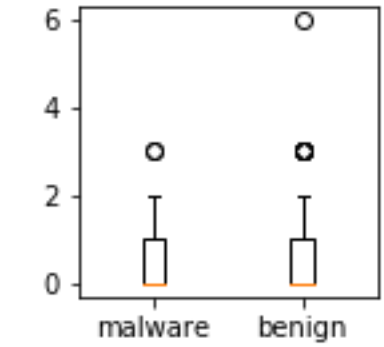
rev_ack_psh_rst_syn_fin_cnt_1

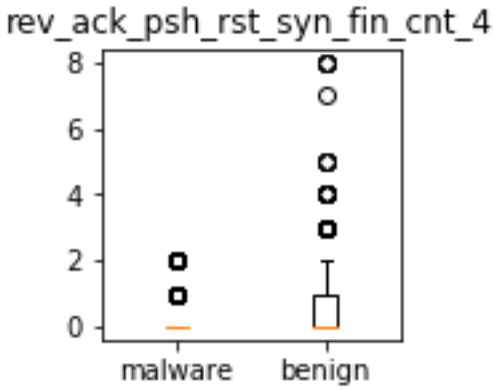


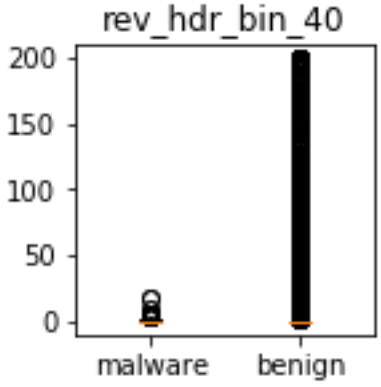
rev_ack_psh_rst_syn_fin_cnt_2

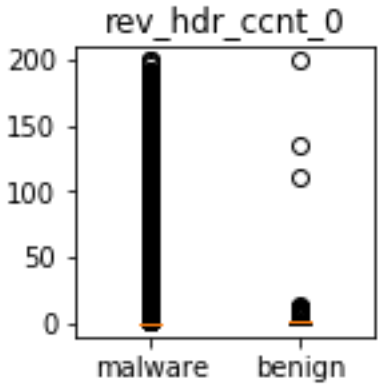


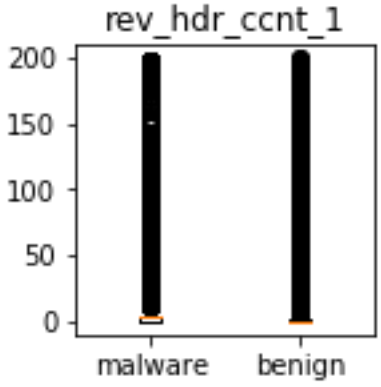
rev_ack_psh_rst_syn_fin_cnt_3

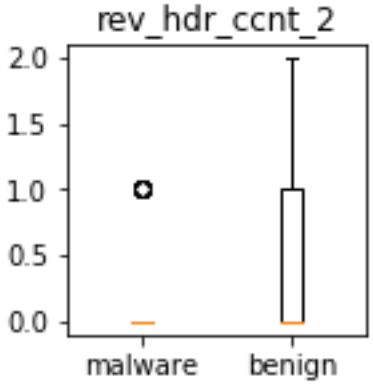


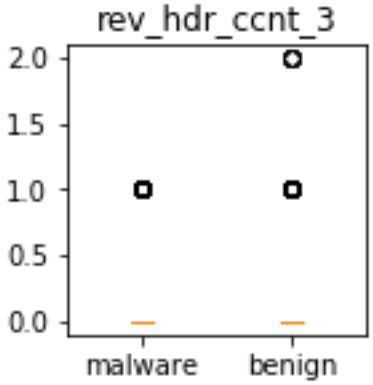

rev_ack_psh_rst_syn_fin_cnt_4
8
6
4
2
0
malware
benign

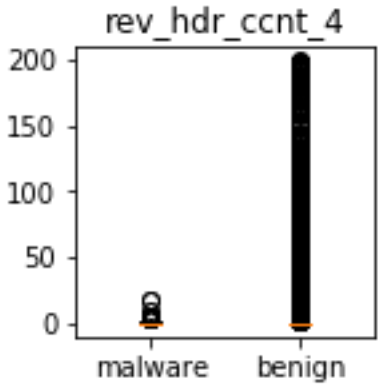

rev_hdr_bin_40
200
150
100
50
0
malware
benign

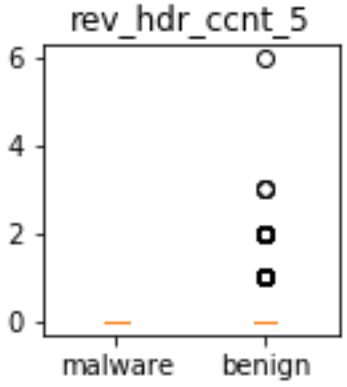

rev_hdr_ccnt_0
200
150
100
50
0
malware
benign

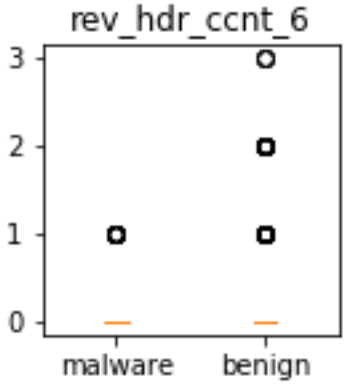

rev_hdr_ccnt_1
200
150
100
50
0
malware
benign

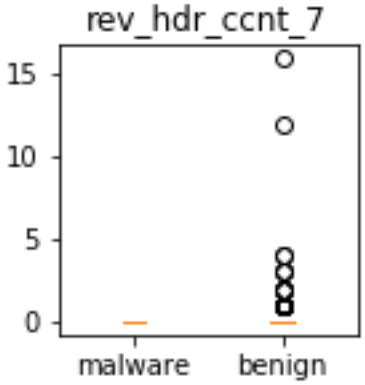

rev_hdr_ccnt_2
2.0
1.5
1.0
0.5
0.0
malware
benign

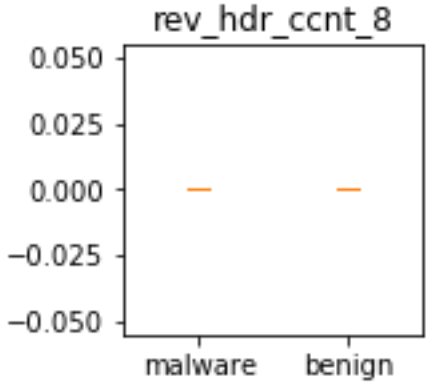

rev_hdr_ccnt_3
2.0
1.5
1.0
0.5
0.0
malware
benign

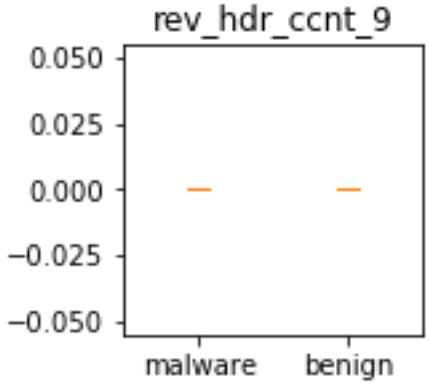

rev_hdr_ccnt_4
200
150
100
50
0
malware
benign

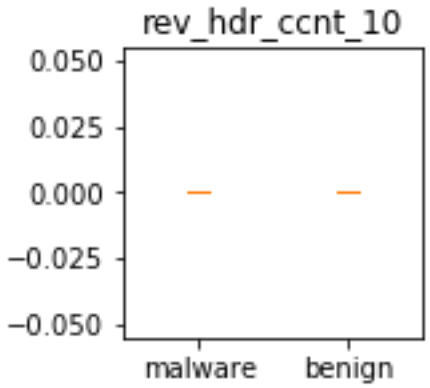

rev_hdr_ccnt_5
6
4
2
0
malware
benign

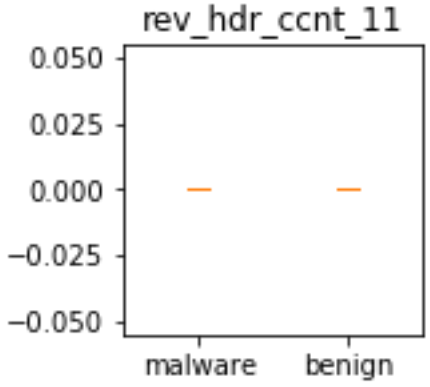

rev_hdr_ccnt_6
3
2
1
0
malware
benign

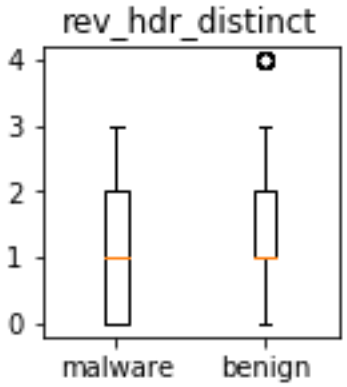

rev_hdr_ccnt_7
15
10
5
0
malware
benign

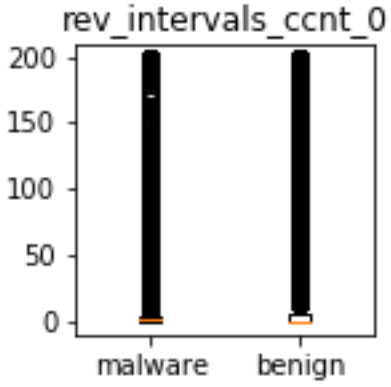

rev_hdr_ccnt_8
0.050
0.025
0.000
−0.025
−0.050
malware
benign

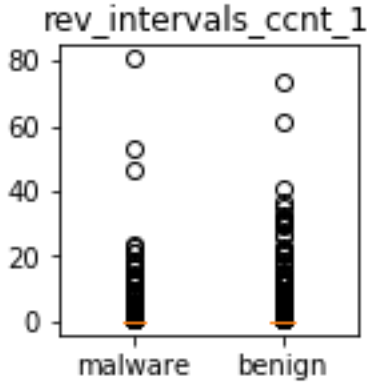

rev_hdr_ccnt_9
0.050
0.025
0.000
−0.025
−0.050
malware
benign

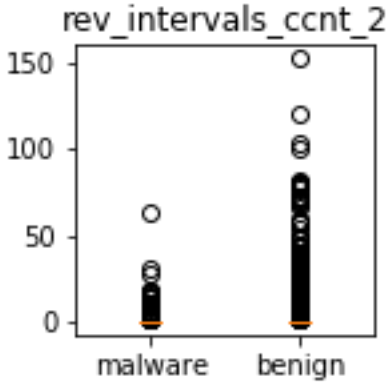

rev_hdr_ccnt_10
0.050
0.025
0.000
−0.025
−0.050
malware
benign

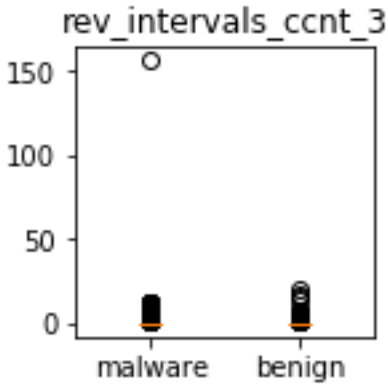

rev_hdr_ccnt_11
0.050
0.025
0.000
−0.025
−0.050
malware
benign

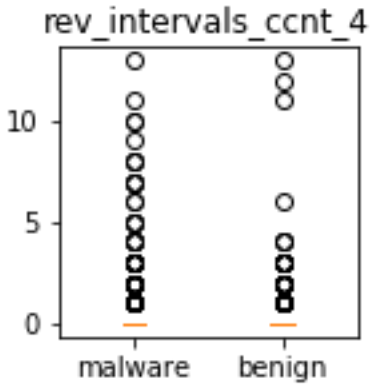

rev_hdr_distinct
4
3
2
1
0
malware
benign

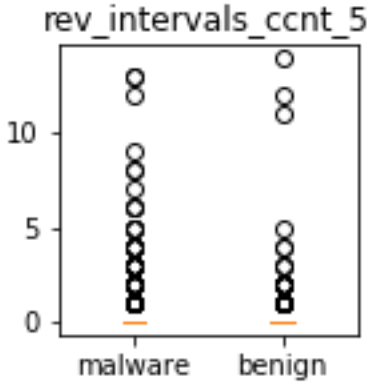

rev_intervals_ccnt_0
200
150
100
50
0
malware
benign

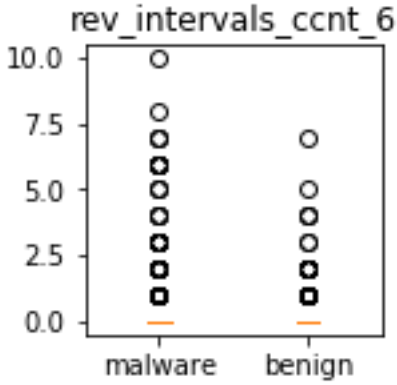

rev_intervals_ccnt_1
80
60
40
20
0
malware
benign

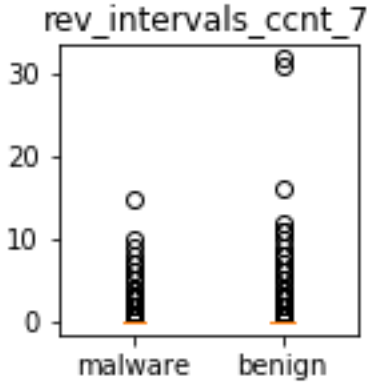

rev_intervals_ccnt_2
150
100
50
0
malware
benign

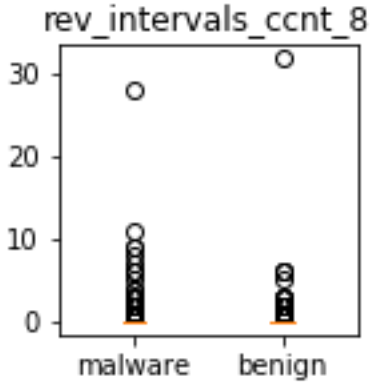

rev_intervals_ccnt_3
150
100
50
0
malware
benign

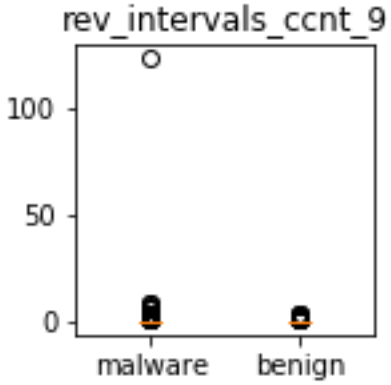

rev_intervals_ccnt_4
10
5
0
malware
benign

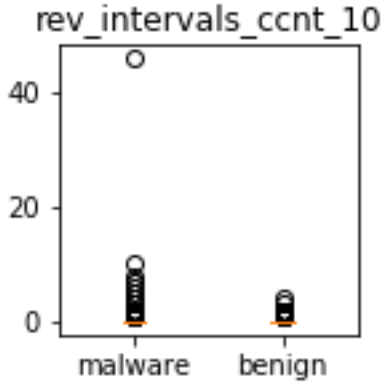

rev_intervals_ccnt_5
10
5
0
malware
benign

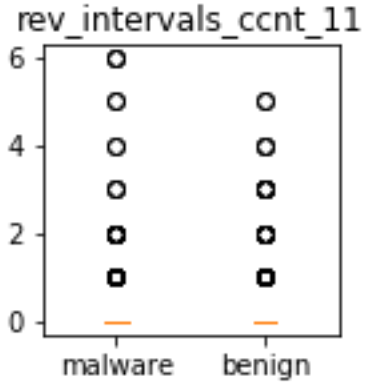

rev_intervals_ccnt_6
10.0
7.5
5.0
2.5
0.0
malware
benign

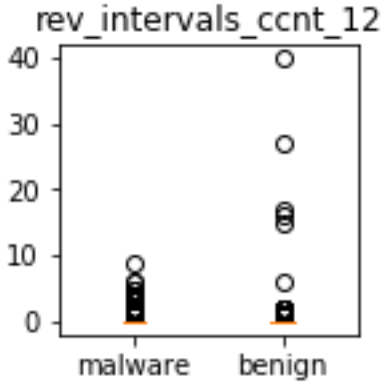

rev_intervals_ccnt_7
30
20
10
0
malware
benign

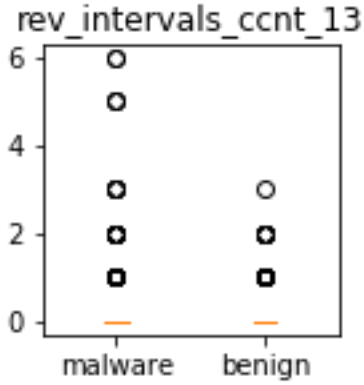

rev_intervals_ccnt_8
30
20
10
0
malware
benign

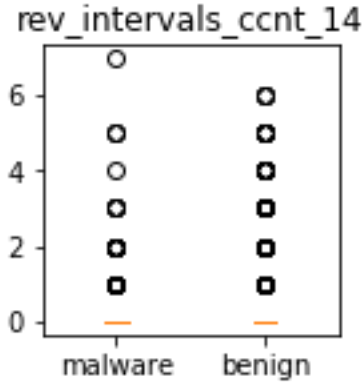

rev_intervals_ccnt_9
100
50
0
malware
benign

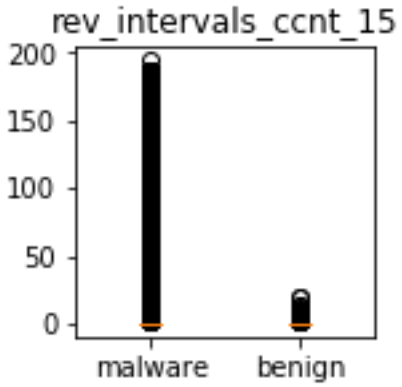

rev_intervals_ccnt_10
40
20
0
malware
benign

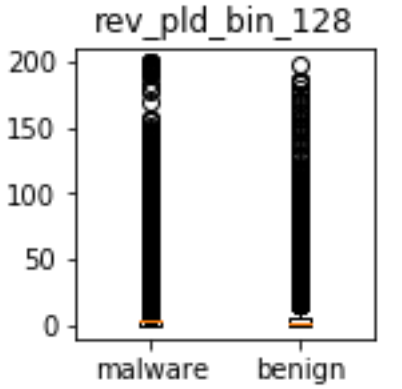

rev_intervals_ccnt_11
6
4
2
0
malware
benign

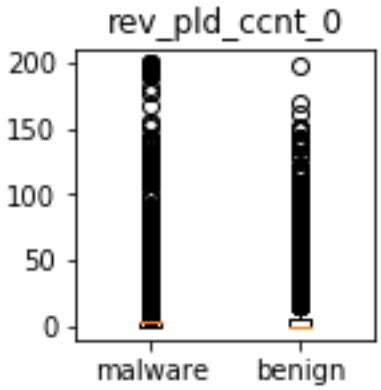

rev_intervals_ccnt_12
40
30
20
10
0
malware
benign

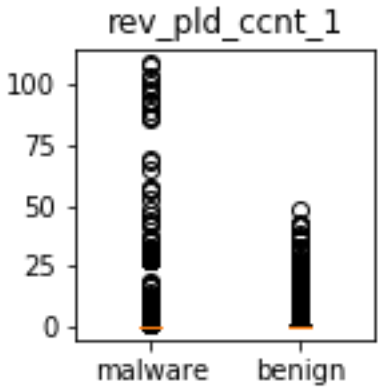

rev_intervals_ccnt_13
6
4
2
0
malware
benign

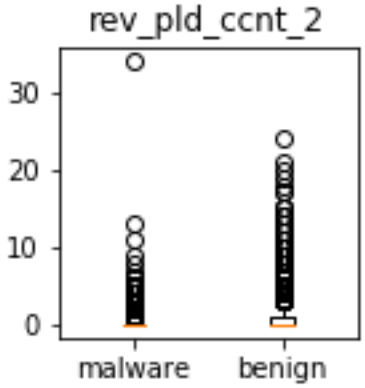

rev_intervals_ccnt_14
6
4
2
0
malware
benign

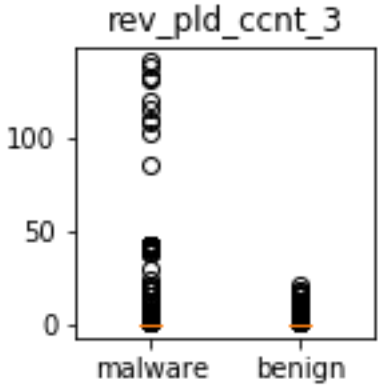

rev_intervals_ccnt_15
200
150
100
50
0
malware
benign

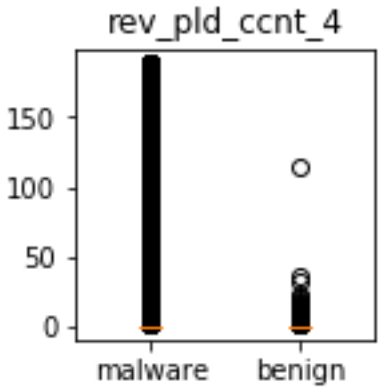

rev_pld_bin_128
200
150
100
50
0
malware
benign

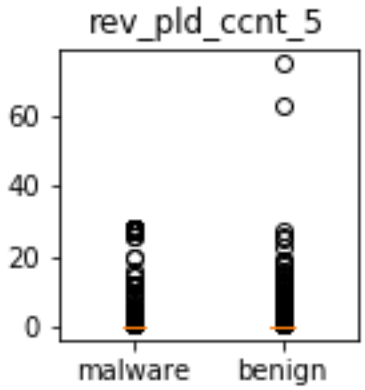

rev_pld_ccnt_0
200
150
100
50
0
malware
benign

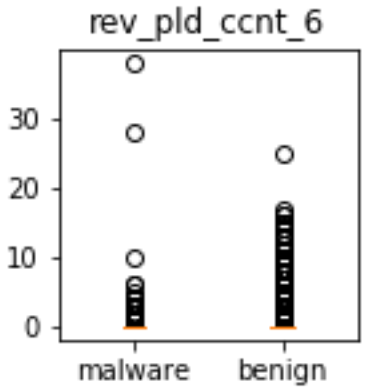

rev_pld_ccnt_1
100
75
50
25
0
malware
benign

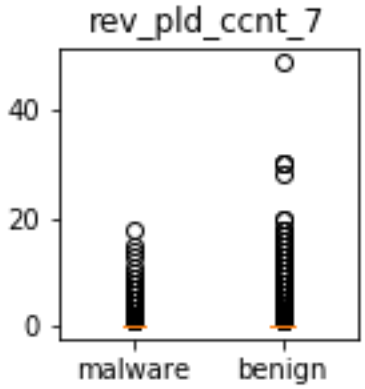

rev_pld_ccnt_2
30
20
10
0
malware
benign

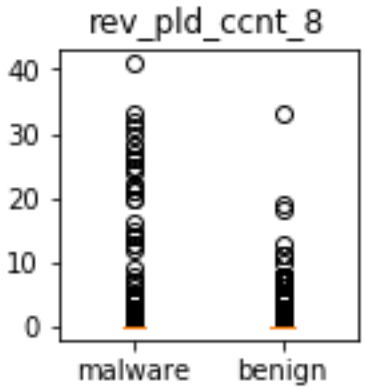

rev_pld_ccnt_3
100
50
0
malware
benign

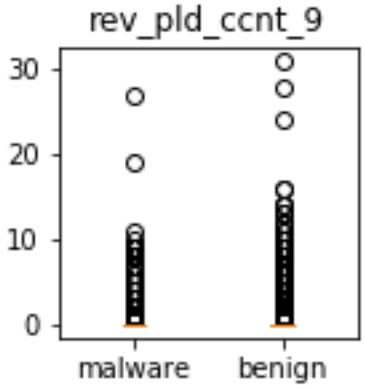

rev_pld_ccnt_4
150
100
50
0
malware
benign

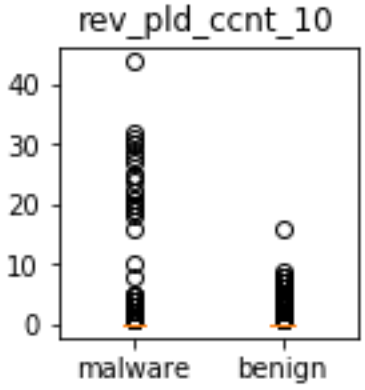

rev_pld_ccnt_5
60
40
20
0
malware
benign

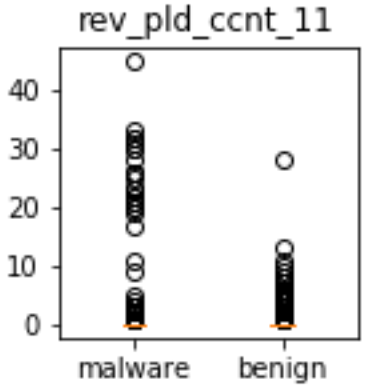

rev_pld_ccnt_6
30
20
10
0
malware
benign

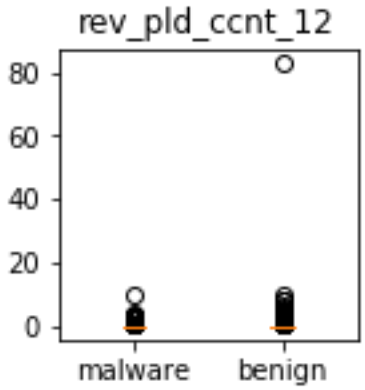

rev_pld_ccnt_7
40
20
0
malware
benign

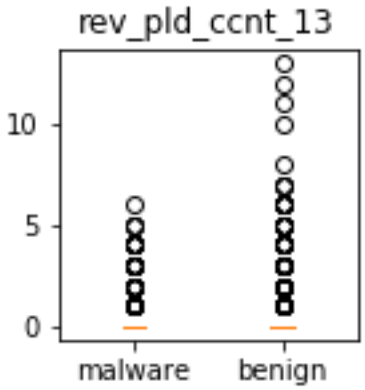

rev_pld_ccnt_8
40
30
20
10
0
malware
benign

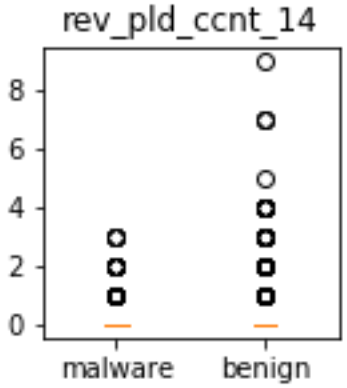

rev_pld_ccnt_9
30
20
10
0
malware
benign

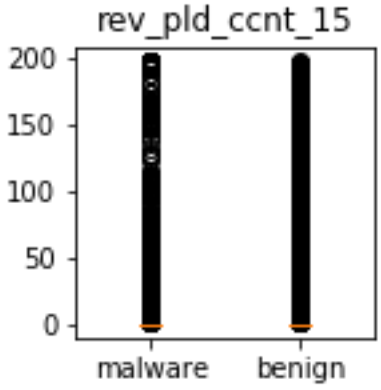

rev_pld_ccnt_10
40
30
20
10
0
malware
benign

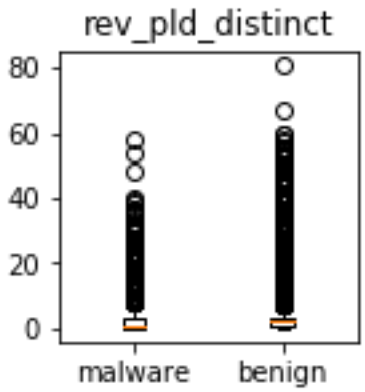

rev_pld_ccnt_11
40
30
20
10
0
malware
benign

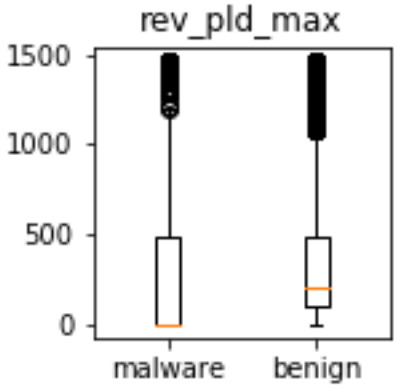

rev_pld_ccnt_12
80
60
40
20
0
malware
benign

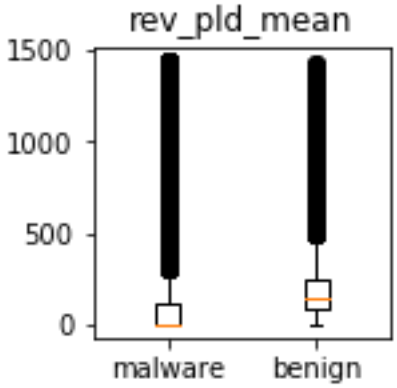

rev_pld_ccnt_13
10
5
0
malware
benign

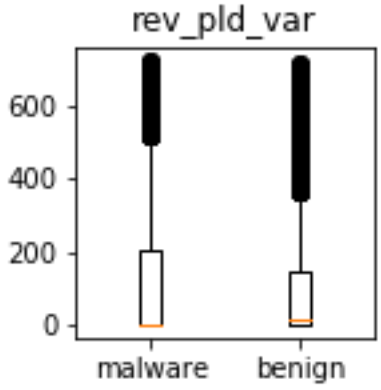

rev_pld_ccnt_14
8
6
4
2
0
malware
benign

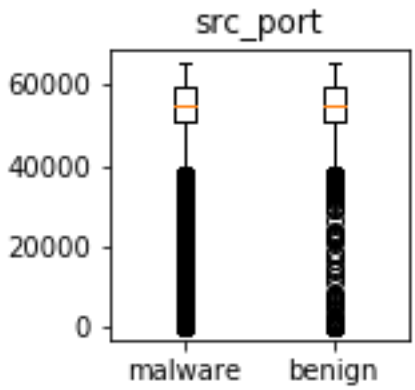

rev_pld_ccnt_15
200
150
100
50
0
malware
benign

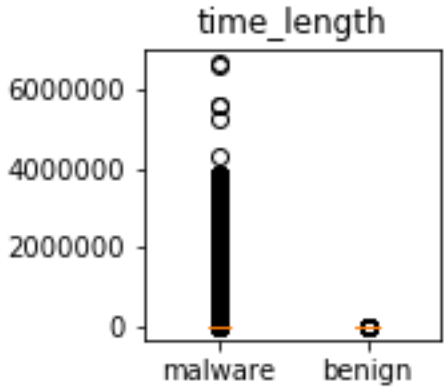

rev_pld_distinct
80
60
40
20
0
malware
benign

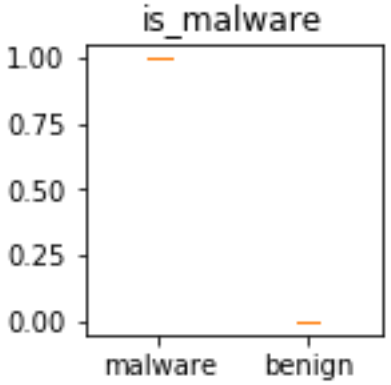

rev_pld_max
1500
1000
500
0
malware
benign


rev_pld_mean
1500
1000
500
0
malware
benign

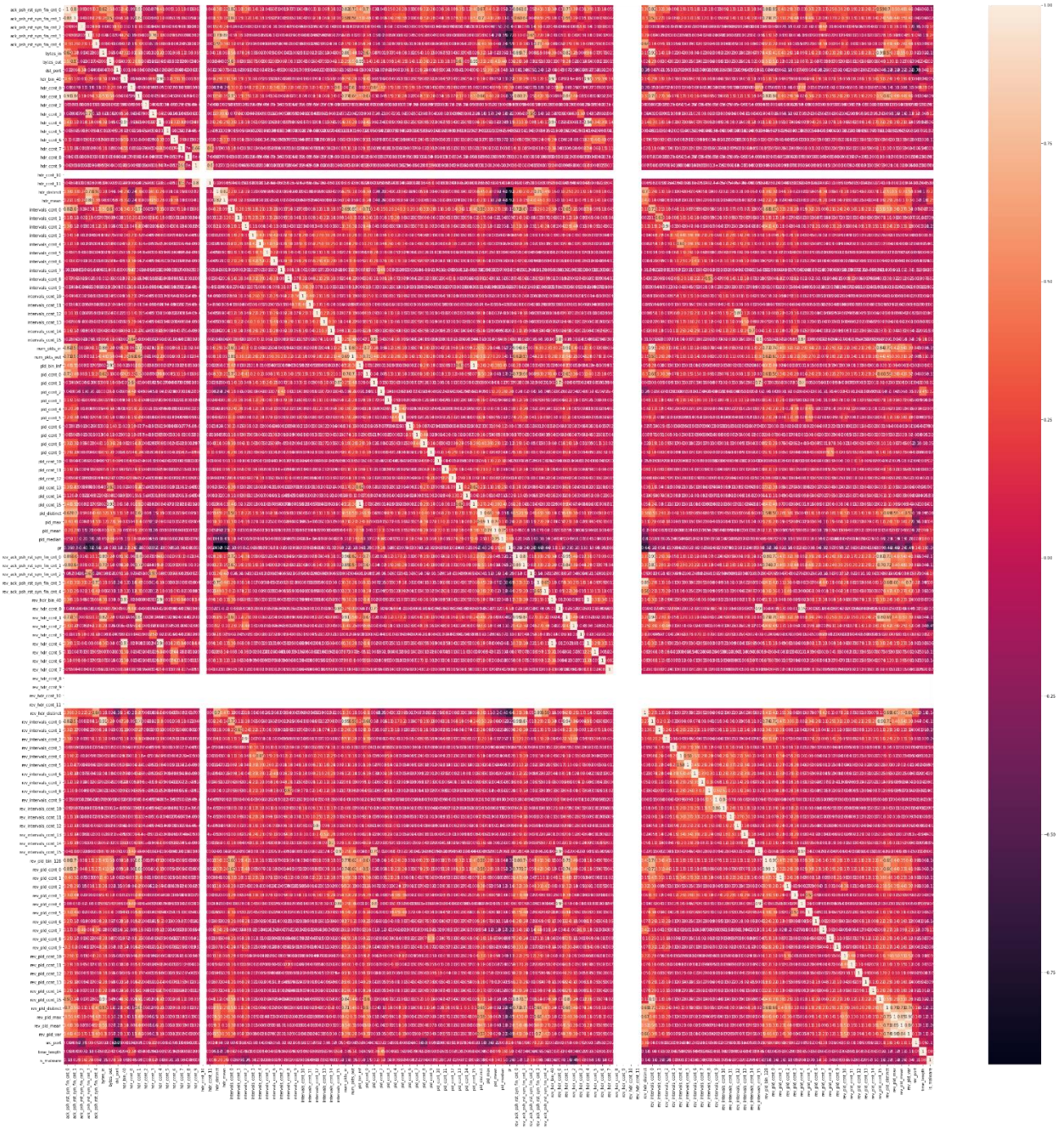

rev_pld_var
600
400
200
0
malware
benign


src_port
60000
40000
20000
0
malware
benign


time_length
6000000
4000000
2000000
0
malware
benign

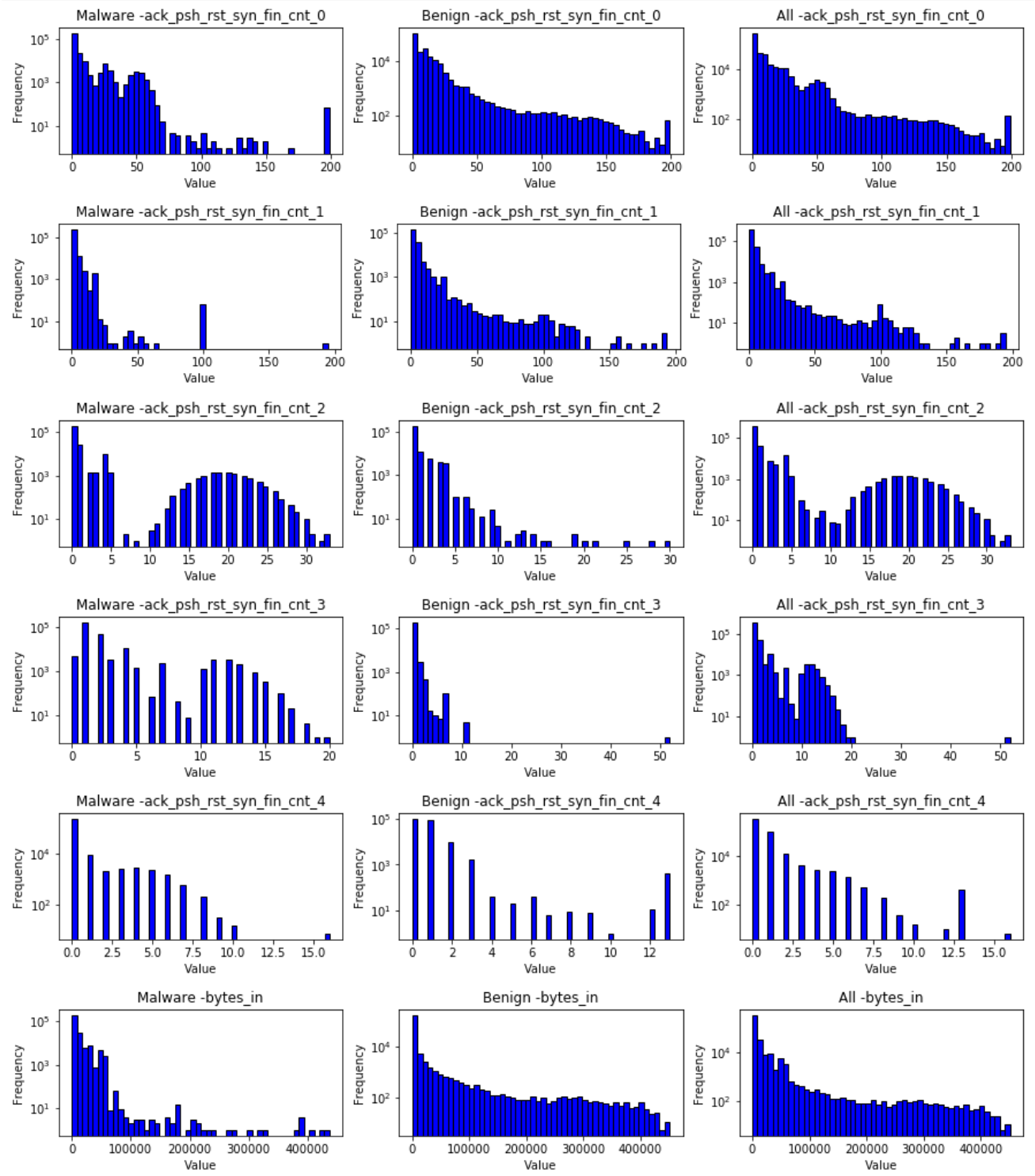

is_malware
1.00
0.75
0.50
0.25
0.00
malware
benign

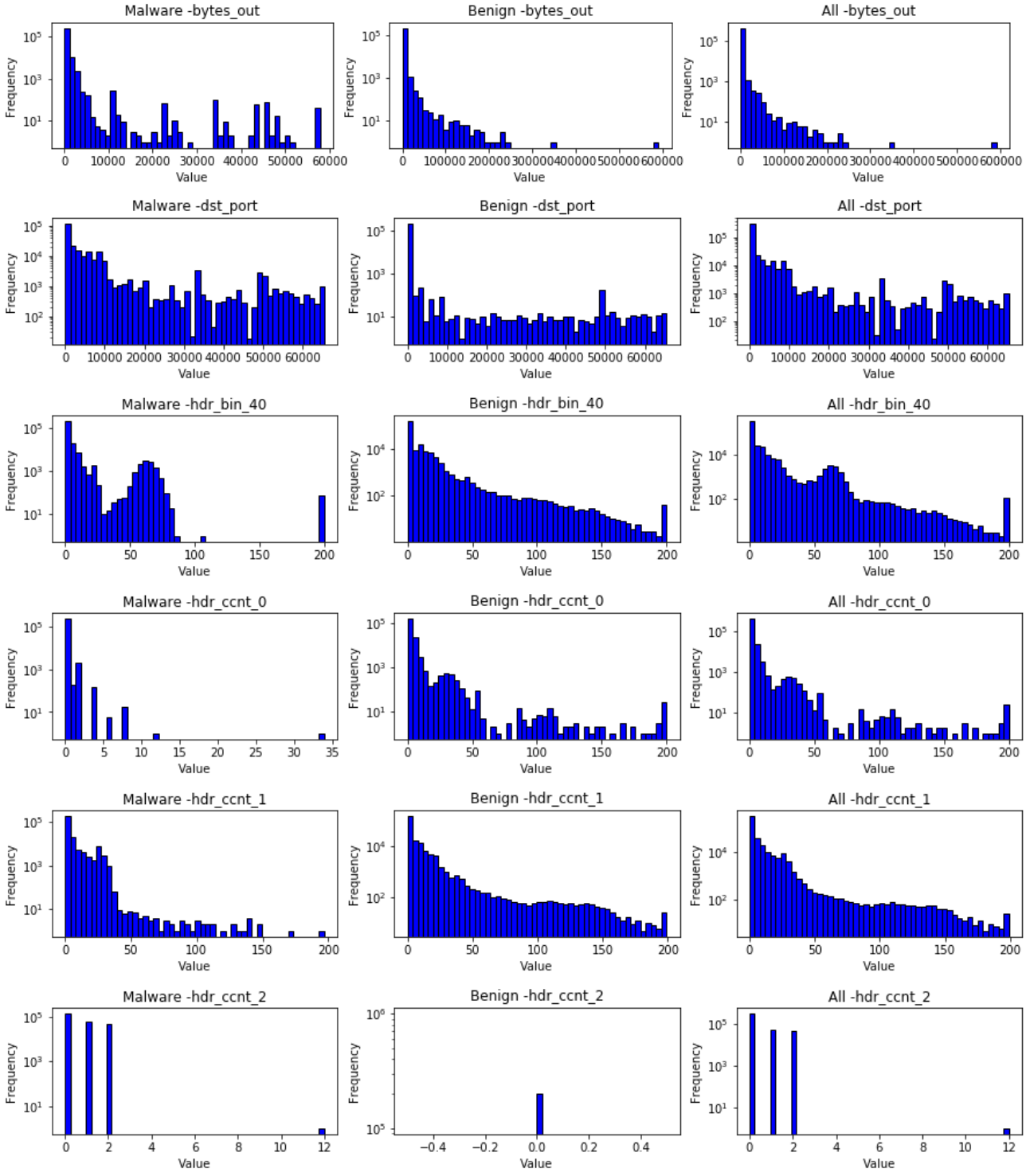
Bivariate Analysis

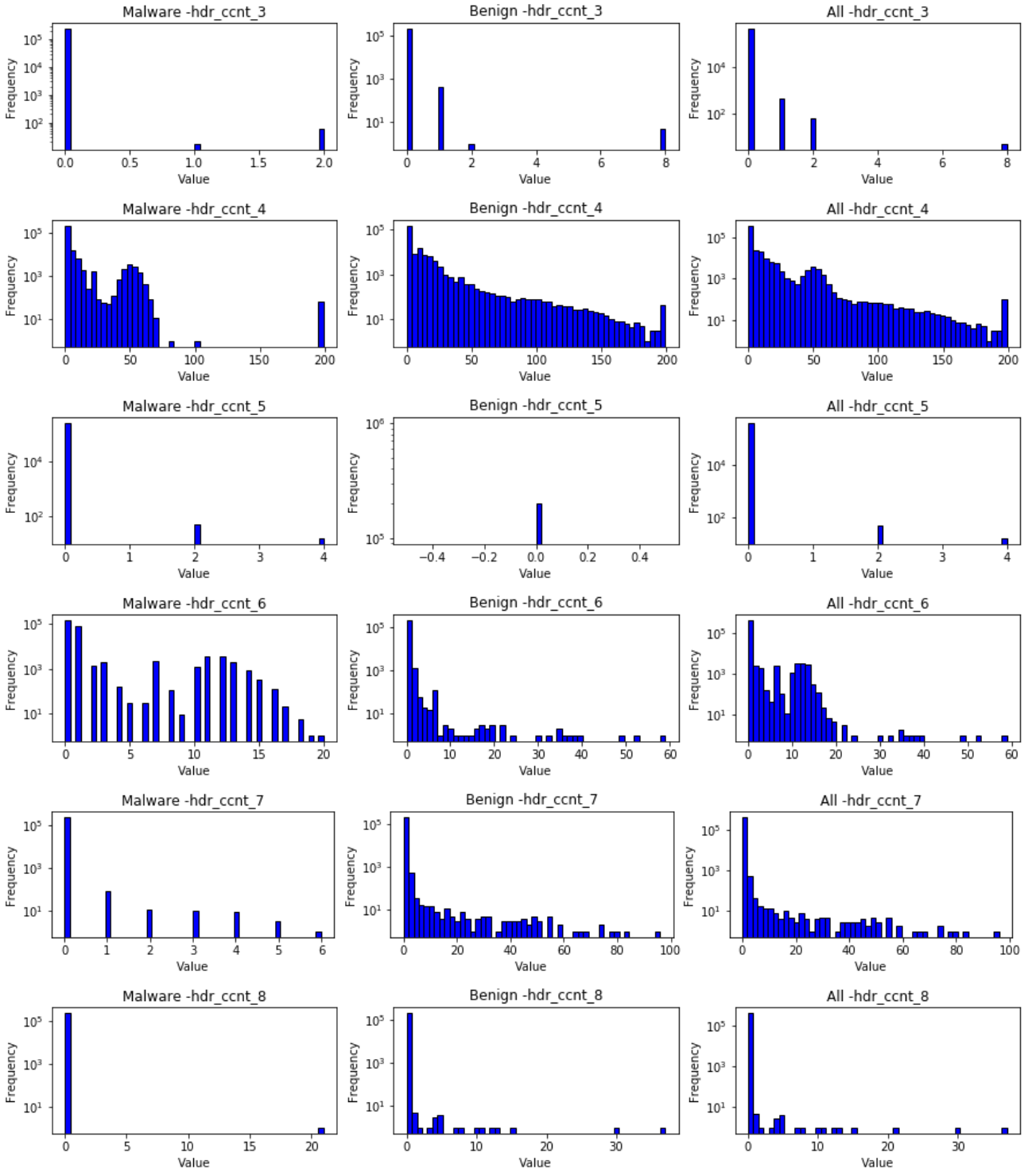


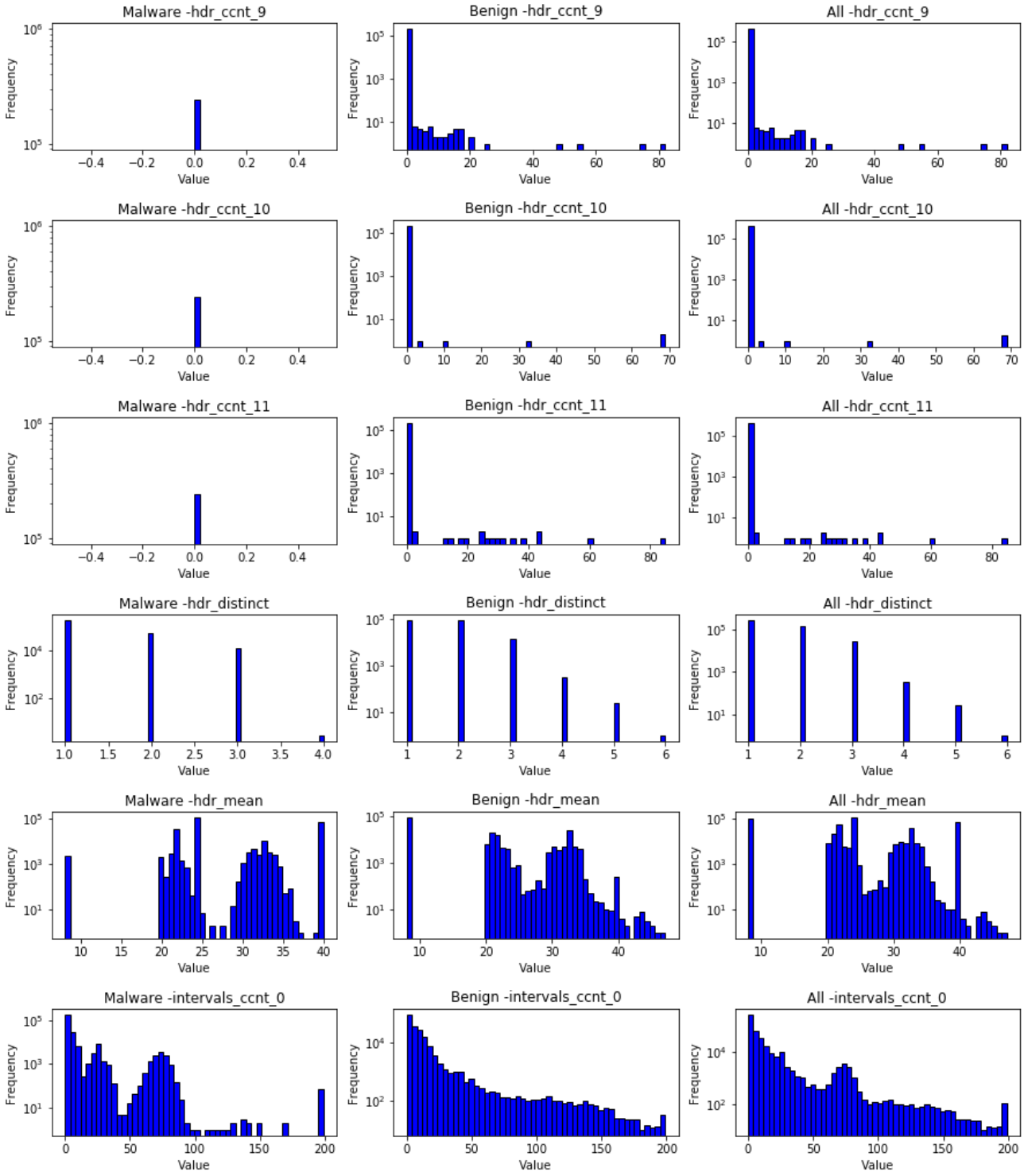
CICIDS Dataset

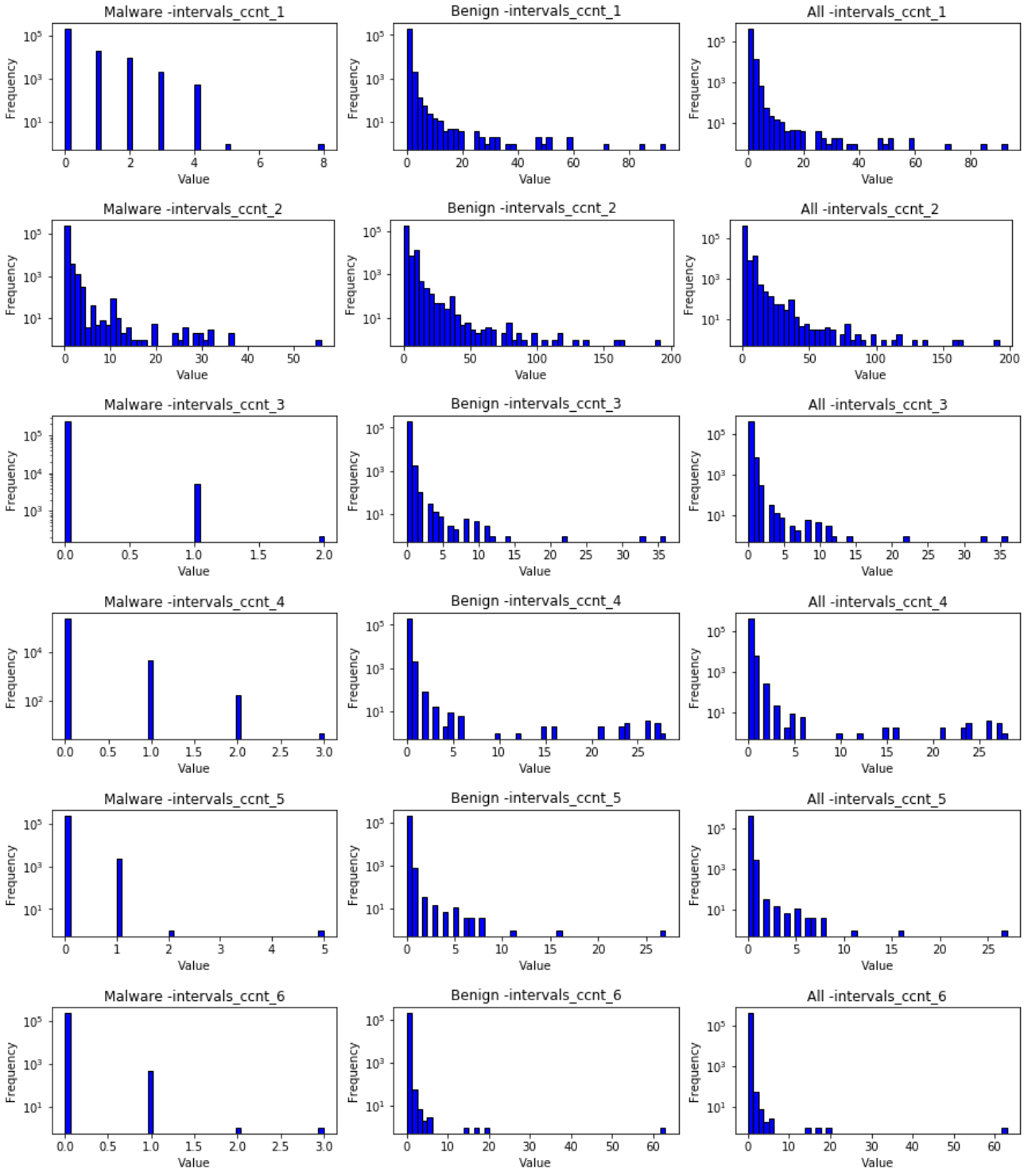
Univariate Analysis – Distribution Plots

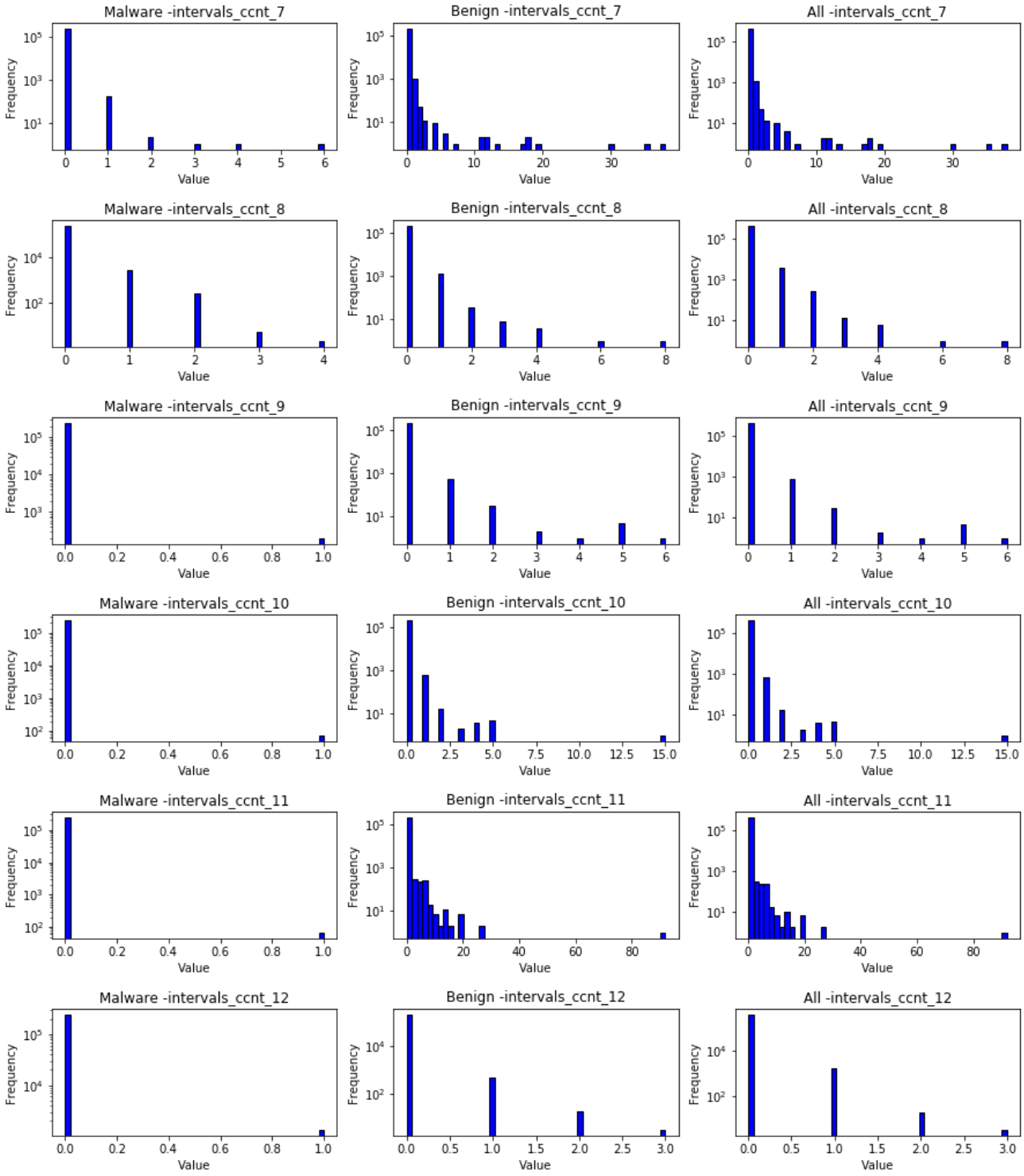


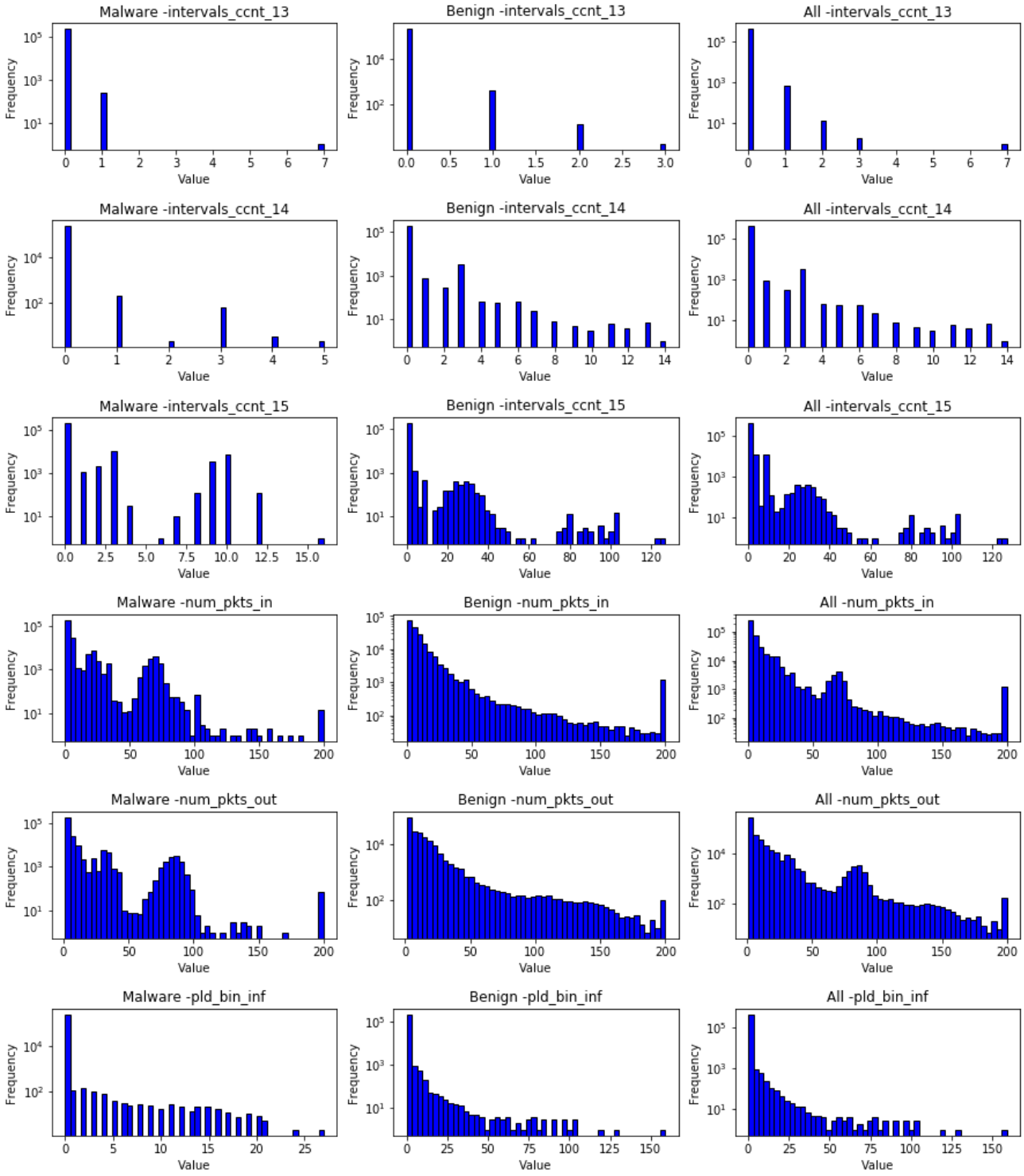

Malware -bytes_out
Benign -bytes_out
All -bytes_out
Malware -dst_port
Benign -dst_port
All -dst_port
Malware -hdr_bin_40
Benign -hdr_bin_40
All -hdr_bin_40
Malware -hdr_ccnt_0
Benign -hdr_ccnt_0
All -hdr_ccnt_0
Malware -hdr_ccnt_1
Benign -hdr_ccnt_1
All -hdr_ccnt_1
Malware -hdr_ccnt_2
Benign -hdr_ccnt_2
All -hdr_ccnt_2
Frequency
Value

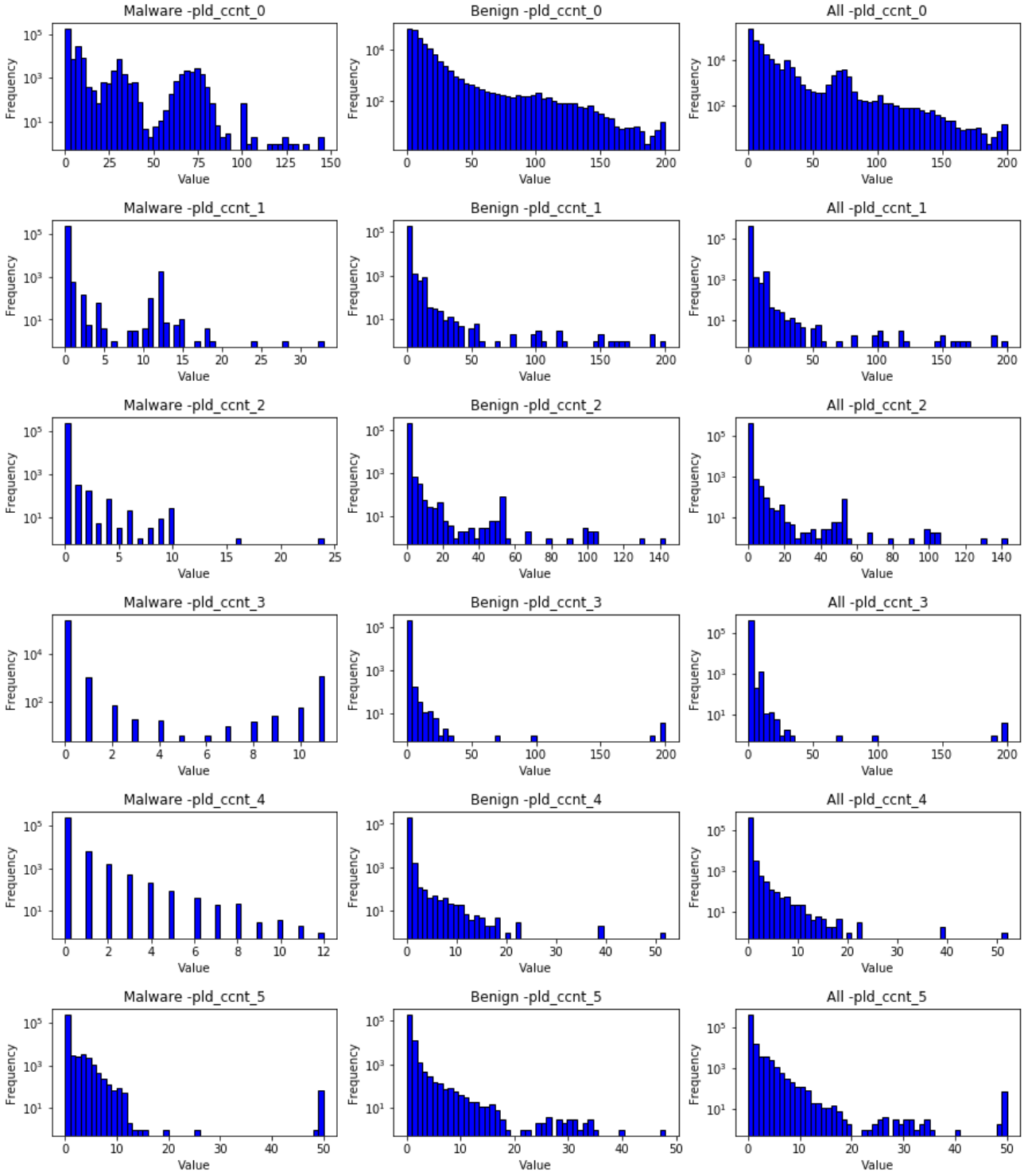

Malware -hdr_ccnt_3
Benign -hdr_ccnt_3
All -hdr_ccnt_3
Malware -hdr_ccnt_4
Benign -hdr_ccnt_4
All -hdr_ccnt_4
Malware -hdr_ccnt_5
Benign -hdr_ccnt_5
All -hdr_ccnt_5
Malware -hdr_ccnt_6
Benign -hdr_ccnt_6
All -hdr_ccnt_6
Malware -hdr_ccnt_7
Benign -hdr_ccnt_7
All -hdr_ccnt_7
Malware -hdr_ccnt_8
Benign -hdr_ccnt_8
All -hdr_ccnt_8
Frequency
Value

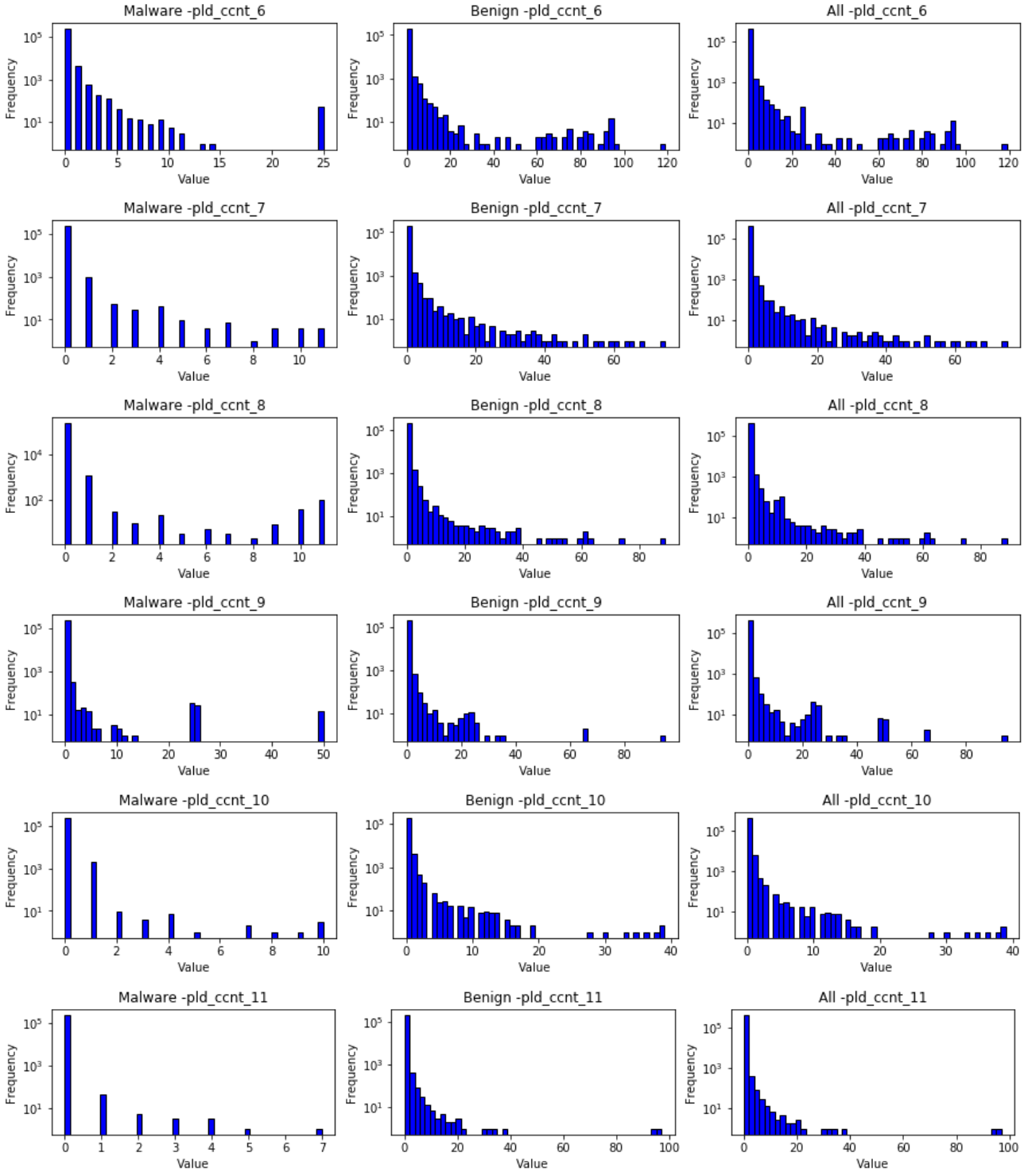

Malware -hdr_ccnt_9
Benign -hdr_ccnt_9
All -hdr_ccnt_9
Malware -hdr_ccnt_10
Benign -hdr_ccnt_10
All -hdr_ccnt_10
Malware -hdr_ccnt_11
Benign -hdr_ccnt_11
All -hdr_ccnt_11
Malware -hdr_distinct
Benign -hdr_distinct
All -hdr_distinct
Malware -hdr_mean
Benign -hdr_mean
All -hdr_mean
Malware -intervals_ccnt_0
Benign -intervals_ccnt_0
All -intervals_ccnt_0
Frequency
Value

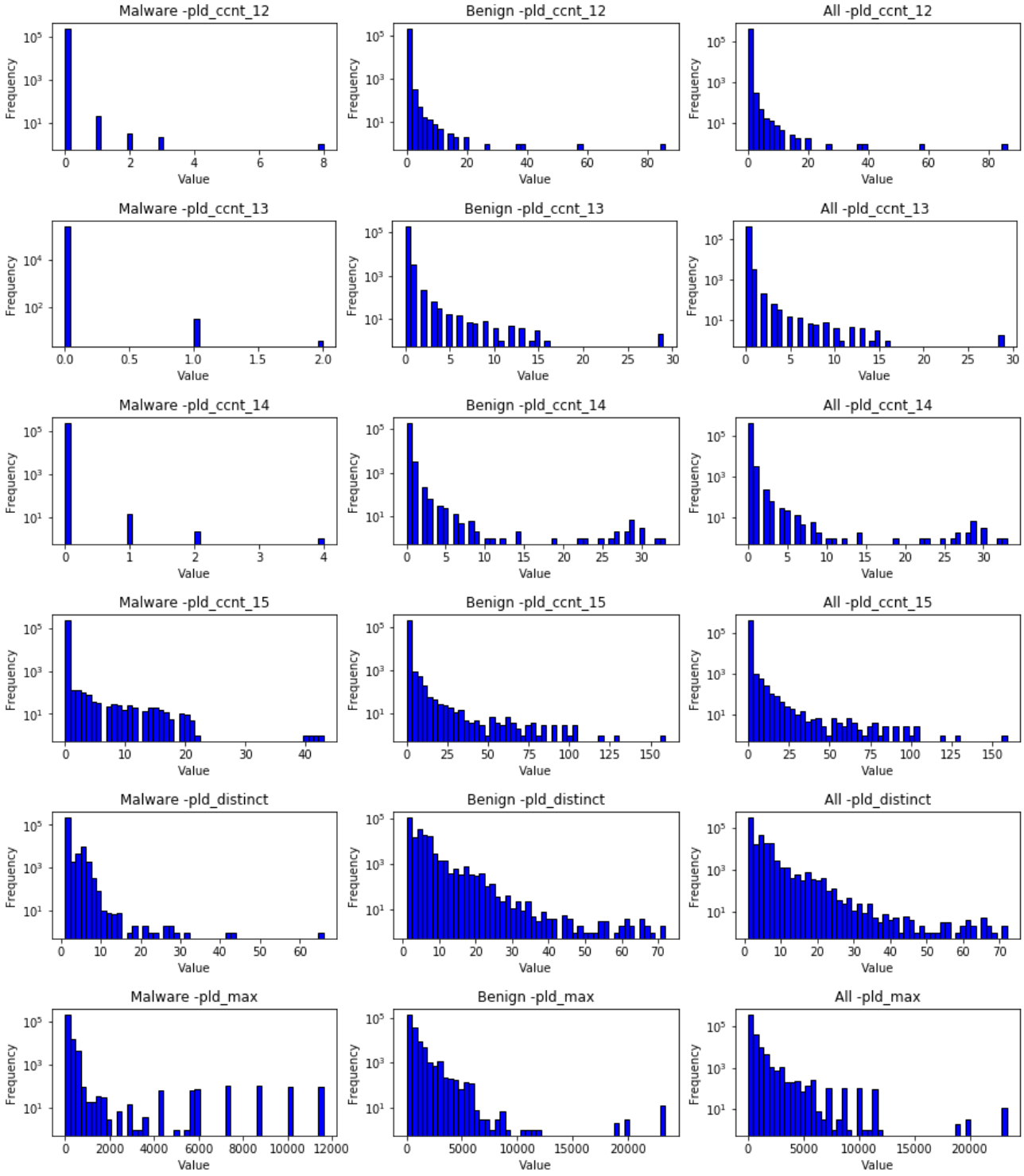

Malware -intervals_ccnt_1
Benign -intervals_ccnt_1
All -intervals_ccnt_1
Malware -intervals_ccnt_2
Benign -intervals_ccnt_2
All -intervals_ccnt_2
Malware -intervals_ccnt_3
Benign -intervals_ccnt_3
All -intervals_ccnt_3
Malware -intervals_ccnt_4
Benign -intervals_ccnt_4
All -intervals_ccnt_4
Malware -intervals_ccnt_5
Benign -intervals_ccnt_5
All -intervals_ccnt_5
Malware -intervals_ccnt_6
Benign -intervals_ccnt_6
All -intervals_ccnt_6
Frequency
Value

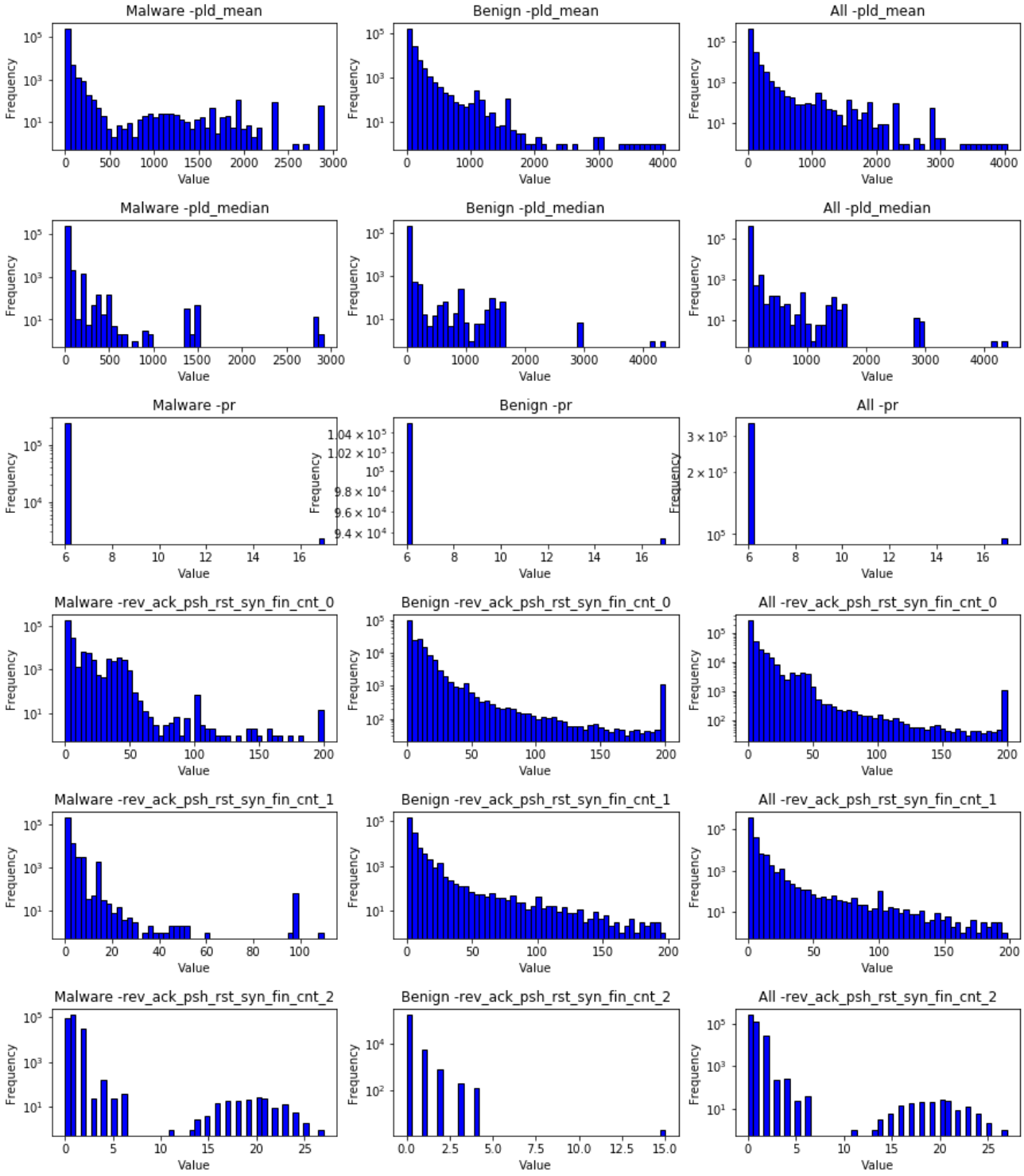

Malware -intervals_ccnt_7
Benign -intervals_ccnt_7
All -intervals_ccnt_7
Malware -intervals_ccnt_8
Benign -intervals_ccnt_8
All -intervals_ccnt_8
Malware -intervals_ccnt_9
Benign -intervals_ccnt_9
All -intervals_ccnt_9
Malware -intervals_ccnt_10
Benign -intervals_ccnt_10
All -intervals_ccnt_10
Malware -intervals_ccnt_11
Benign -intervals_ccnt_11
All -intervals_ccnt_11
Malware -intervals_ccnt_12
Benign -intervals_ccnt_12
All -intervals_ccnt_12
Frequency
Value

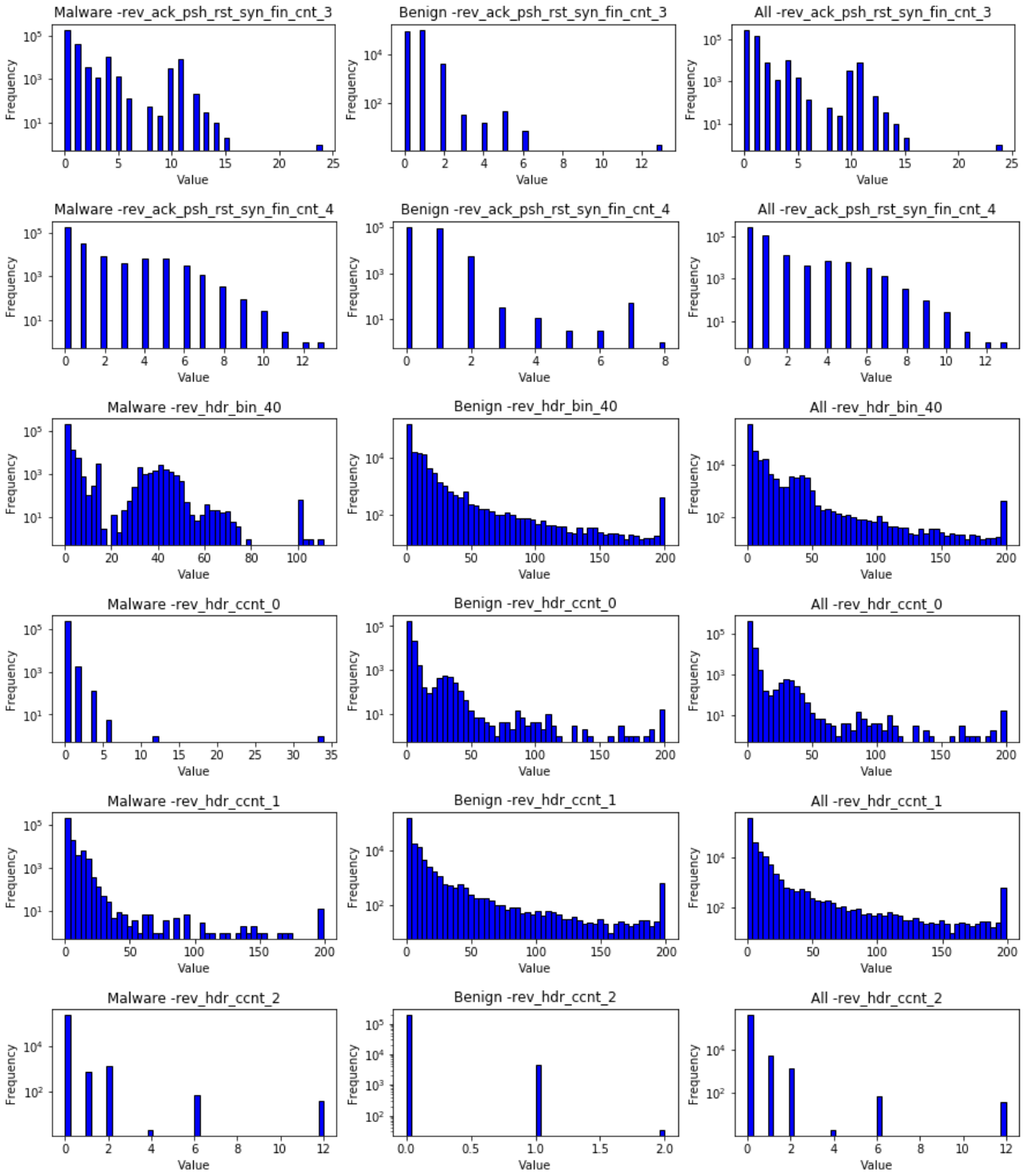

Malware -intervals_ccnt_13
Benign -intervals_ccnt_13
All -intervals_ccnt_13
Malware -intervals_ccnt_14
Benign -intervals_ccnt_14
All -intervals_ccnt_14
Malware -intervals_ccnt_15
Benign -intervals_ccnt_15
All -intervals_ccnt_15
Malware -num_pkts_in
Benign -num_pkts_in
All -num_pkts_in
Malware -num_pkts_out
Benign -num_pkts_out
All -num_pkts_out
Malware -pld_bin_inf
Benign -pld_bin_inf
All -pld_bin_inf
Frequency
Value

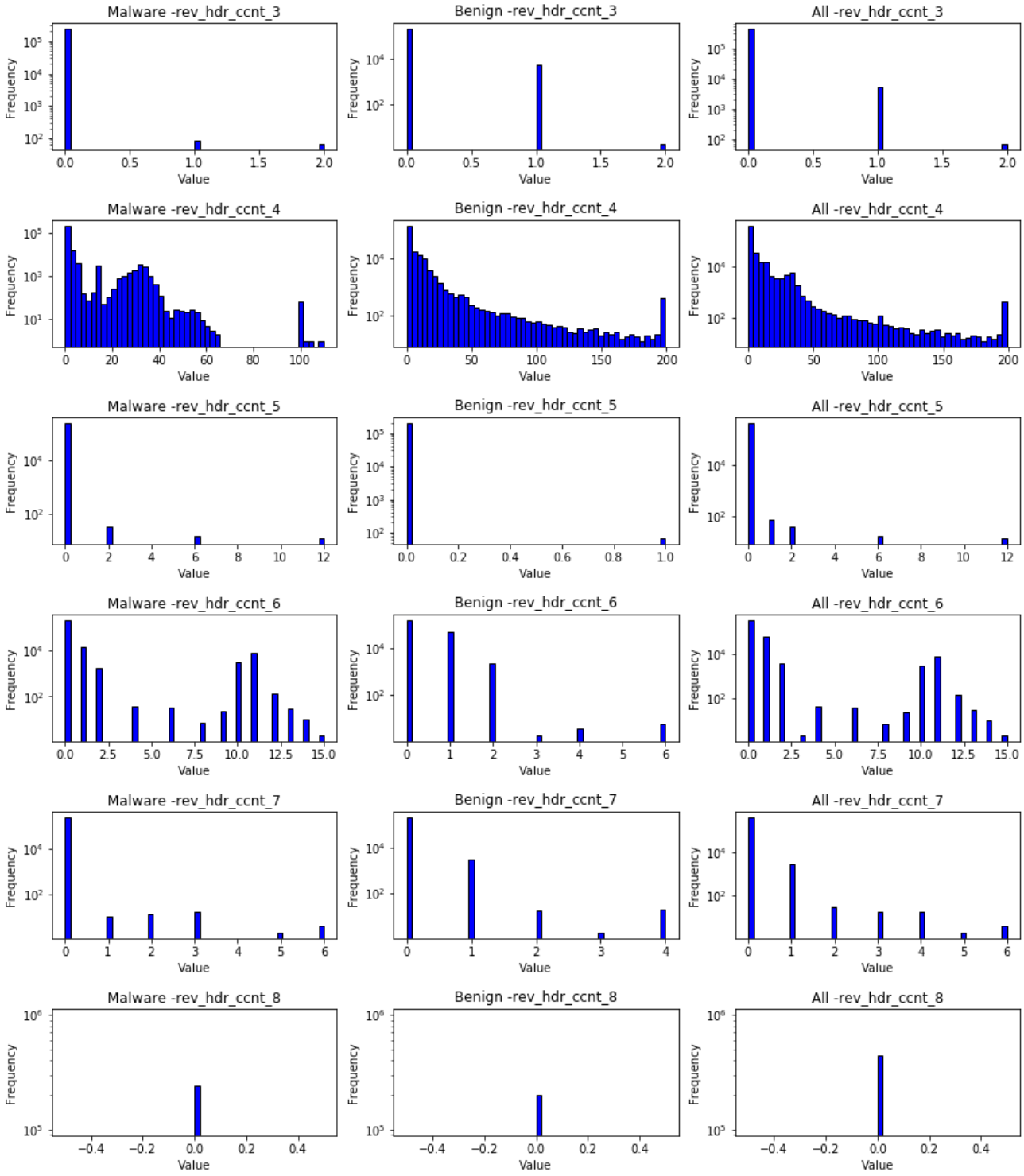

Malware -pld_ccnt_0
Benign -pld_ccnt_0
All -pld_ccnt_0
Malware -pld_ccnt_1
Benign -pld_ccnt_1
All -pld_ccnt_1
Malware -pld_ccnt_2
Benign -pld_ccnt_2
All -pld_ccnt_2
Malware -pld_ccnt_3
Benign -pld_ccnt_3
All -pld_ccnt_3
Malware -pld_ccnt_4
Benign -pld_ccnt_4
All -pld_ccnt_4
Malware -pld_ccnt_5
Benign -pld_ccnt_5
All -pld_ccnt_5
Frequency
Value

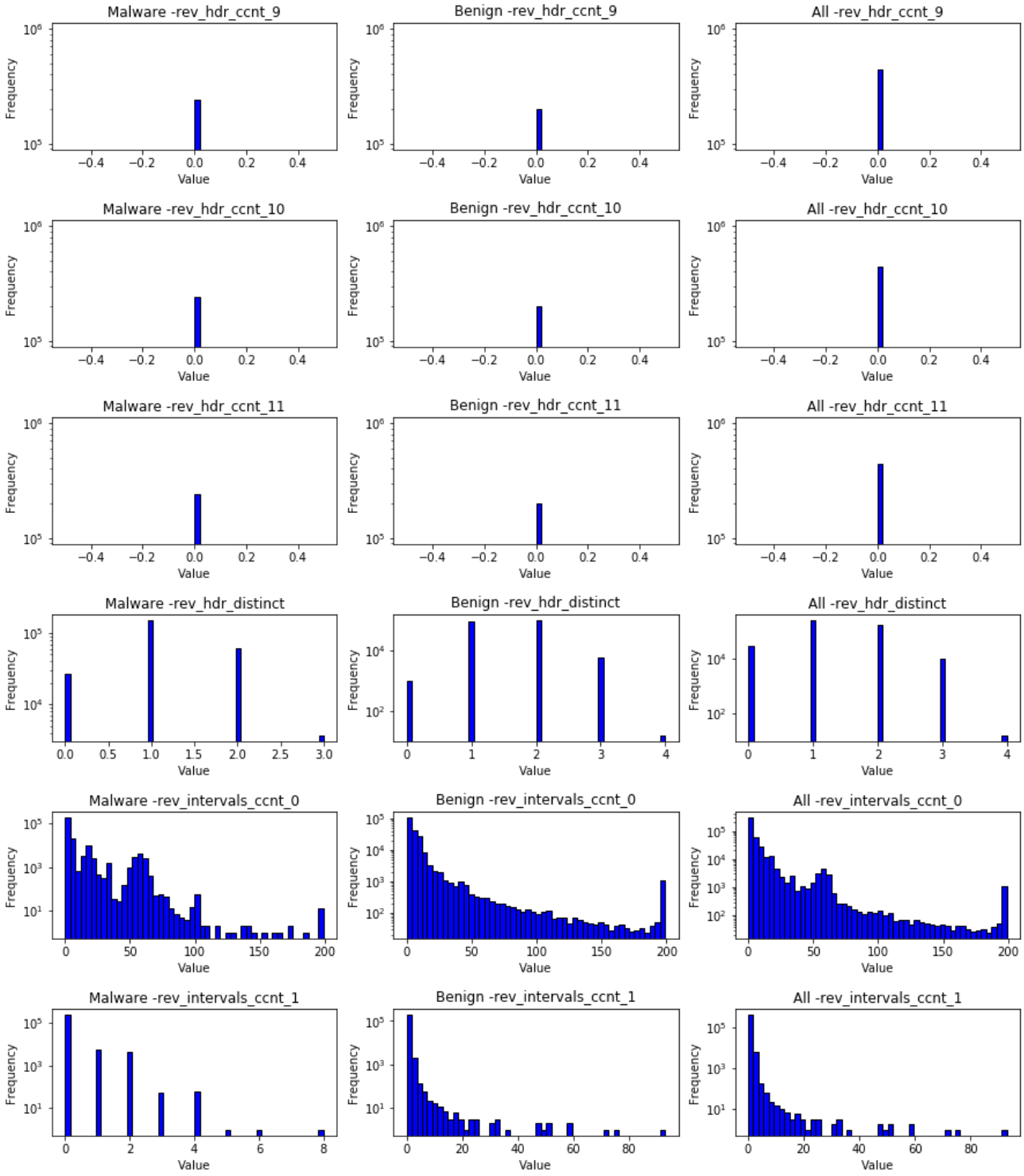

Malware -pld_ccnt_6
Benign -pld_ccnt_6
All -pld_ccnt_6
Malware -pld_ccnt_7
Benign -pld_ccnt_7
All -pld_ccnt_7
Malware -pld_ccnt_8
Benign -pld_ccnt_8
All -pld_ccnt_8
Malware -pld_ccnt_9
Benign -pld_ccnt_9
All -pld_ccnt_9
Malware -pld_ccnt_10
Benign -pld_ccnt_10
All -pld_ccnt_10
Malware -pld_ccnt_11
Benign -pld_ccnt_11
All -pld_ccnt_11
Frequency
Value

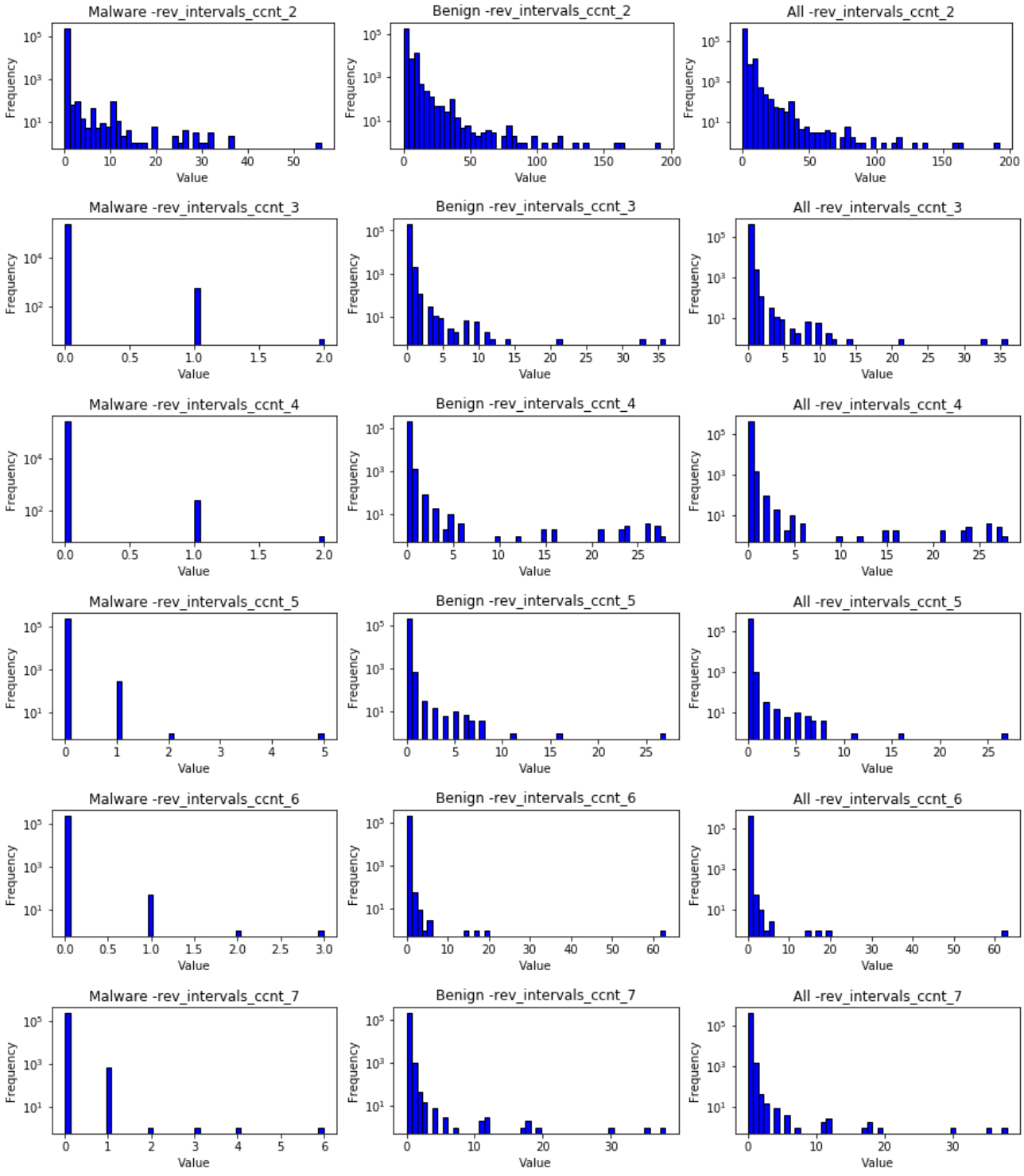

Malware -pld_ccnt_12
Benign -pld_ccnt_12
All -pld_ccnt_12
Malware -pld_ccnt_13
Benign -pld_ccnt_13
All -pld_ccnt_13
Malware -pld_ccnt_14
Benign -pld_ccnt_14
All -pld_ccnt_14
Malware -pld_ccnt_15
Benign -pld_ccnt_15
All -pld_ccnt_15
Malware -pld_distinct
Benign -pld_distinct
All -pld_distinct
Malware -pld_max
Benign -pld_max
All -pld_max
Frequency
Value

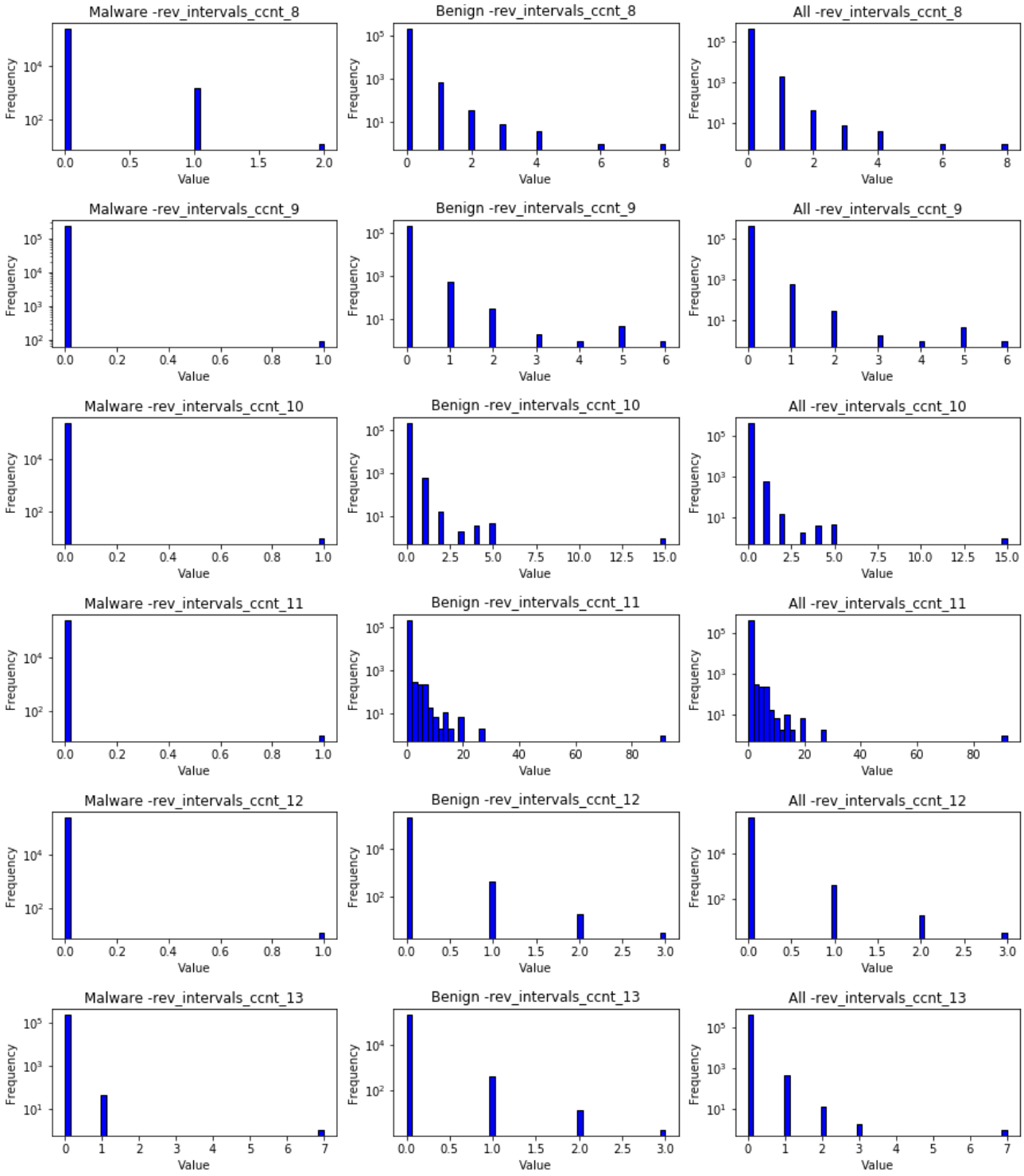

Malware -pld_mean
Benign -pld_mean
All -pld_mean
Malware -pld_median
Benign -pld_median
All -pld_median
Malware -pr
Benign -pr
All -pr
Malware -rev_ack_psh_rst_syn_fin_cnt_0
Benign -rev_ack_psh_rst_syn_fin_cnt_0
All -rev_ack_psh_rst_syn_fin_cnt_0
Malware -rev_ack_psh_rst_syn_fin_cnt_1
Benign -rev_ack_psh_rst_syn_fin_cnt_1
All -rev_ack_psh_rst_syn_fin_cnt_1
Malware -rev_ack_psh_rst_syn_fin_cnt_2
Benign -rev_ack_psh_rst_syn_fin_cnt_2
All -rev_ack_psh_rst_syn_fin_cnt_2
Frequency
Value

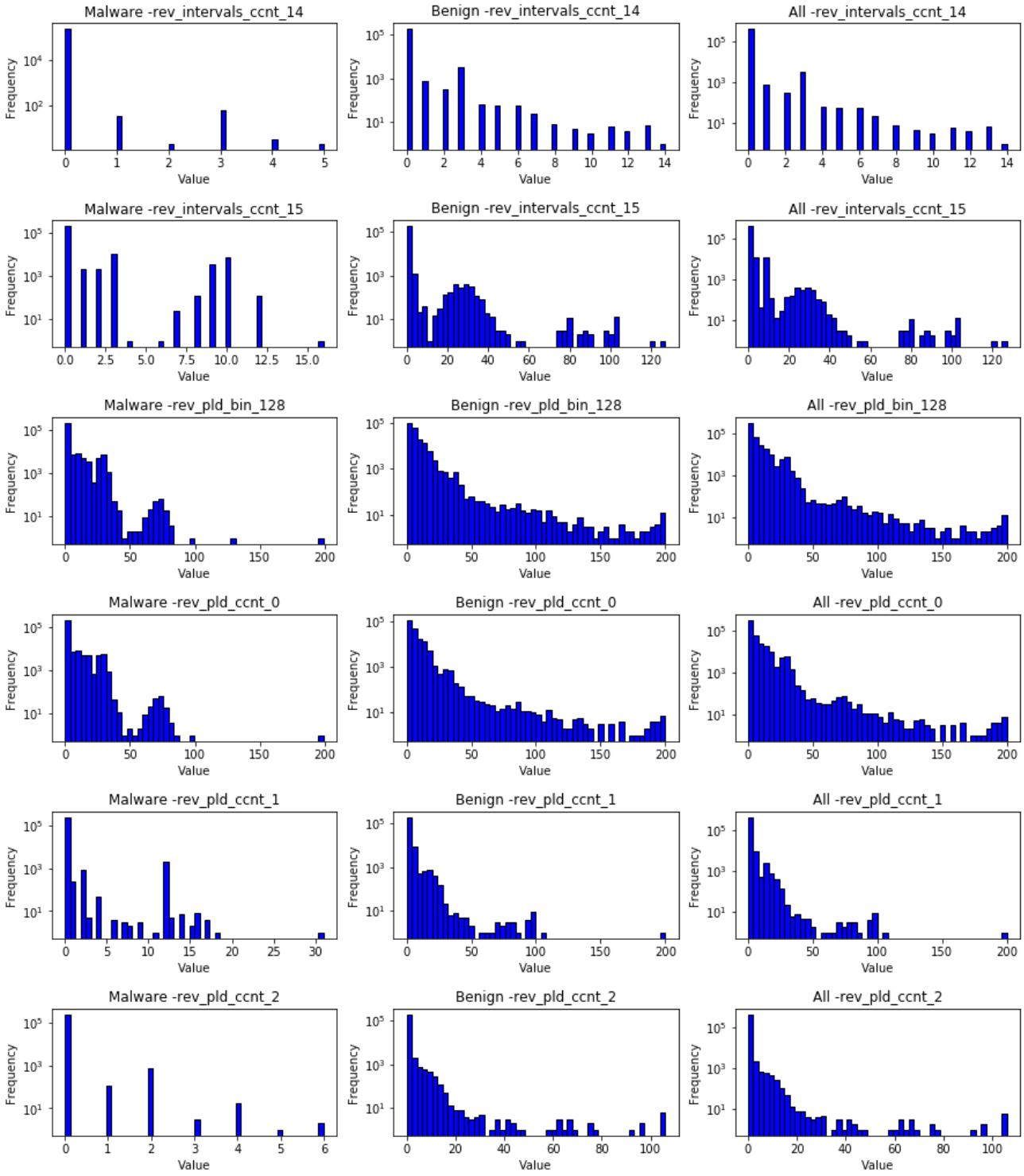

Malware -rev_ack_psh_rst_syn_fin_cnt_3
Benign -rev_ack_psh_rst_syn_fin_cnt_3
All -rev_ack_psh_rst_syn_fin_cnt_3
Malware -rev_ack_psh_rst_syn_fin_cnt_4
Benign -rev_ack_psh_rst_syn_fin_cnt_4
All -rev_ack_psh_rst_syn_fin_cnt_4
Malware -rev_hdr_bin_40
Benign -rev_hdr_bin_40
All -rev_hdr_bin_40
Malware -rev_hdr_ccnt_0
Benign -rev_hdr_ccnt_0
All -rev_hdr_ccnt_0
Malware -rev_hdr_ccnt_1
Benign -rev_hdr_ccnt_1
All -rev_hdr_ccnt_1
Malware -rev_hdr_ccnt_2
Benign -rev_hdr_ccnt_2
All -rev_hdr_ccnt_2
Frequency
Value

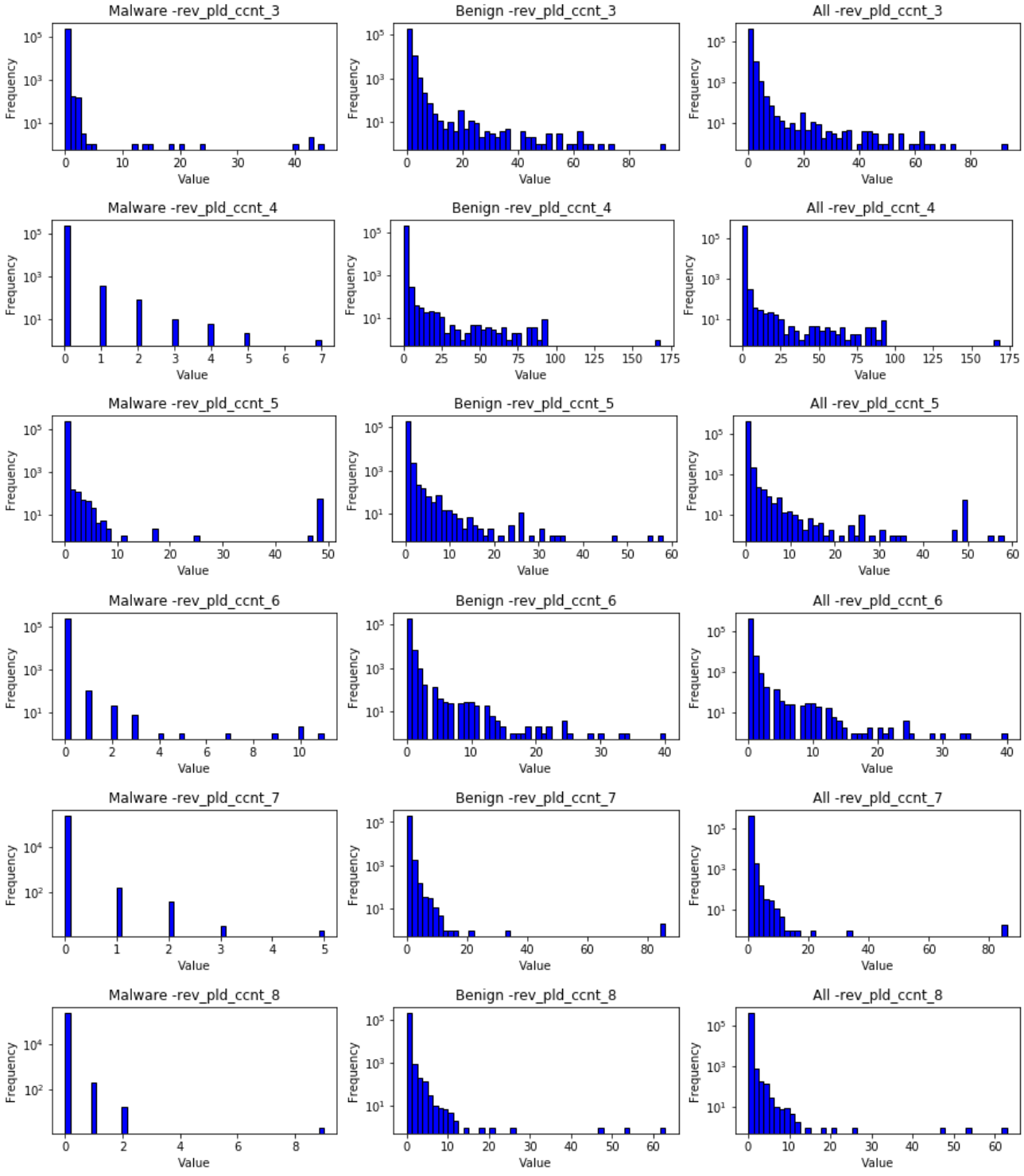

Malware -rev_hdr_ccnt_3
Benign -rev_hdr_ccnt_3
All -rev_hdr_ccnt_3
Malware -rev_hdr_ccnt_4
Benign -rev_hdr_ccnt_4
All -rev_hdr_ccnt_4
Malware -rev_hdr_ccnt_5
Benign -rev_hdr_ccnt_5
All -rev_hdr_ccnt_5
Malware -rev_hdr_ccnt_6
Benign -rev_hdr_ccnt_6
All -rev_hdr_ccnt_6
Malware -rev_hdr_ccnt_7
Benign -rev_hdr_ccnt_7
All -rev_hdr_ccnt_7
Malware -rev_hdr_ccnt_8
Benign -rev_hdr_ccnt_8
All -rev_hdr_ccnt_8
Frequency
Value

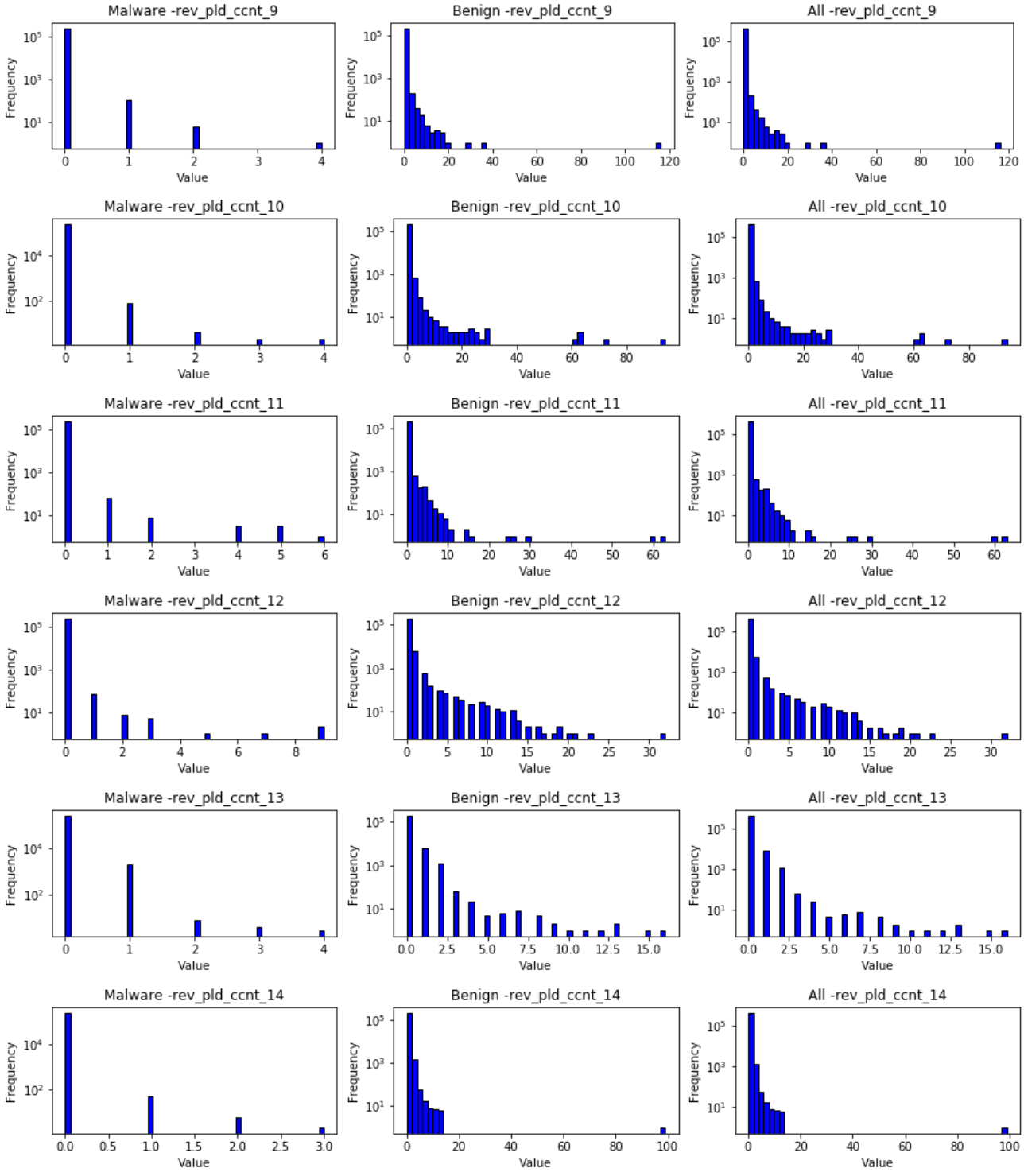

Malware -rev_hdr_ccnt_9
Benign -rev_hdr_ccnt_9
All -rev_hdr_ccnt_9
Malware -rev_hdr_ccnt_10
Benign -rev_hdr_ccnt_10
All -rev_hdr_ccnt_10
Malware -rev_hdr_ccnt_11
Benign -rev_hdr_ccnt_11
All -rev_hdr_ccnt_11
Malware -rev_hdr_distinct
Benign -rev_hdr_distinct
All -rev_hdr_distinct
Malware -rev_intervals_ccnt_0
Benign -rev_intervals_ccnt_0
All -rev_intervals_ccnt_0
Malware -rev_intervals_ccnt_1
Benign -rev_intervals_ccnt_1
All -rev_intervals_ccnt_1
Frequency
Value

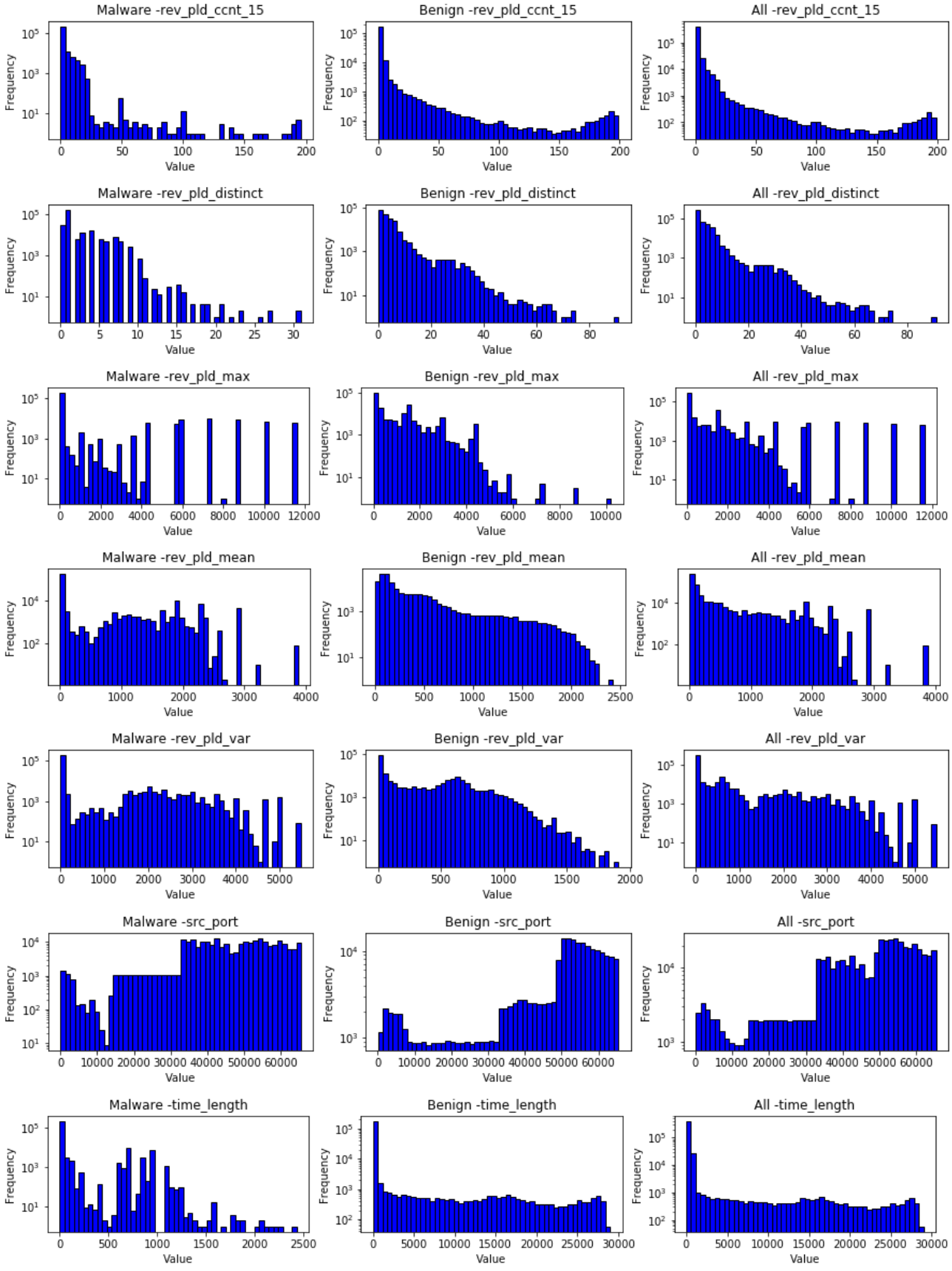

Malware -rev_intervals_ccnt_2
Benign -rev_intervals_ccnt_2
All -rev_intervals_ccnt_2
Malware -rev_intervals_ccnt_3
Benign -rev_intervals_ccnt_3
All -rev_intervals_ccnt_3
Malware -rev_intervals_ccnt_4
Benign -rev_intervals_ccnt_4
All -rev_intervals_ccnt_4
Malware -rev_intervals_ccnt_5
Benign -rev_intervals_ccnt_5
All -rev_intervals_ccnt_5
Malware -rev_intervals_ccnt_6
Benign -rev_intervals_ccnt_6
All -rev_intervals_ccnt_6
Malware -rev_intervals_ccnt_7
Benign -rev_intervals_ccnt_7
All -rev_intervals_ccnt_7
Frequency
Value

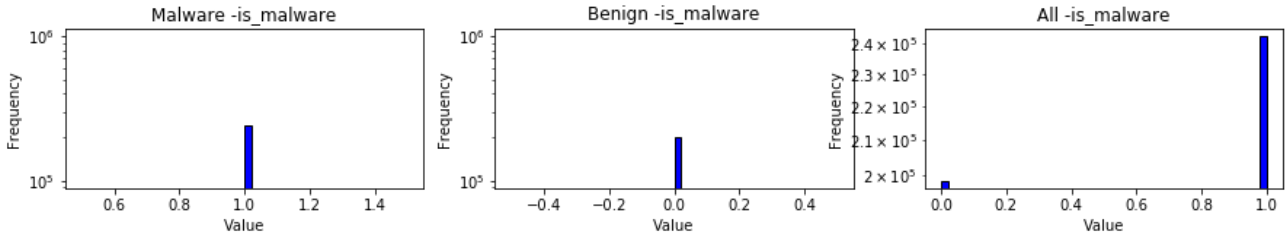

Malware -rev_intervals_ccnt_8
Benign -rev_intervals_ccnt_8
All -rev_intervals_ccnt_8
Malware -rev_intervals_ccnt_9
Benign -rev_intervals_ccnt_9
All -rev_intervals_ccnt_9
Malware -rev_intervals_ccnt_10
Benign -rev_intervals_ccnt_10
All -rev_intervals_ccnt_10
Malware -rev_intervals_ccnt_11
Benign -rev_intervals_ccnt_11
All -rev_intervals_ccnt_11
Malware -rev_intervals_ccnt_12
Benign -rev_intervals_ccnt_12
All -rev_intervals_ccnt_12
Malware -rev_intervals_ccnt_13
Benign -rev_intervals_ccnt_13
All -rev_intervals_ccnt_13
Frequency
Value


Malware -rev_intervals_ccnt_14
Benign -rev_intervals_ccnt_14
All -rev_intervals_ccnt_14
Malware -rev_intervals_ccnt_15
Benign -rev_intervals_ccnt_15
All -rev_intervals_ccnt_15
Malware -rev_pld_bin_128
Benign -rev_pld_bin_128
All -rev_pld_bin_128
Malware -rev_pld_ccnt_0
Benign -rev_pld_ccnt_0
All -rev_pld_ccnt_0
Malware -rev_pld_ccnt_1
Benign -rev_pld_ccnt_1
All -rev_pld_ccnt_1
Malware -rev_pld_ccnt_2
Benign -rev_pld_ccnt_2
All -rev_pld_ccnt_2
Frequency
Value

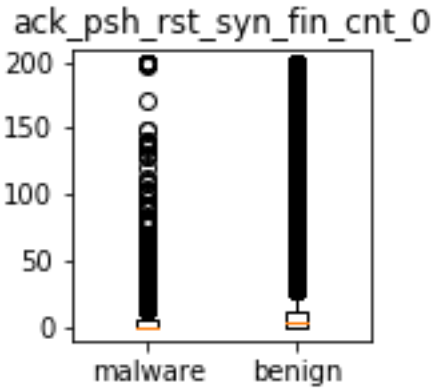

Malware -rev_pld_ccnt_3
Benign -rev_pld_ccnt_3
All -rev_pld_ccnt_3
Malware -rev_pld_ccnt_4
Benign -rev_pld_ccnt_4
All -rev_pld_ccnt_4
Malware -rev_pld_ccnt_5
Benign -rev_pld_ccnt_5
All -rev_pld_ccnt_5
Malware -rev_pld_ccnt_6
Benign -rev_pld_ccnt_6
All -rev_pld_ccnt_6
Malware -rev_pld_ccnt_7
Benign -rev_pld_ccnt_7
All -rev_pld_ccnt_7
Malware -rev_pld_ccnt_8
Benign -rev_pld_ccnt_8
All -rev_pld_ccnt_8
Frequency
Value

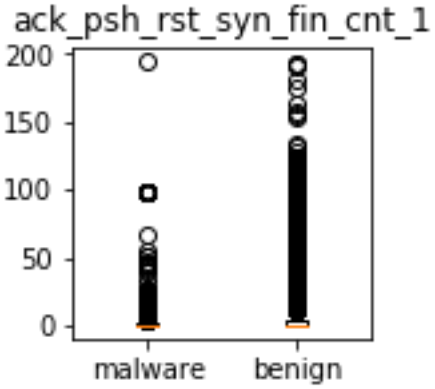

Malware -rev_pld_ccnt_9
Benign -rev_pld_ccnt_9
All -rev_pld_ccnt_9
Malware -rev_pld_ccnt_10
Benign -rev_pld_ccnt_10
All -rev_pld_ccnt_10
Malware -rev_pld_ccnt_11
Benign -rev_pld_ccnt_11
All -rev_pld_ccnt_11
Malware -rev_pld_ccnt_12
Benign -rev_pld_ccnt_12
All -rev_pld_ccnt_12
Malware -rev_pld_ccnt_13
Benign -rev_pld_ccnt_13
All -rev_pld_ccnt_13
Malware -rev_pld_ccnt_14
Benign -rev_pld_ccnt_14
All -rev_pld_ccnt_14
Frequency
Value

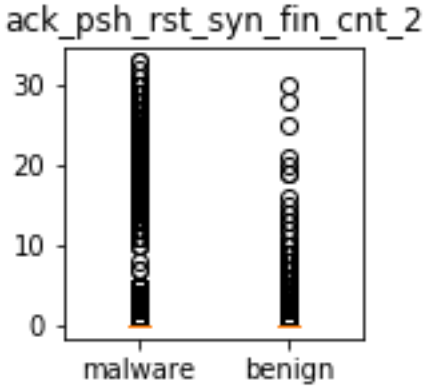

Malware -rev_pld_ccnt_15
Benign -rev_pld_ccnt_15
All -rev_pld_ccnt_15
Malware -rev_pld_distinct
Benign -rev_pld_distinct
All -rev_pld_distinct
Malware -rev_pld_max
Benign -rev_pld_max
All -rev_pld_max
Malware -rev_pld_mean
Benign -rev_pld_mean
All -rev_pld_mean
Malware -rev_pld_var
Benign -rev_pld_var
All -rev_pld_var
Malware -src_port
Benign -src_port
All -src_port
Malware -time_length
Benign -time_length
All -time_length
Frequency
Value

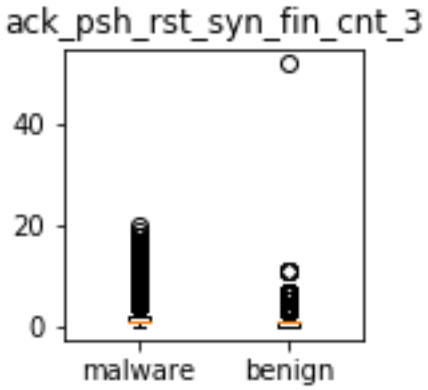


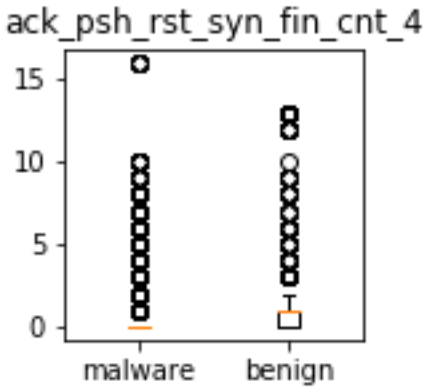
Univariate Analysis - Box Plots

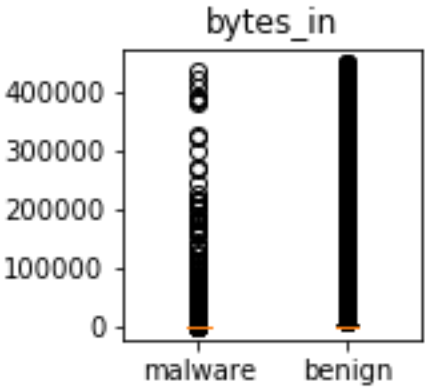


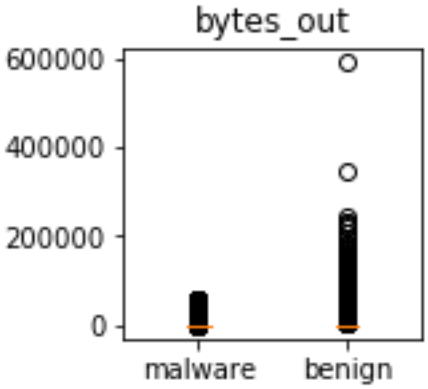


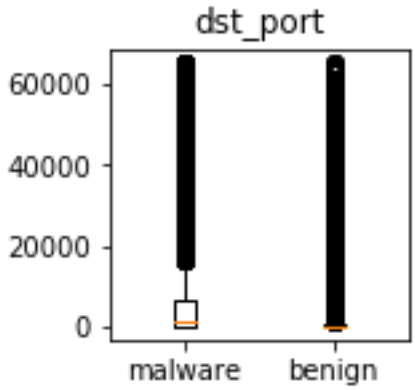


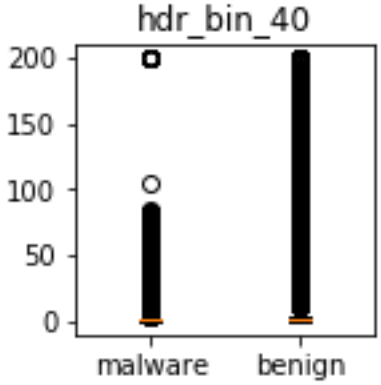

ack_psh_rst_syn_fin_cnt_3
40
20
0
malware
benign

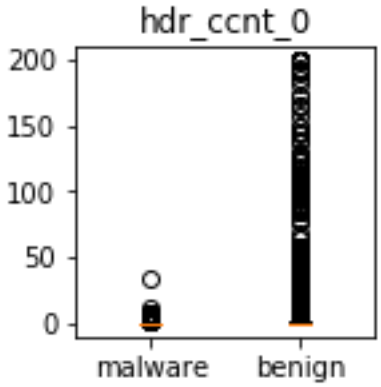

ack_psh_rst_syn_fin_cnt_4
15
10
5
0
malware
benign

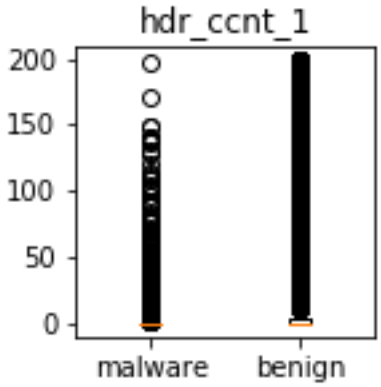

bytes_in
400000
300000
200000
100000
0
malware
benign

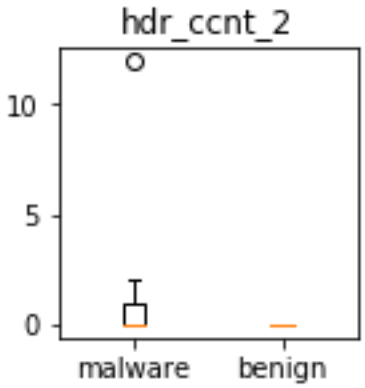

bytes_out
600000
400000
200000
0
malware
benign

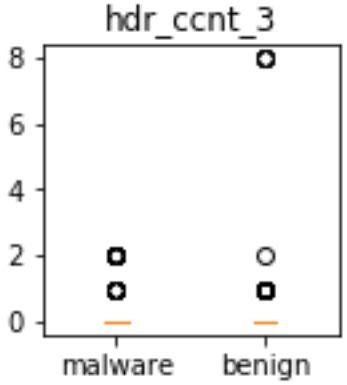

dst_port
60000
40000
20000
0
malware
benign

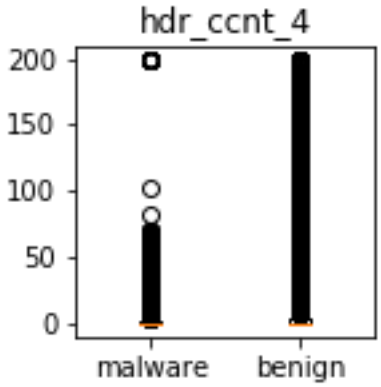

hdr_bin_40
200
150
100
50
0
malware
benign

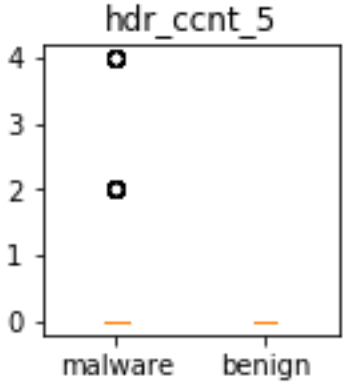

hdr_ccnt_0
200
150
100
50
0
malware
benign

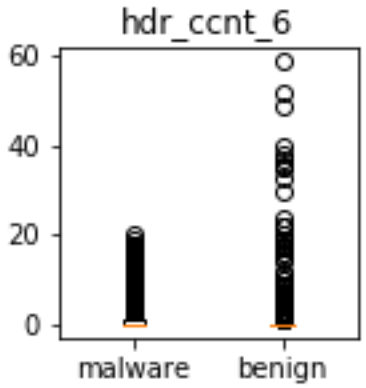

hdr_ccnt_1
200
150
100
50
0
malware
benign

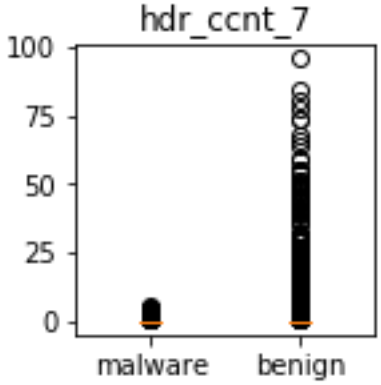

hdr_ccnt_2
10
5
0
malware
benign

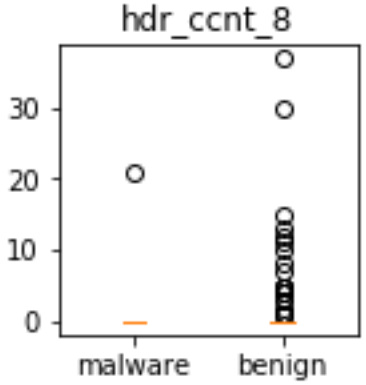

hdr_ccnt_3
8
6
4
2
0
malware
benign

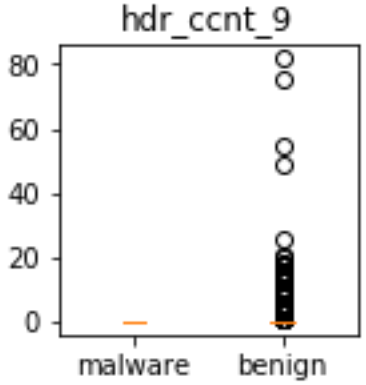

hdr_ccnt_4
200
150
100
50
0
malware
benign

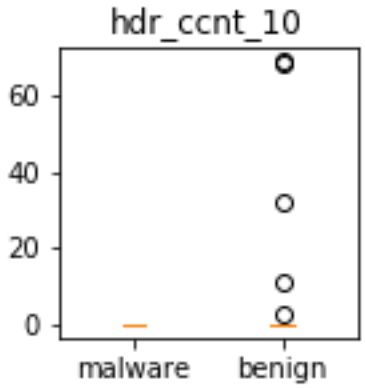

hdr_ccnt_5
4
3
2
1
0
malware
benign

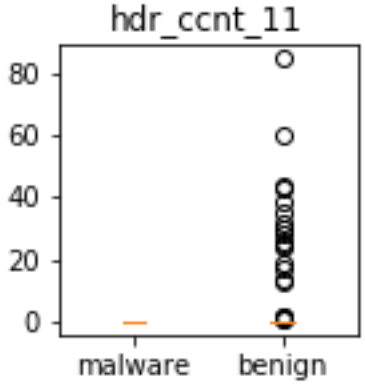

hdr_ccnt_6
60
40
20
0
malware
benign

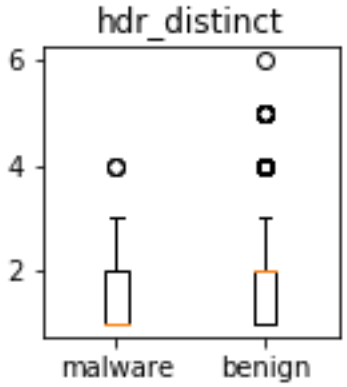

hdr_ccnt_7
100
75
50
25
0
malware
benign

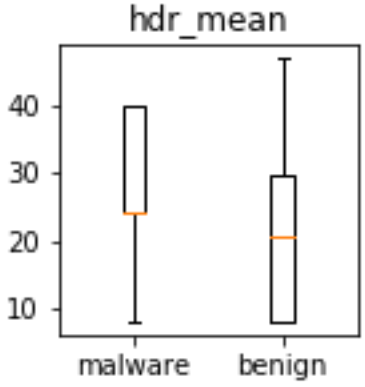

hdr_ccnt_8
30
20
10
0
malware
benign

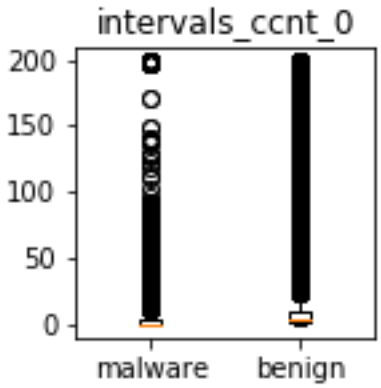

hdr_ccnt_9
80
60
40
20
0
malware
benign

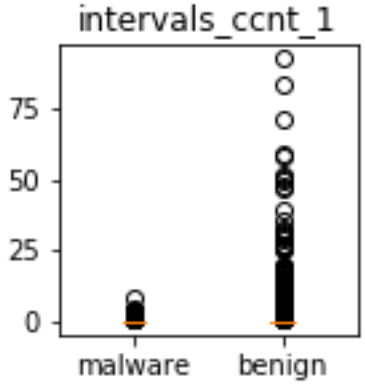

hdr_ccnt_10
60
40
20
0
malware
benign

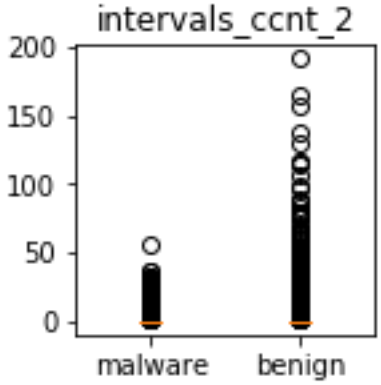

hdr_ccnt_11
80
60
40
20
0
malware
benign

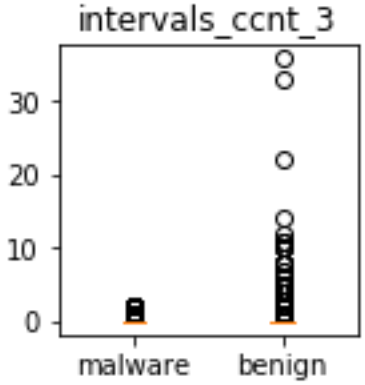

hdr_distinct
6
4
2
malware
benign

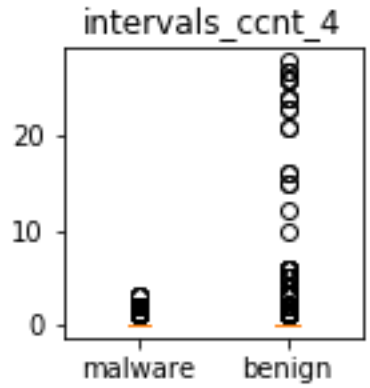

hdr_mean
40
30
20
10
malware
benign

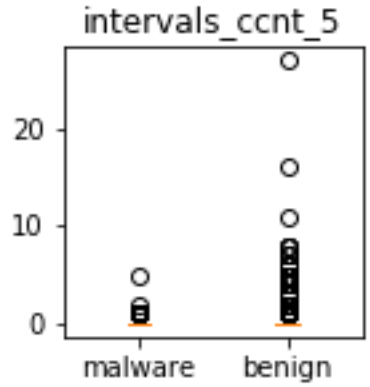

intervals_ccnt_0
200
150
100
50
0
malware
benign

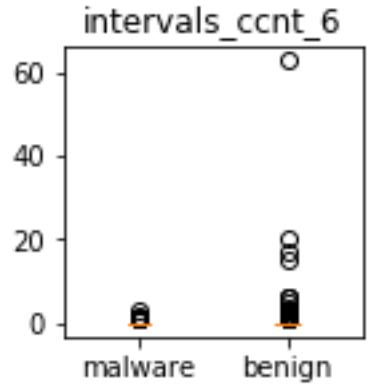

intervals_ccnt_1
75
50
25
0
malware
benign

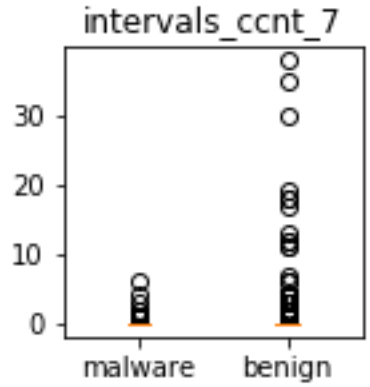

intervals_ccnt_2
200
150
100
50
0
malware
benign

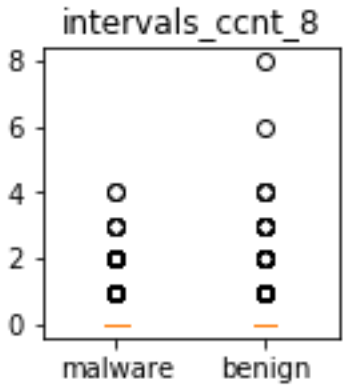

intervals_ccnt_3
30
20
10
0
malware
benign

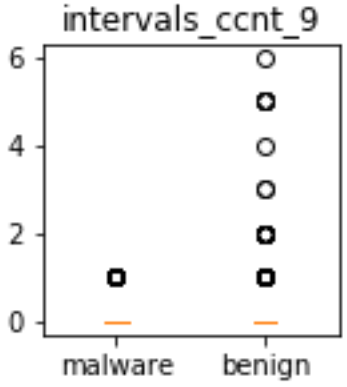

intervals_ccnt_4
20
10
0
malware
benign

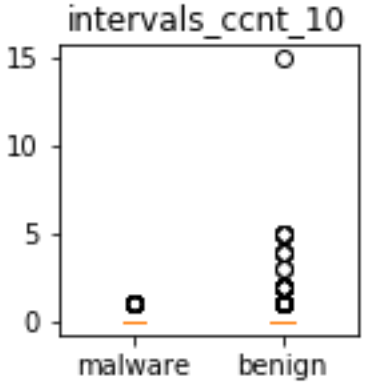

intervals_ccnt_5
20
10
0
malware
benign

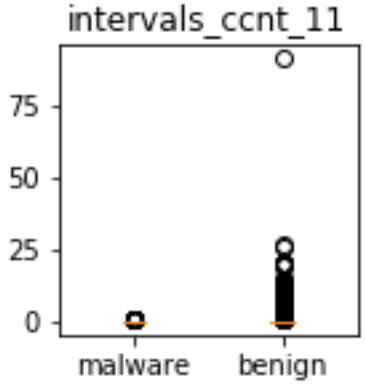

intervals_ccnt_6
60
40
20
0
malware
benign

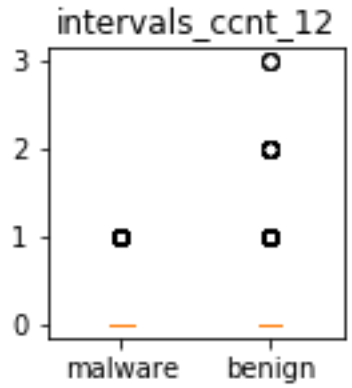

intervals_ccnt_7
30
20
10
0
malware
benign

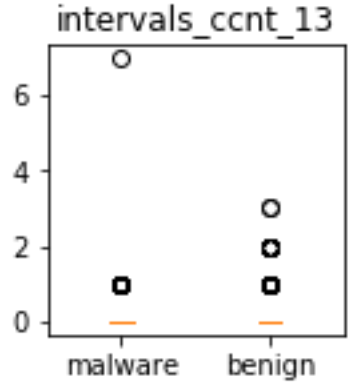

intervals_ccnt_8
8
6
4
2
0
malware
benign

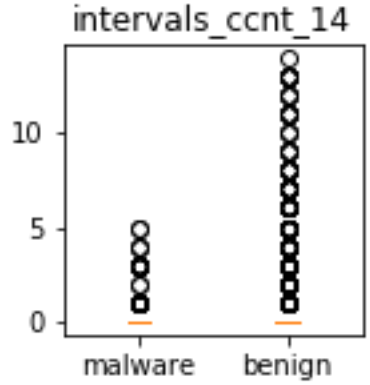

intervals_ccnt_9
6
4
2
0
malware
benign

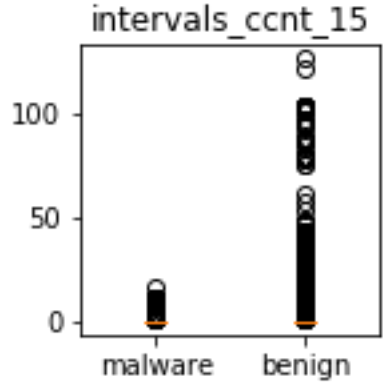

intervals_ccnt_10
15
10
5
0
malware
benign

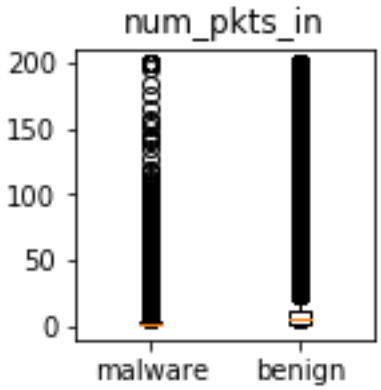

intervals_ccnt_11
75
50
25
0
malware
benign

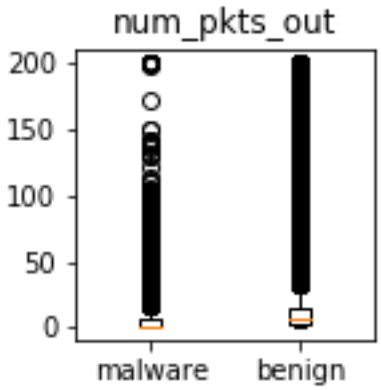

intervals_ccnt_12
3
2
1
0
malware
benign

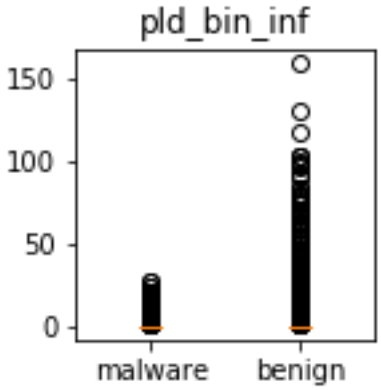

intervals_ccnt_13
6
4
2
0
malware
benign

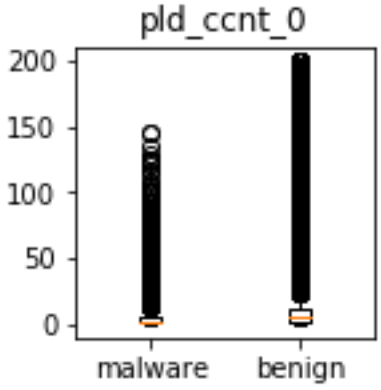

intervals_ccnt_14
10
5
0
malware
benign

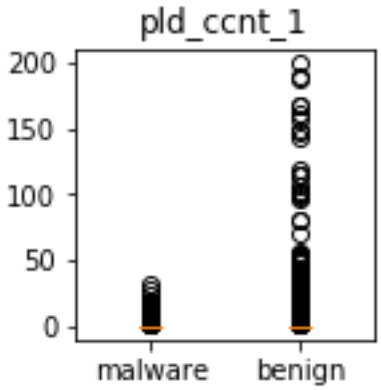

intervals_ccnt_15
100
50
0
malware
benign

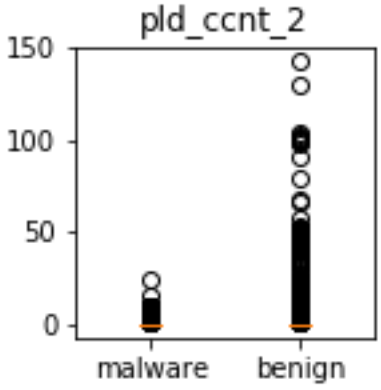

num_pkts_in
200
150
100
50
0
malware
benign

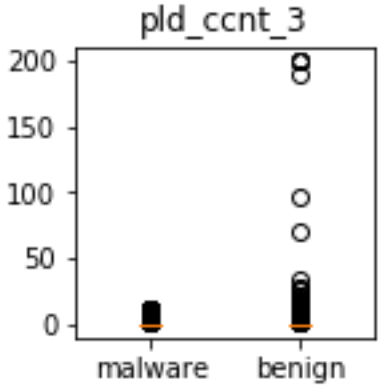

num_pkts_out
200
150
100
50
0
malware
benign

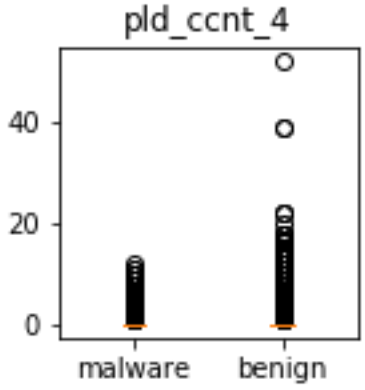

pld_bin_inf
150
100
50
0
malware
benign

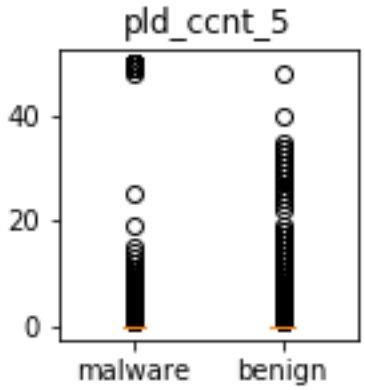

pld_ccnt_0
200
150
100
50
0
malware
benign

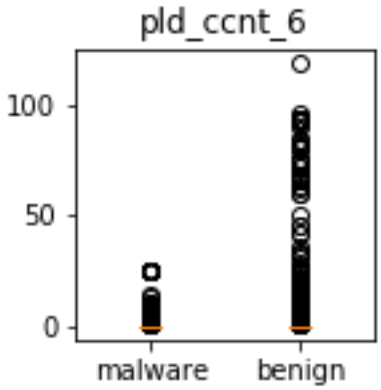

pld_ccnt_1
200
150
100
50
0
malware
benign

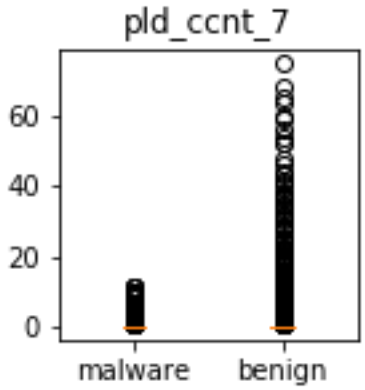

pld_ccnt_2
150
100
50
0
malware
benign

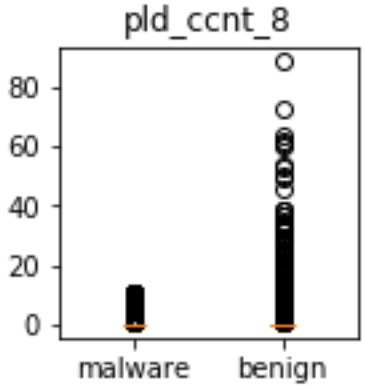

pld_ccnt_3
200
150
100
50
0
malware
benign

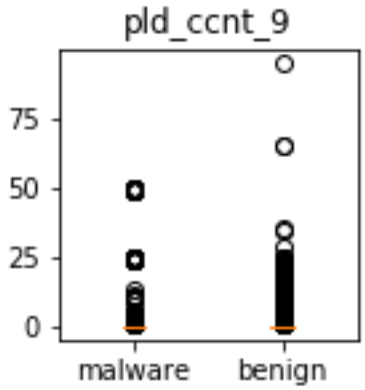

pld_ccnt_4
40
20
0
malware
benign

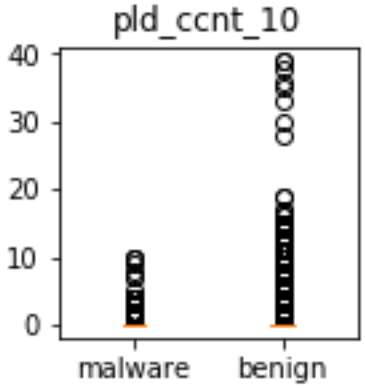

pld_ccnt_5
40
20
0
malware
benign

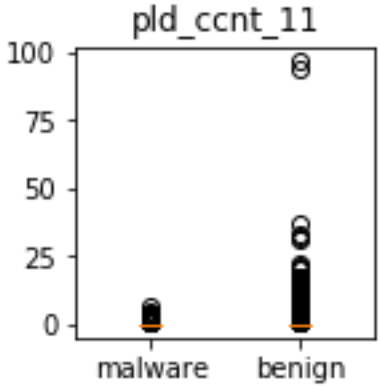

pld_ccnt_6
100
50
0
malware
benign

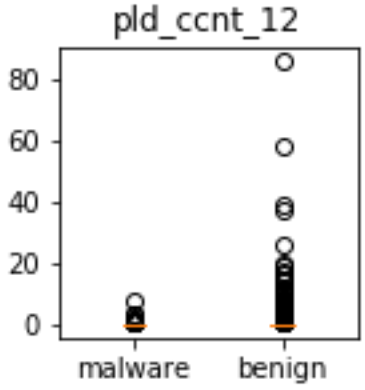

pld_ccnt_7
60
40
20
0
malware
benign

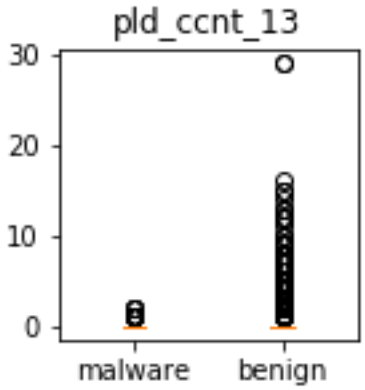

pld_ccnt_8
80
60
40
20
0
malware
benign

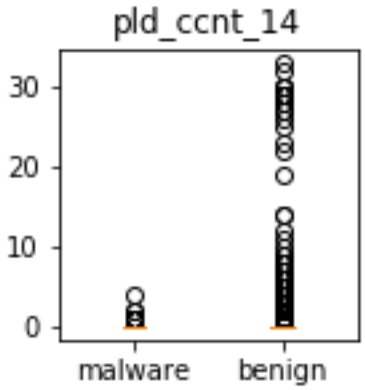

pld_ccnt_9
75
50
25
0
malware
benign

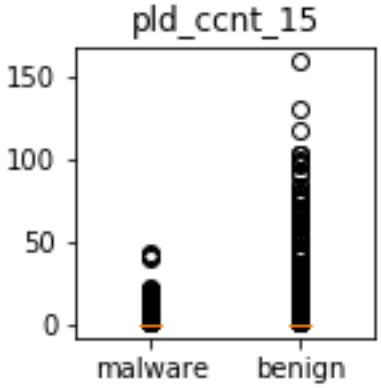

pld_ccnt_10
40
30
20
10
0
malware
benign

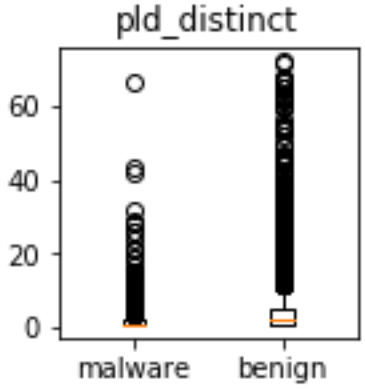

pld_ccnt_11
100
75
50
25
0
malware
benign

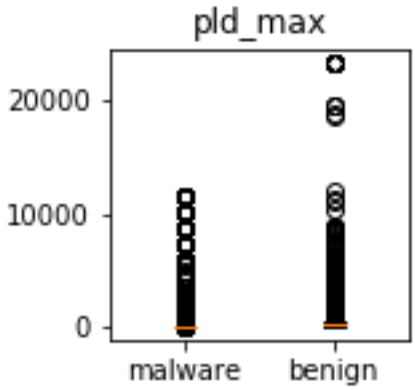

pld_ccnt_12
80
60
40
20
0
malware
benign

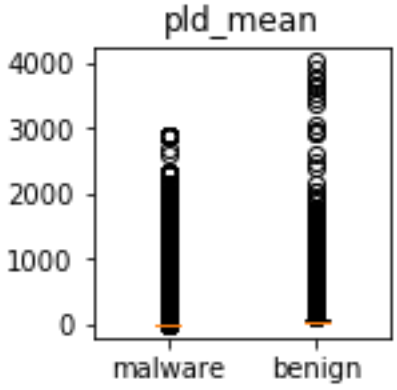

pld_ccnt_13
30
20
10
0
malware
benign

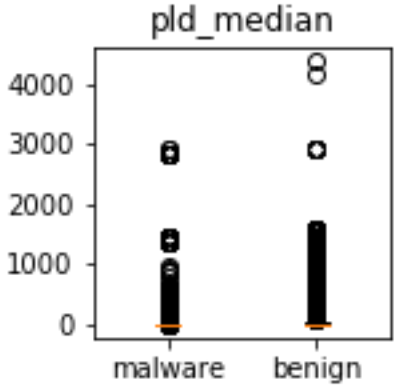

pld_ccnt_14
30
20
10
0
malware
benign

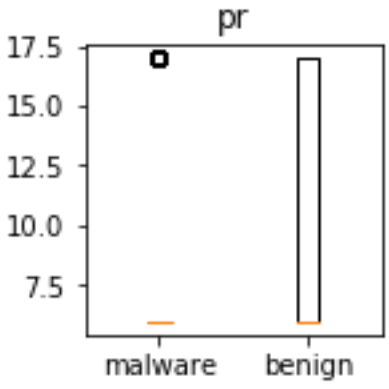

pld_ccnt_15
150
100
50
0
malware
benign


pld_distinct
60
40
20
0
malware
benign

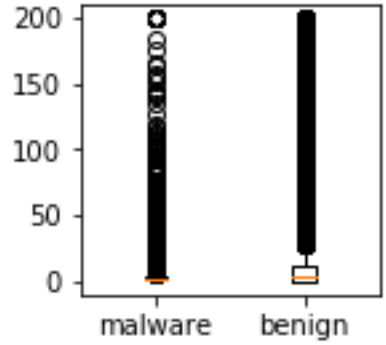

pld_max
20000
10000
0
malware
benign


pld_mean
4000
3000
2000
1000
0
malware
benign

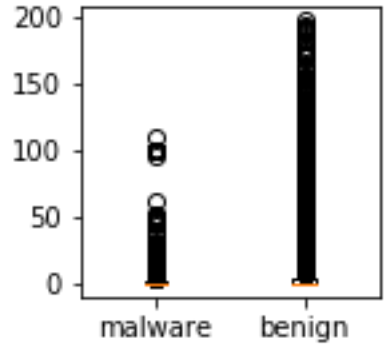

pld_median
4000
3000
2000
1000
0
malware
benign


pr
17.5
15.0
12.5
10.0
7.5
malware
benign

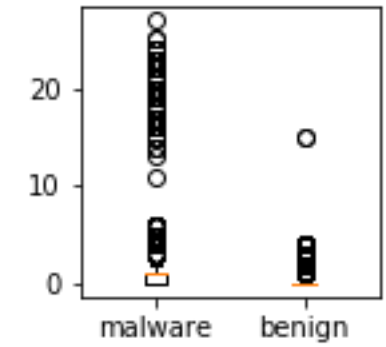
rev_ack_psh_rst_syn_fin_cnt_0



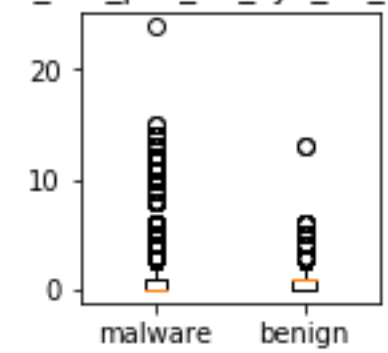
rev_ack_psh_rst_syn_fin_cnt_1

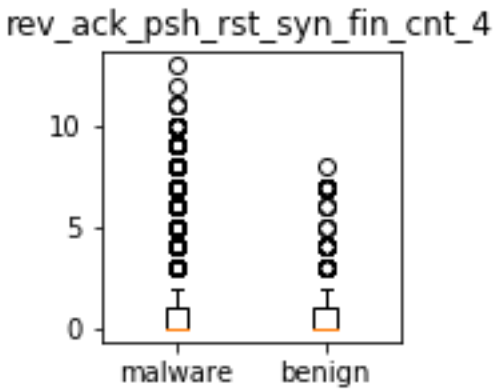


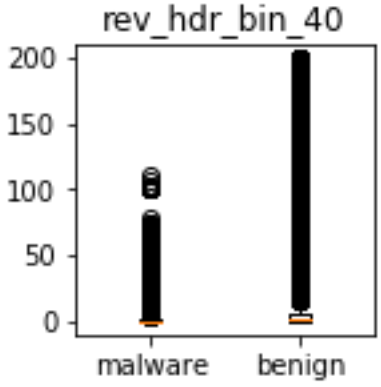
rev_ack_psh_rst_syn_fin_cnt_2

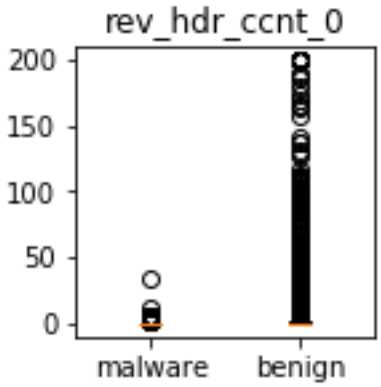


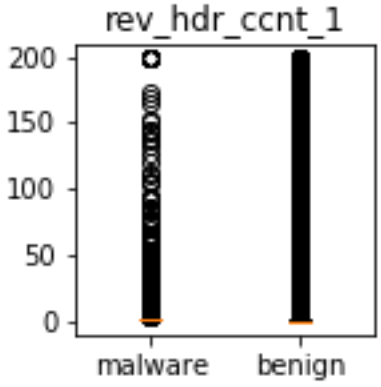
rev_ack_psh_rst_syn_fin_cnt_3

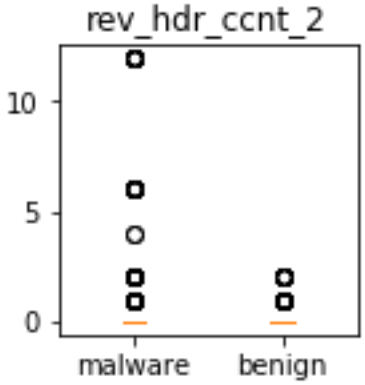


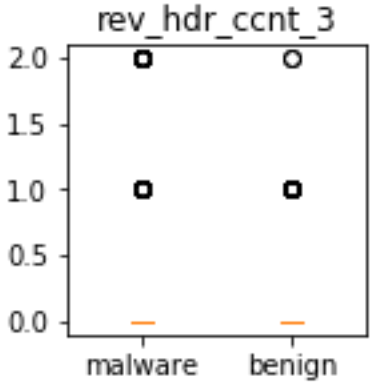

rev_ack_psh_rst_syn_fin_cnt_4
10
5
0
malware
benign

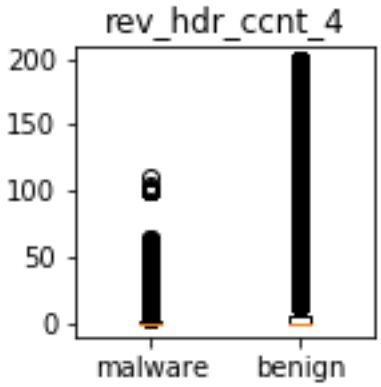

rev_hdr_bin_40
200
150
100
50
0
malware
benign

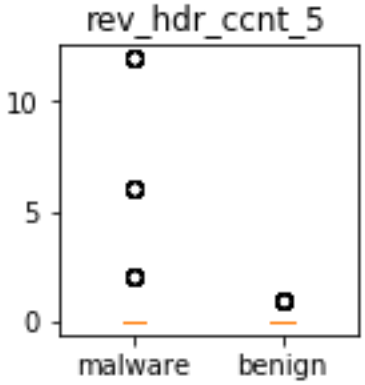

rev_hdr_ccnt_0
200
150
100
50
0
malware
benign

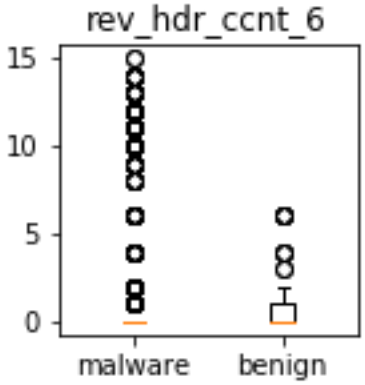

rev_hdr_ccnt_1
200
150
100
50
0
malware
benign

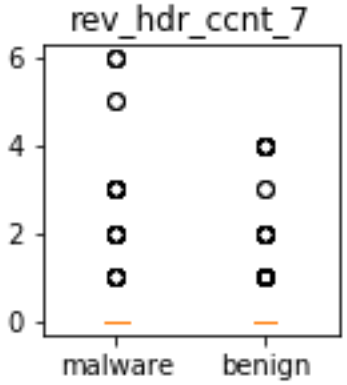

rev_hdr_ccnt_2
10
5
0
malware
benign

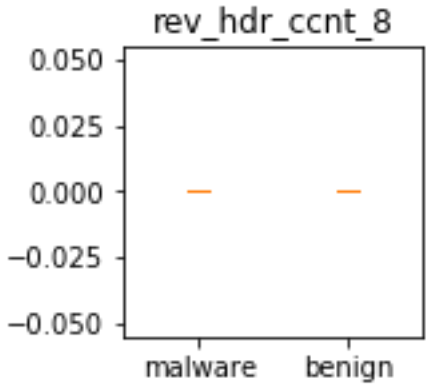

rev_hdr_ccnt_3
2.0
1.5
1.0
0.5
0.0
malware
benign

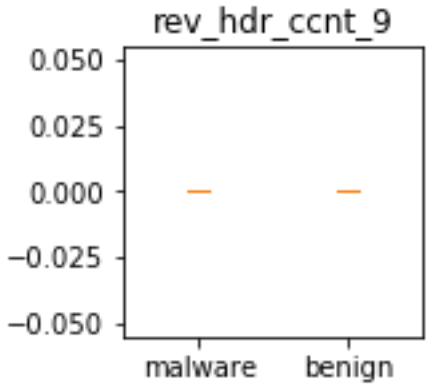

rev_hdr_ccnt_4
200
150
100
50
0
malware
benign

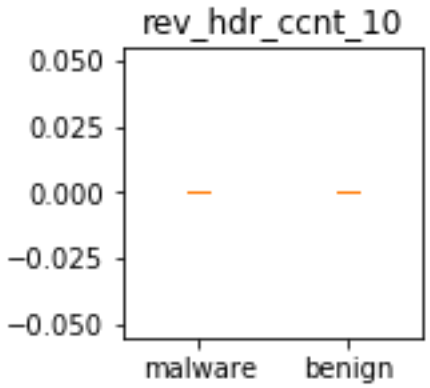

rev_hdr_ccnt_5
10
5
0
malware
benign

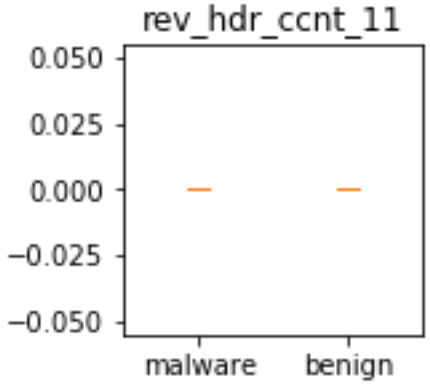

rev_hdr_ccnt_6
15
10
5
0
malware
benign

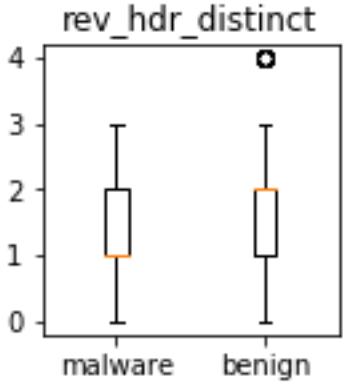

rev_hdr_ccnt_7
6
4
2
0
malware
benign

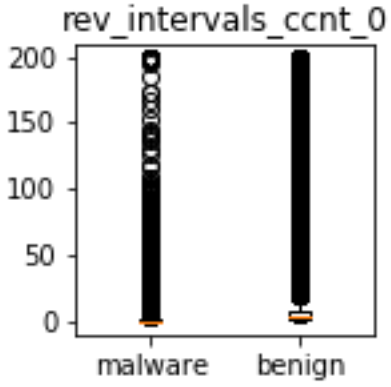

rev_hdr_ccnt_8
0.050
0.025
0.000
−0.025
−0.050
malware
benign

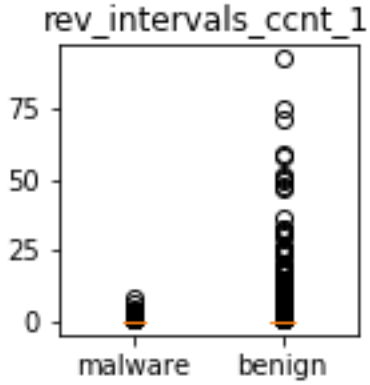

rev_hdr_ccnt_9
0.050
0.025
0.000
−0.025
−0.050
malware
benign

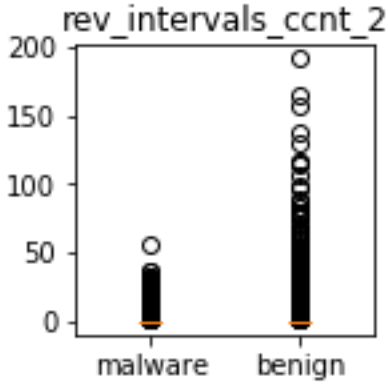

rev_hdr_ccnt_10
0.050
0.025
0.000
−0.025
−0.050
malware
benign

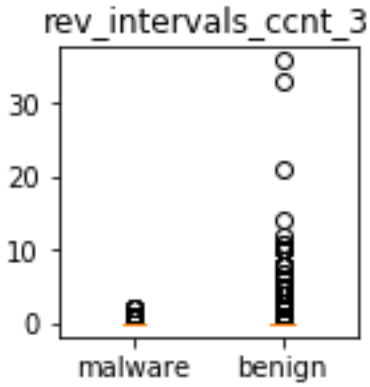

rev_hdr_ccnt_11
0.050
0.025
0.000
−0.025
−0.050
malware
benign

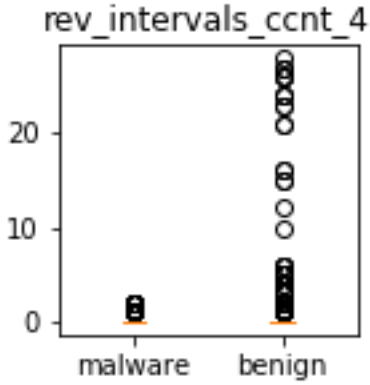

rev_hdr_distinct
4
3
2
1
0
malware
benign

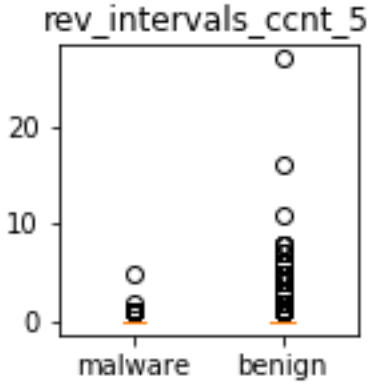

rev_intervals_ccnt_0
200
150
100
50
0
malware
benign

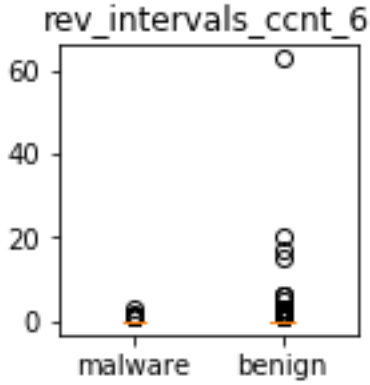

rev_intervals_ccnt_1
75
50
25
0
malware
benign

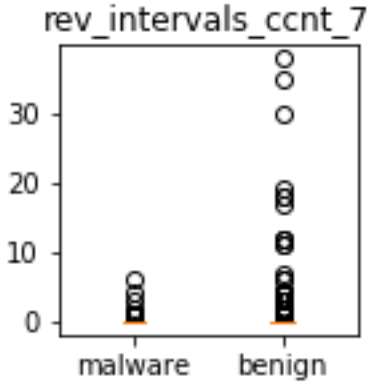

rev_intervals_ccnt_2
200
150
100
50
0
malware
benign

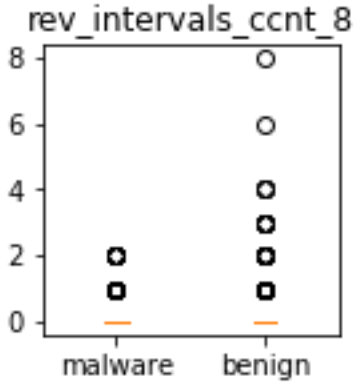

rev_intervals_ccnt_3
30
20
10
0
malware
benign

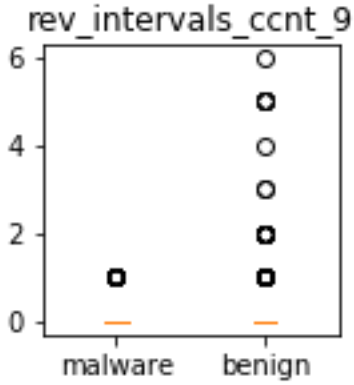

rev_intervals_ccnt_4
20
10
0
malware
benign

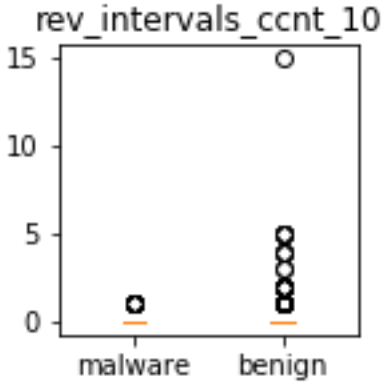

rev_intervals_ccnt_5
20
10
0
malware
benign

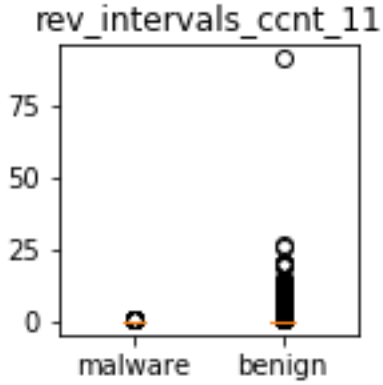

rev_intervals_ccnt_6
60
40
20
0
malware
benign

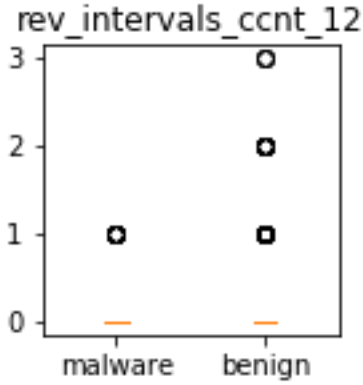

rev_intervals_ccnt_7
30
20
10
0
malware
benign

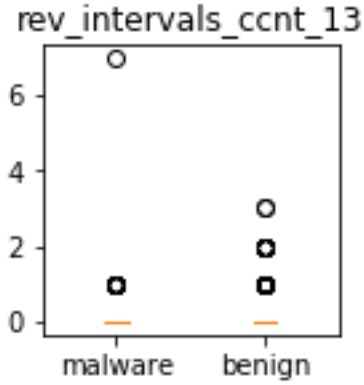

rev_intervals_ccnt_8
8
6
4
2
0
malware
benign

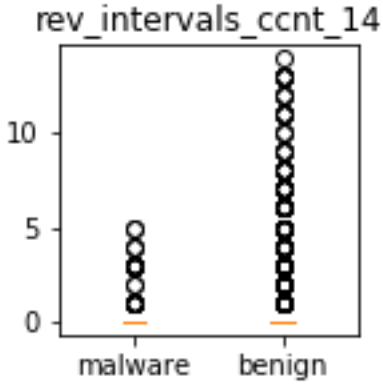

rev_intervals_ccnt_9
6
4
2
0
malware
benign

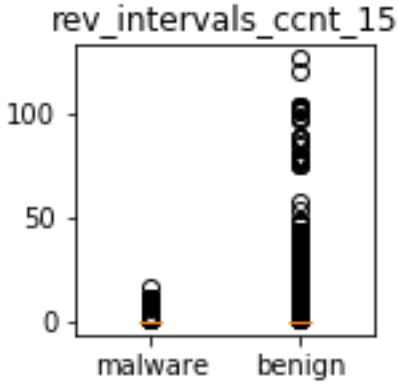

rev_intervals_ccnt_10
15
10
5
0
malware
benign

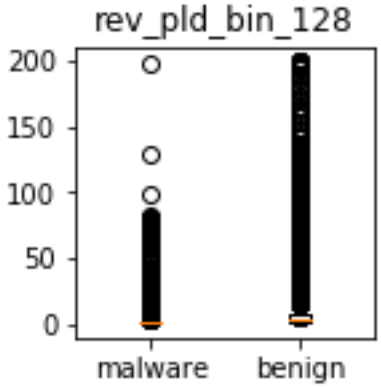

rev_intervals_ccnt_11
75
50
25
0
malware
benign

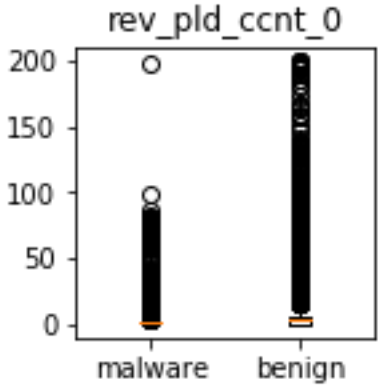

rev_intervals_ccnt_12
3
2
1
0
malware
benign

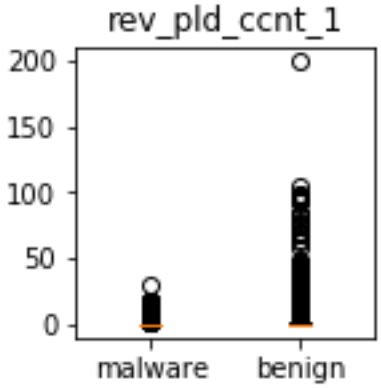

rev_intervals_ccnt_13
6
4
2
0
malware
benign

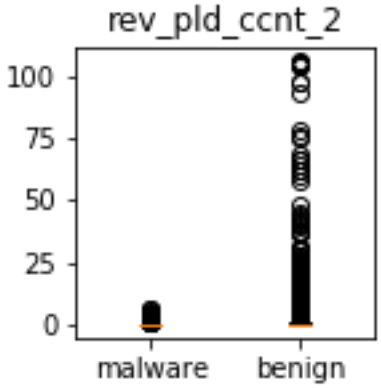

rev_intervals_ccnt_14
10
5
0
malware
benign

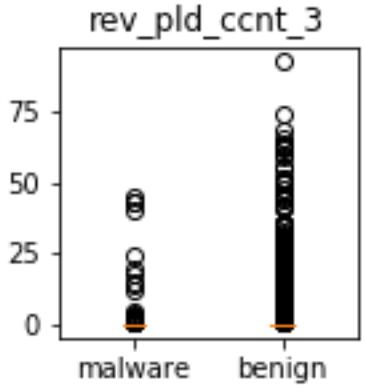

rev_intervals_ccnt_15
100
50
0
malware
benign

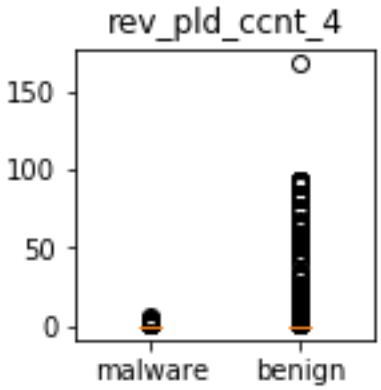

rev_pld_bin_128
200
150
100
50
0
malware
benign

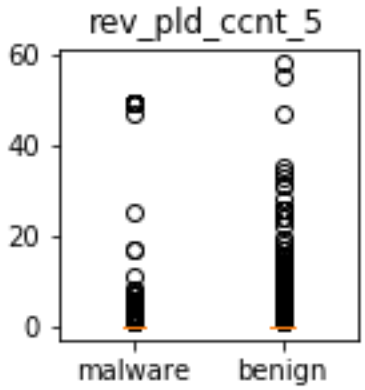

rev_pld_ccnt_0
200
150
100
50
0
malware
benign

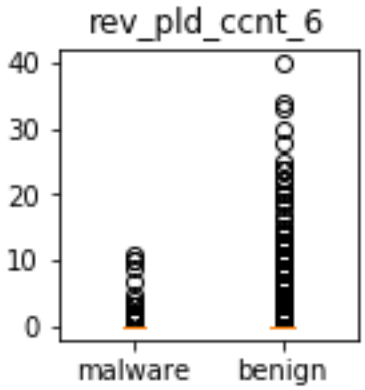

rev_pld_ccnt_1
200
150
100
50
0
malware
benign

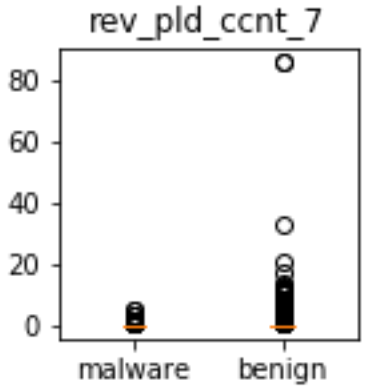

rev_pld_ccnt_2
100
75
50
25
0
malware
benign

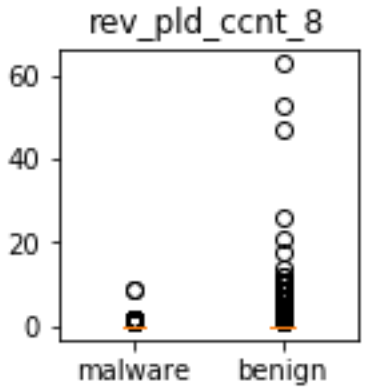

rev_pld_ccnt_3
75
50
25
0
malware
benign

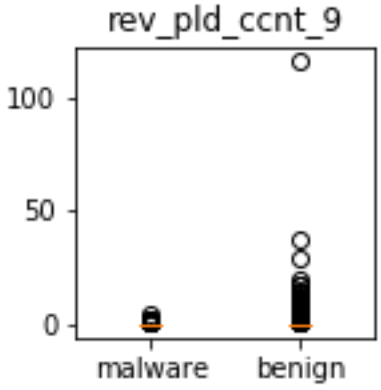

rev_pld_ccnt_4
150
100
50
0
malware
benign

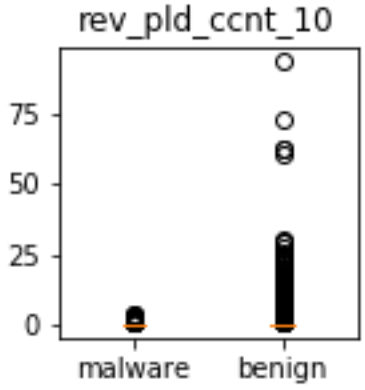

rev_pld_ccnt_5
60
40
20
0
malware
benign

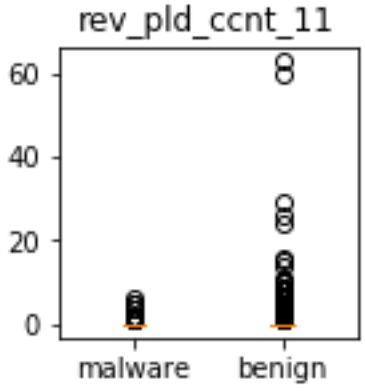

rev_pld_ccnt_6
40
30
20
10
0
malware
benign

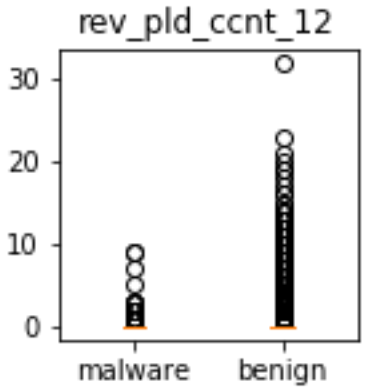

rev_pld_ccnt_7
80
60
40
20
0
malware
benign

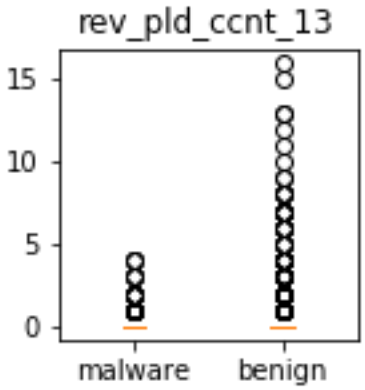

rev_pld_ccnt_8
60
40
20
0
malware
benign

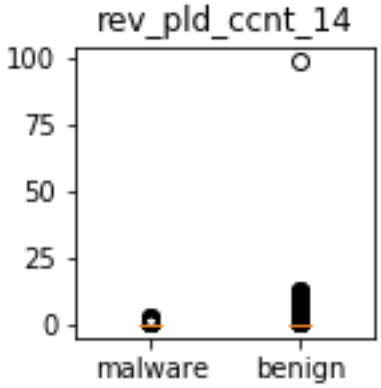

rev_pld_ccnt_9
100
50
0
malware
benign

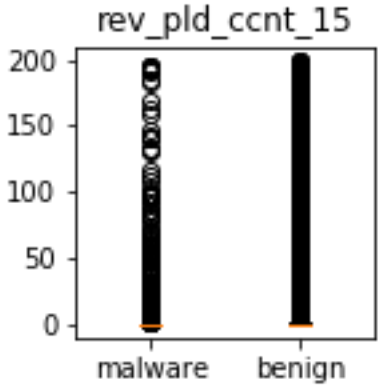

rev_pld_ccnt_10
75
50
25
0
malware
benign

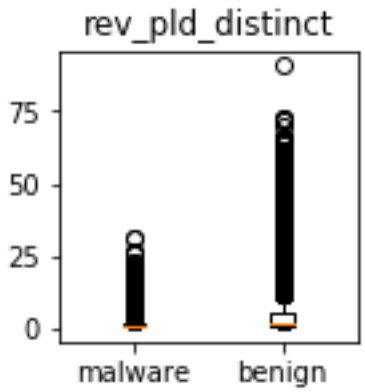

rev_pld_ccnt_11
60
40
20
0
malware
benign

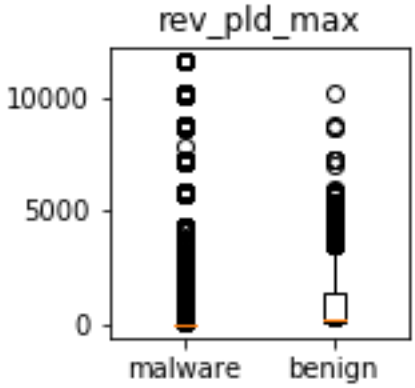

rev_pld_ccnt_12
30
20
10
0
malware
benign

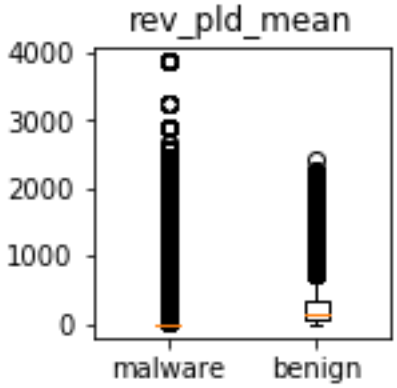

rev_pld_ccnt_13
15
10
5
0
malware
benign

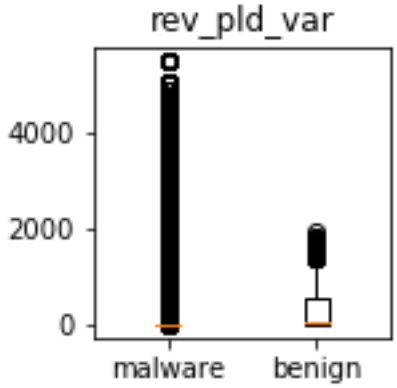

rev_pld_ccnt_14
100
75
50
25
0
malware
benign

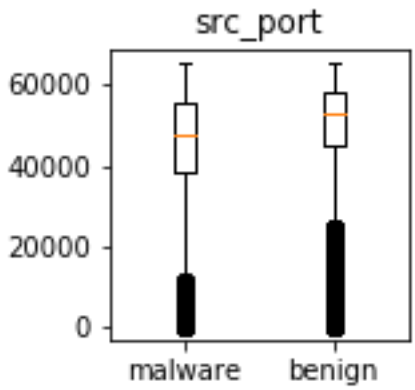

rev_pld_ccnt_15
200
150
100
50
0
malware
benign

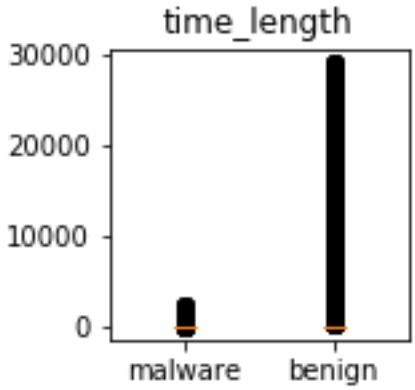

rev_pld_distinct
75
50
25
0
malware
benign

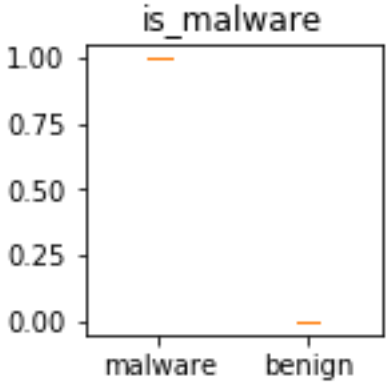

rev_pld_max
10000
5000
0
malware
benign


rev_pld_mean
4000
3000
2000
1000
0
malware
benign

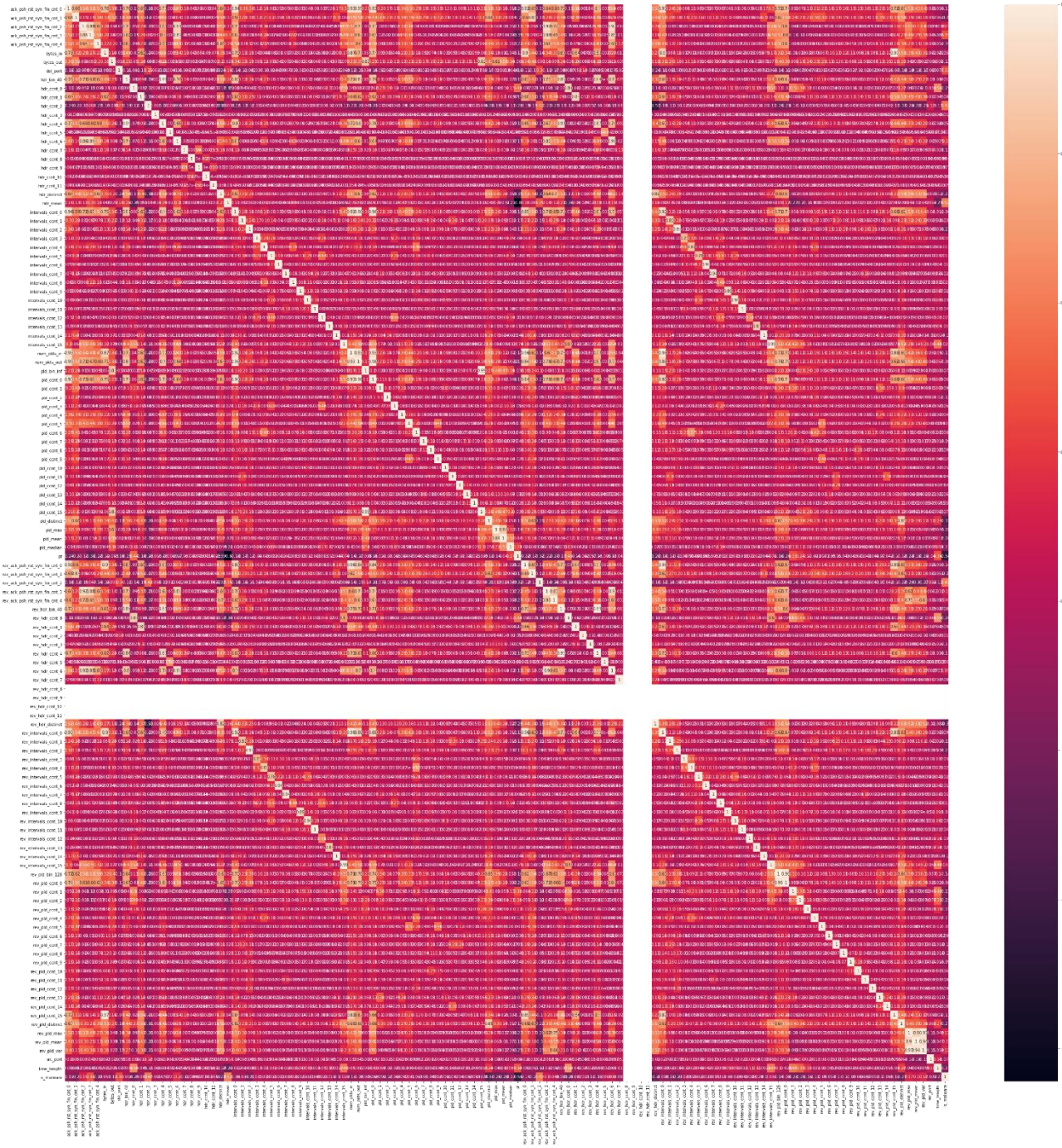

rev_pld_var
4000
2000
0
malware
benign


src_port
60000
40000
20000
0
malware
benign


time_length
30000
20000
10000
0
malware
benign

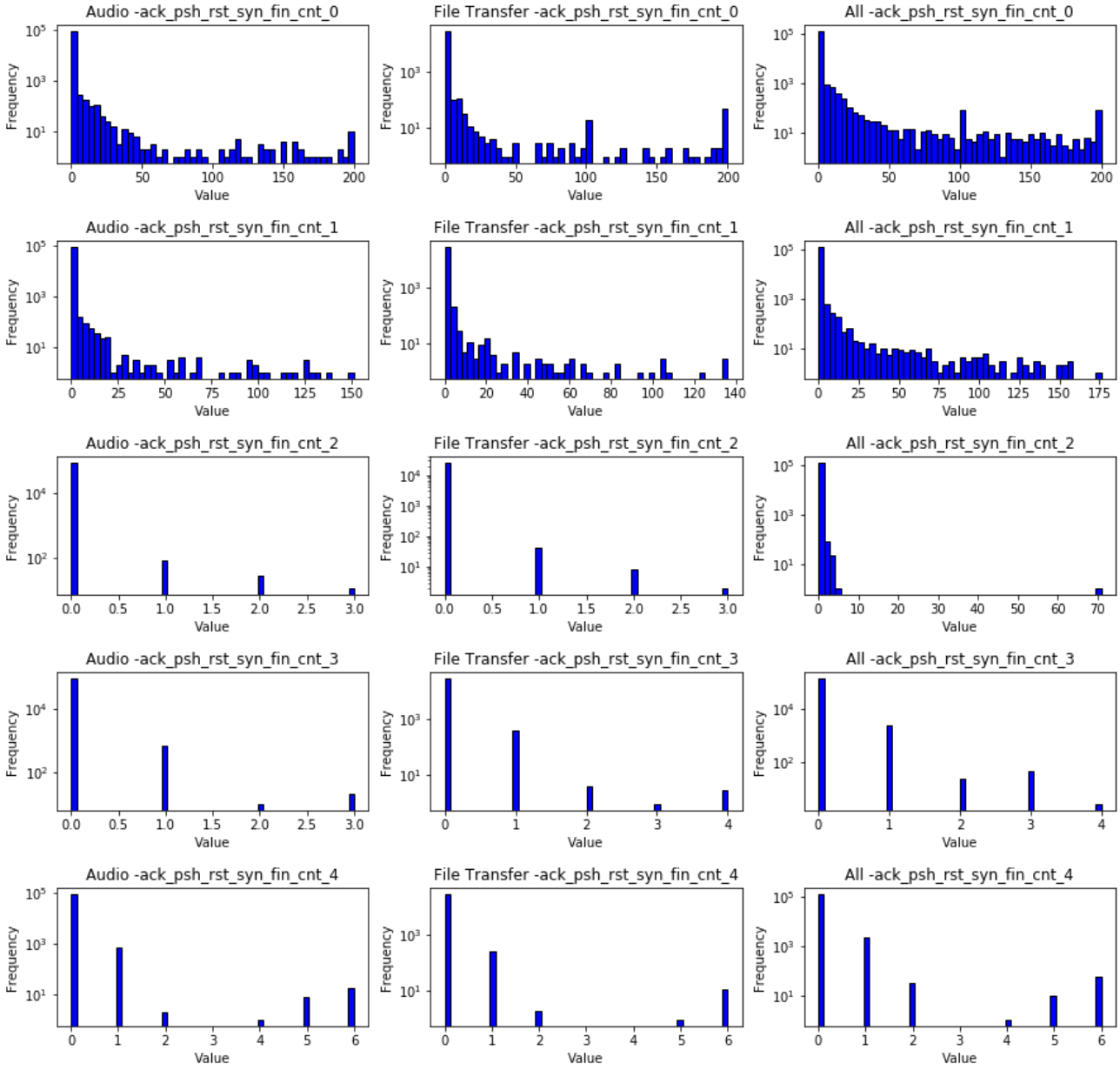

is_malware
1.00
0.75
0.50
0.25
0.00
malware
benign

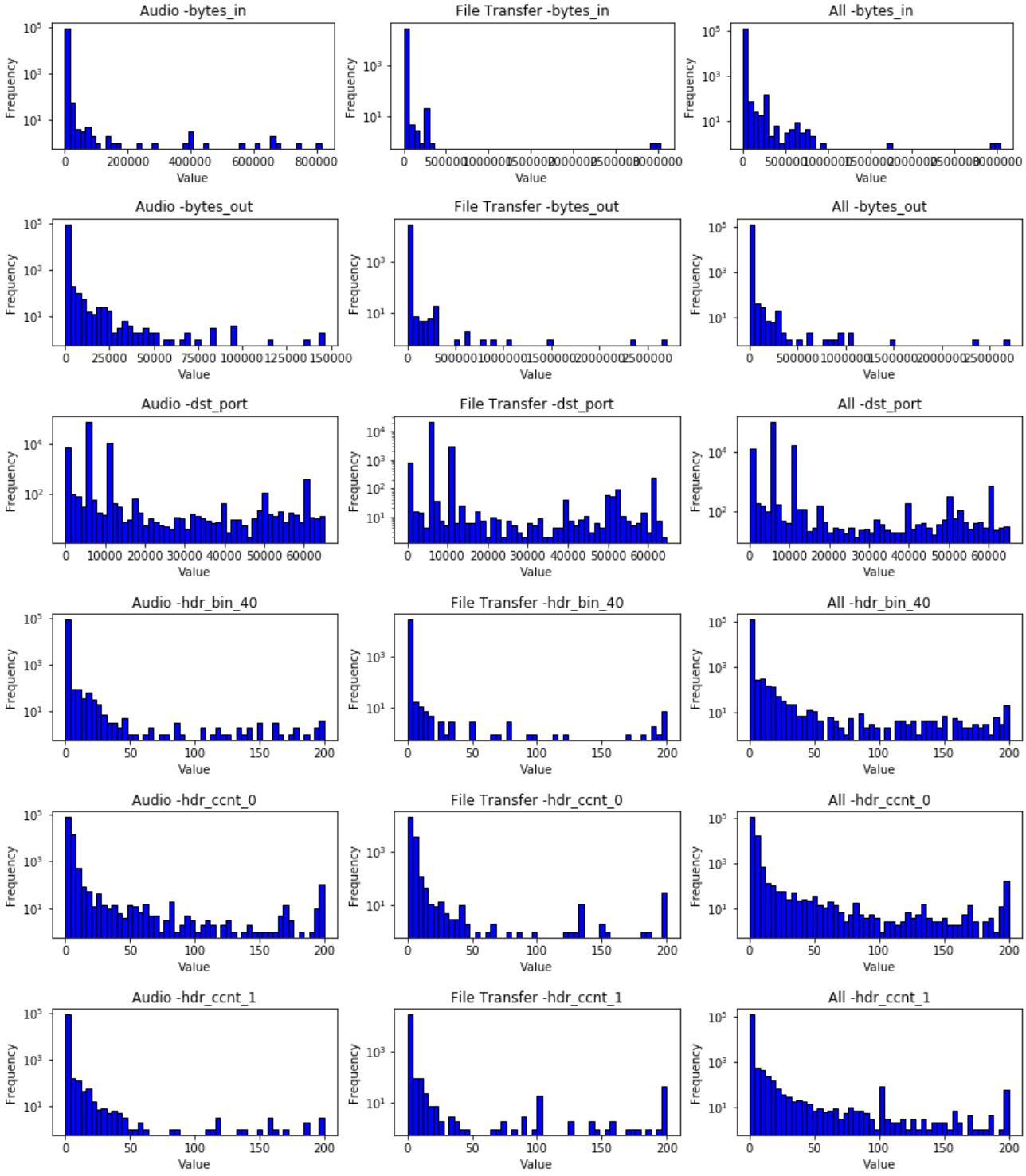
Bivariate Analysis

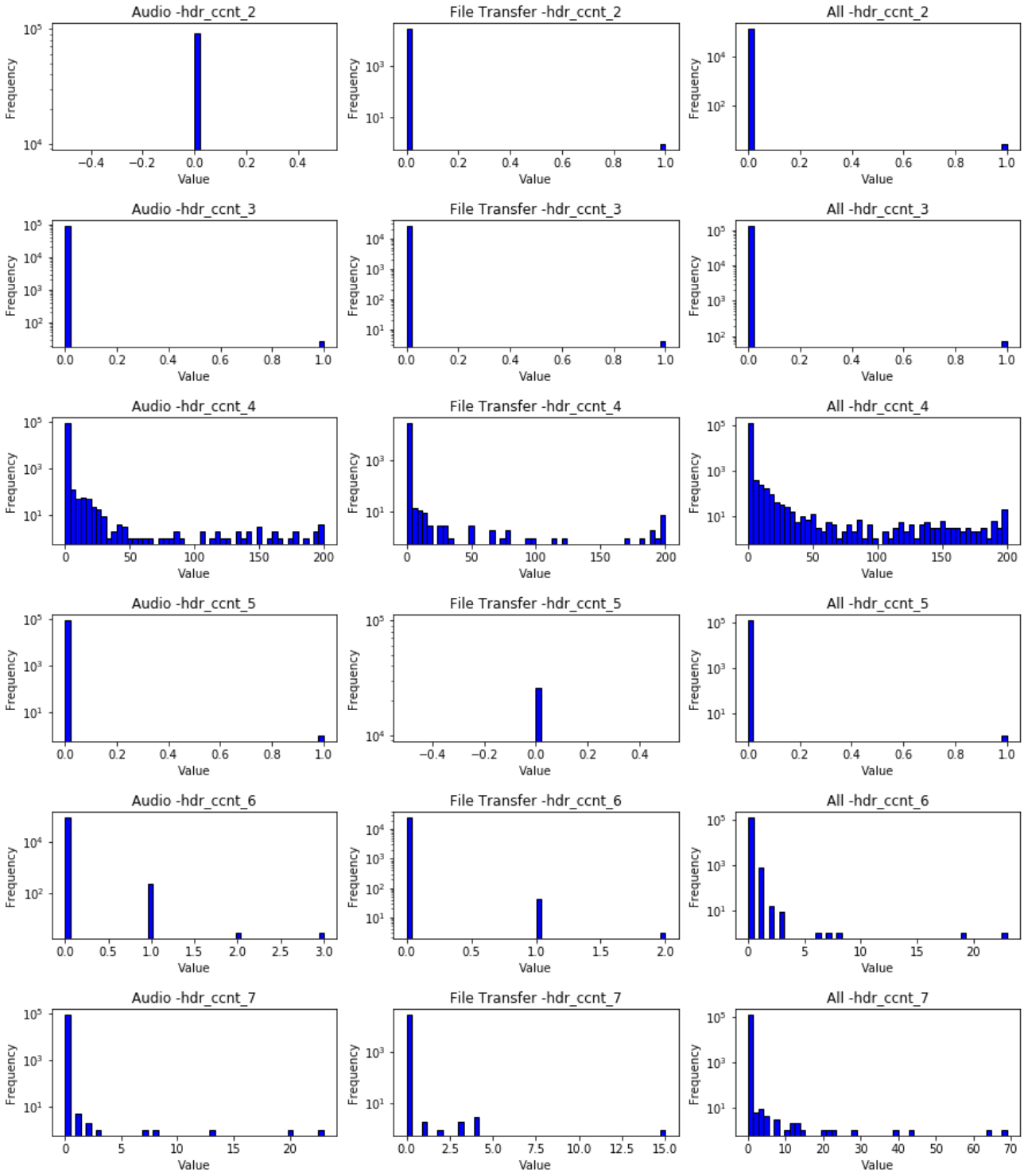


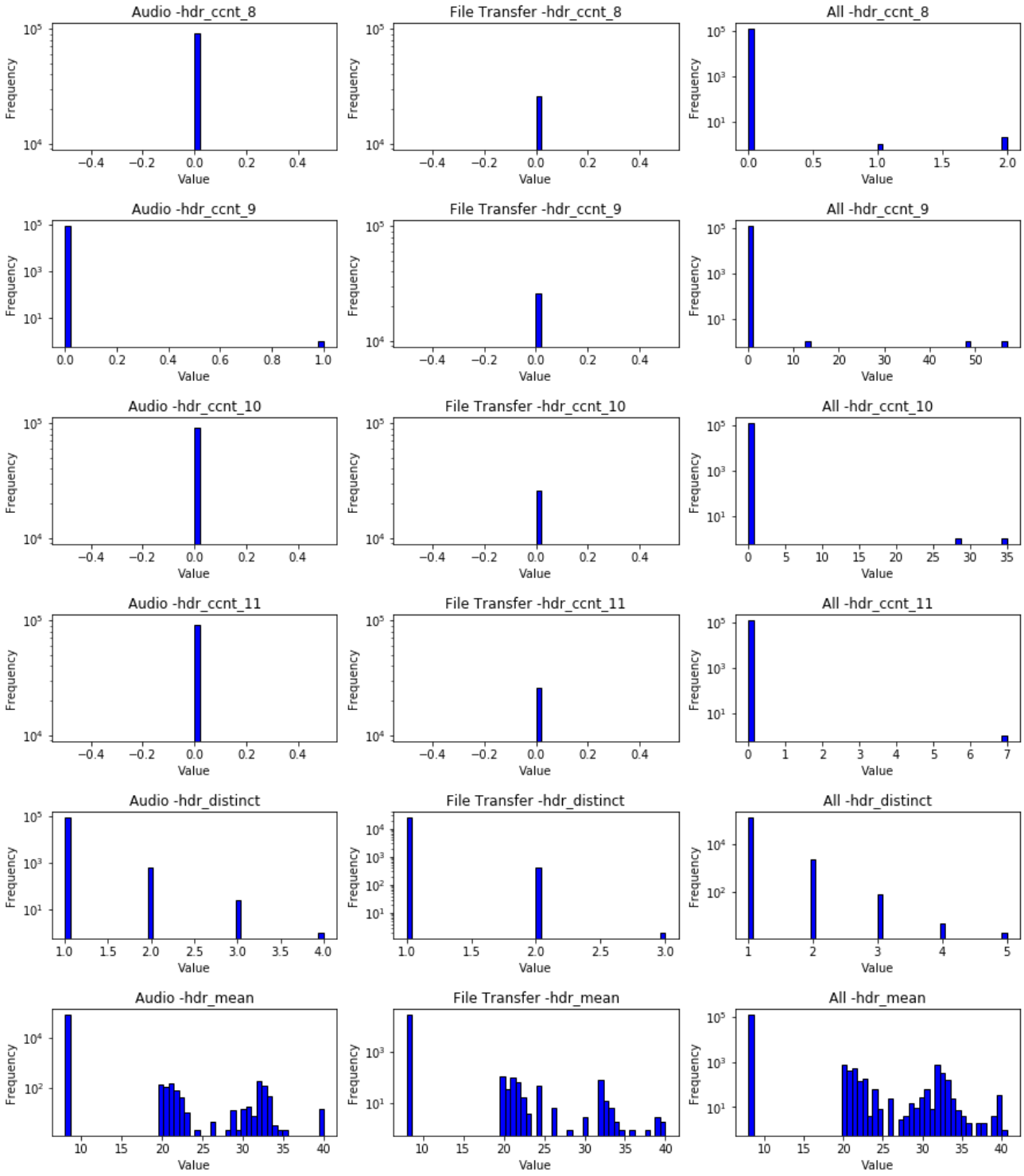
Non-Vpn Dataset

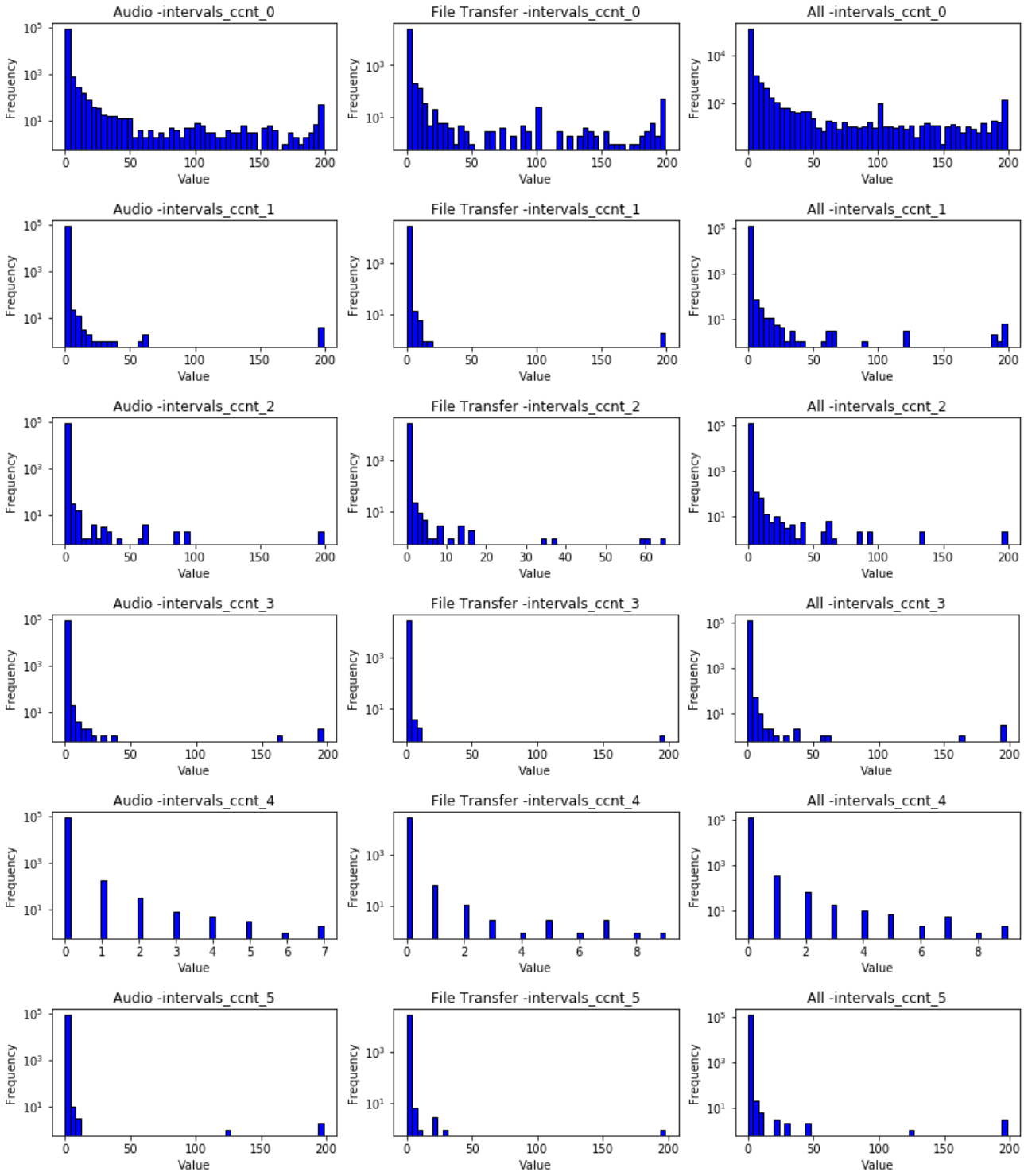
Univariate Analysis – Distribution Plots

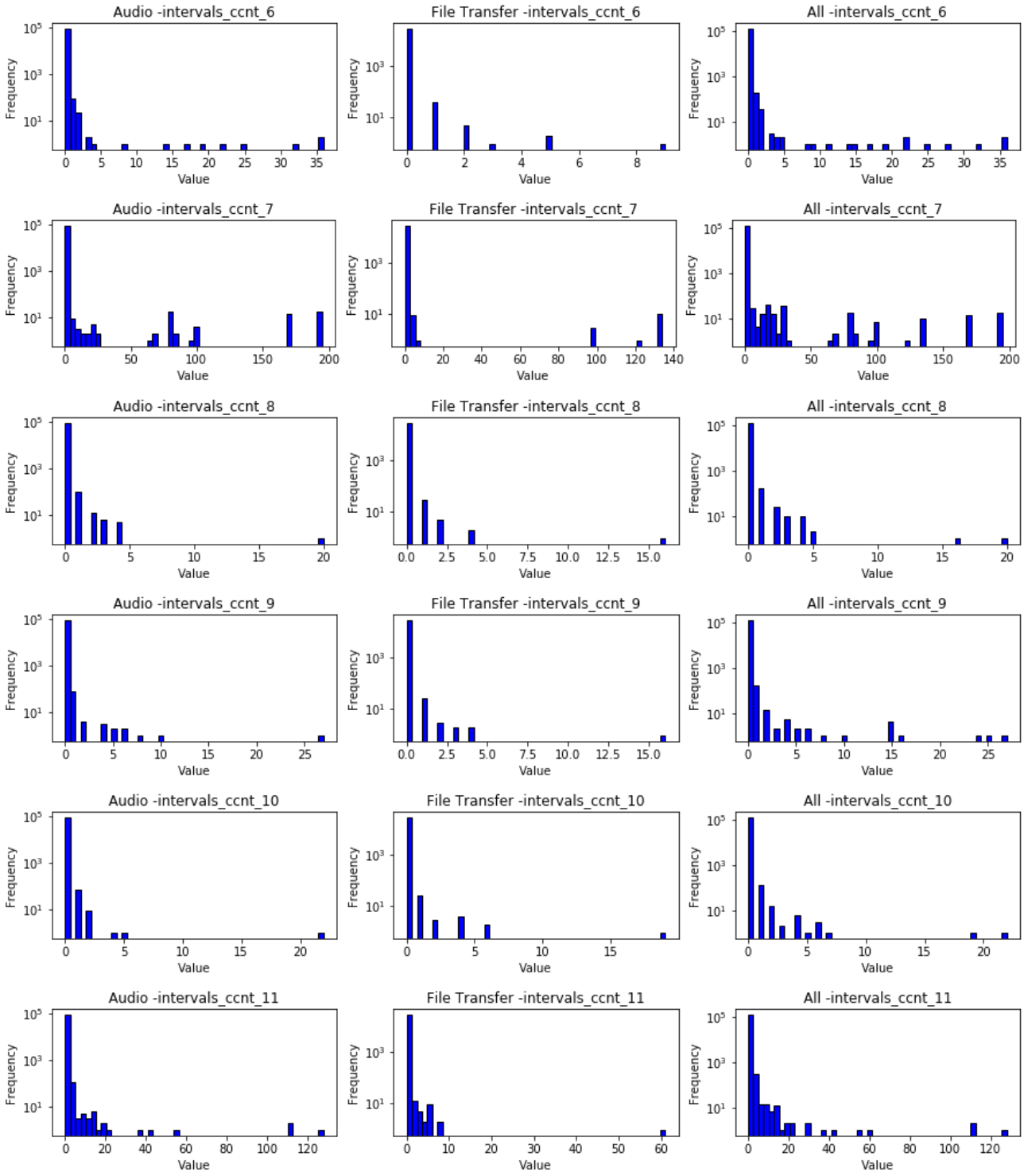


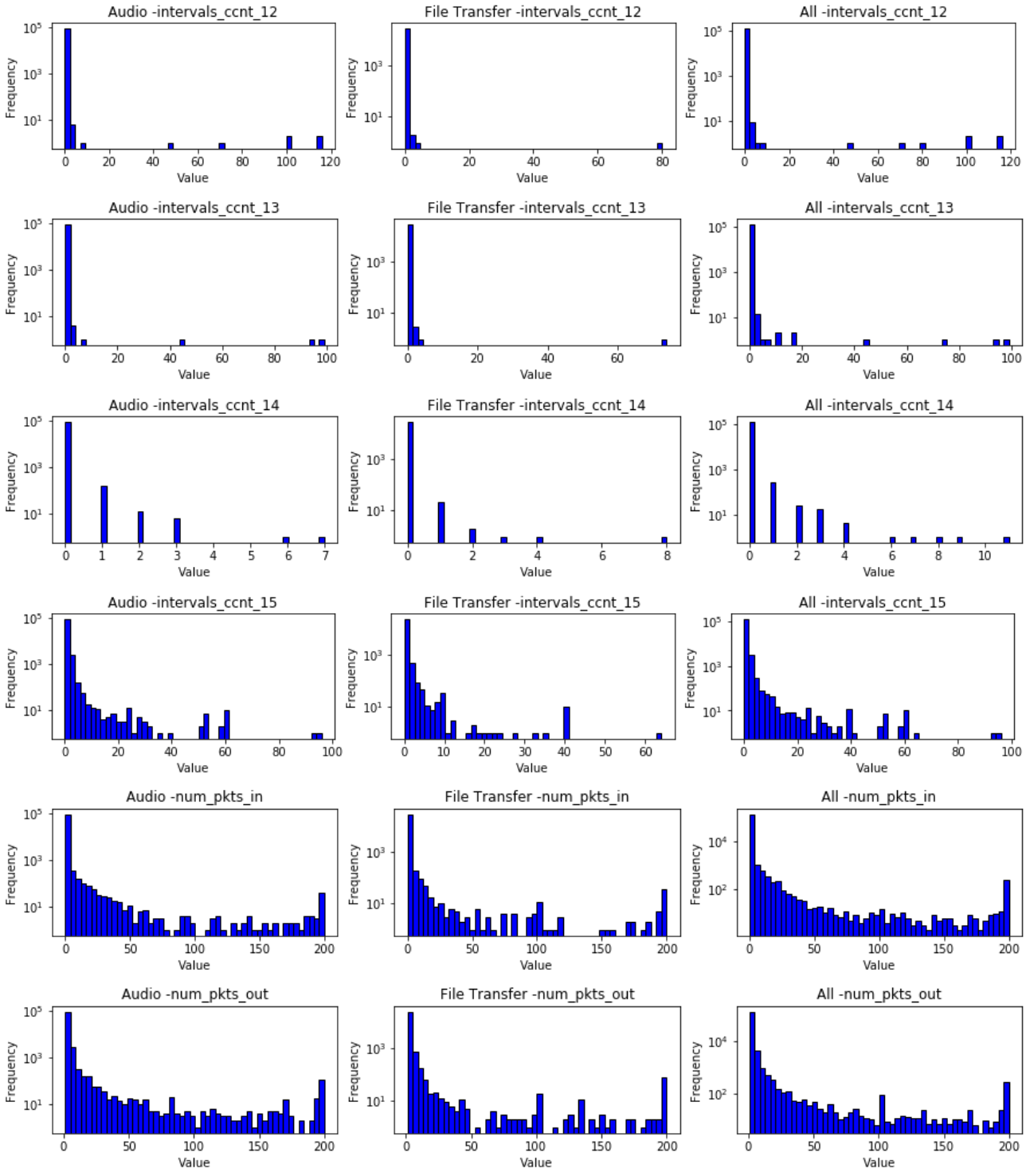

Audio -bytes_in
File Transfer -bytes_in
All -bytes_in
Audio -bytes_out
File Transfer -bytes_out
All -bytes_out
Audio -dst_port
File Transfer -dst_port
All -dst_port
Audio -hdr_bin_40
File Transfer -hdr_bin_40
All -hdr_bin_40
Audio -hdr_ccnt_0
File Transfer -hdr_ccnt_0
All -hdr_ccnt_0
Audio -hdr_ccnt_1
File Transfer -hdr_ccnt_1
All -hdr_ccnt_1
Frequency
Value

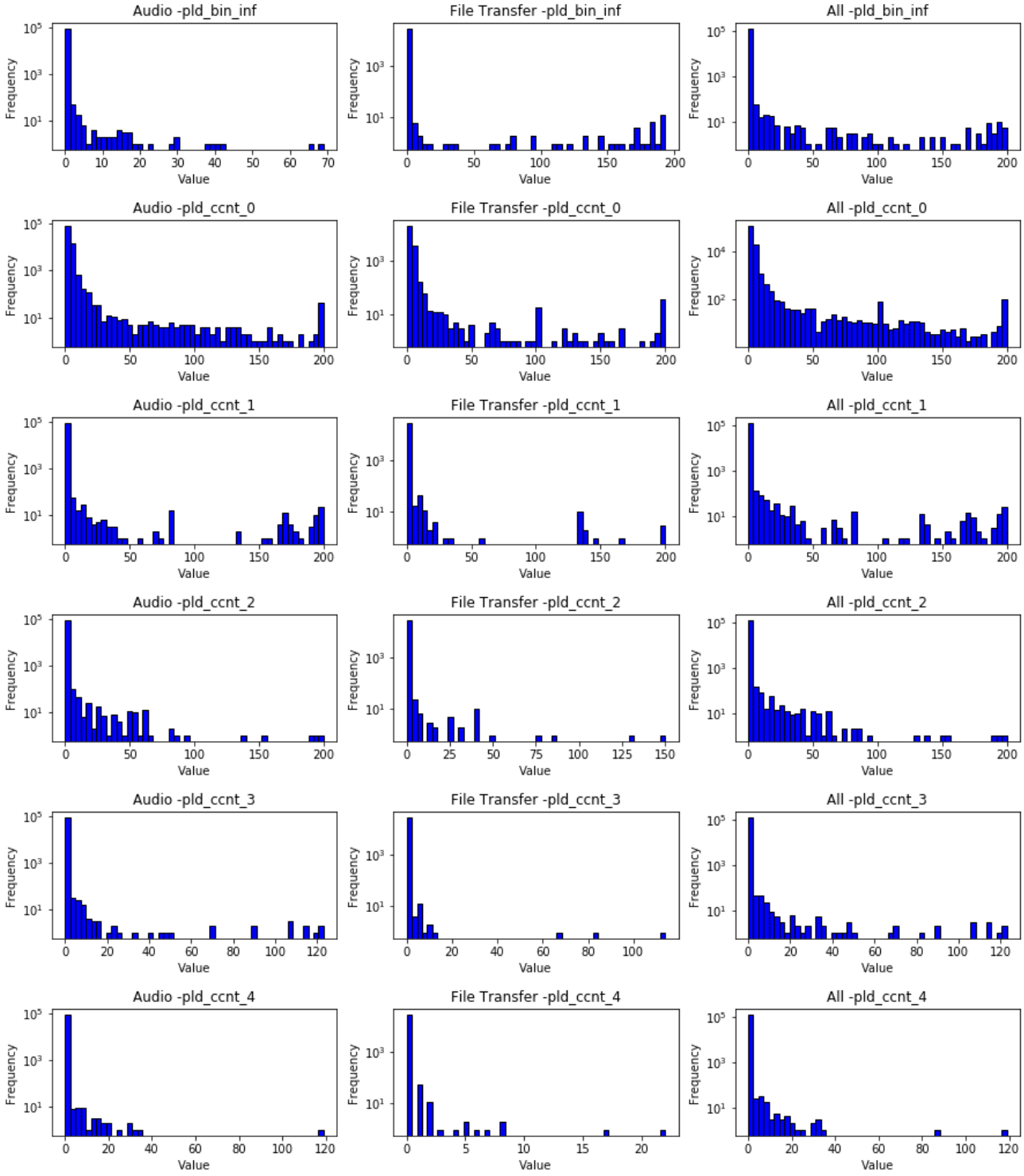

Audio -hdr_ccnt_2
File Transfer -hdr_ccnt_2
All -hdr_ccnt_2
Audio -hdr_ccnt_3
File Transfer -hdr_ccnt_3
All -hdr_ccnt_3
Audio -hdr_ccnt_4
File Transfer -hdr_ccnt_4
All -hdr_ccnt_4
Audio -hdr_ccnt_5
File Transfer -hdr_ccnt_5
All -hdr_ccnt_5
Audio -hdr_ccnt_6
File Transfer -hdr_ccnt_6
All -hdr_ccnt_6
Audio -hdr_ccnt_7
File Transfer -hdr_ccnt_7
All -hdr_ccnt_7
Frequency
Value

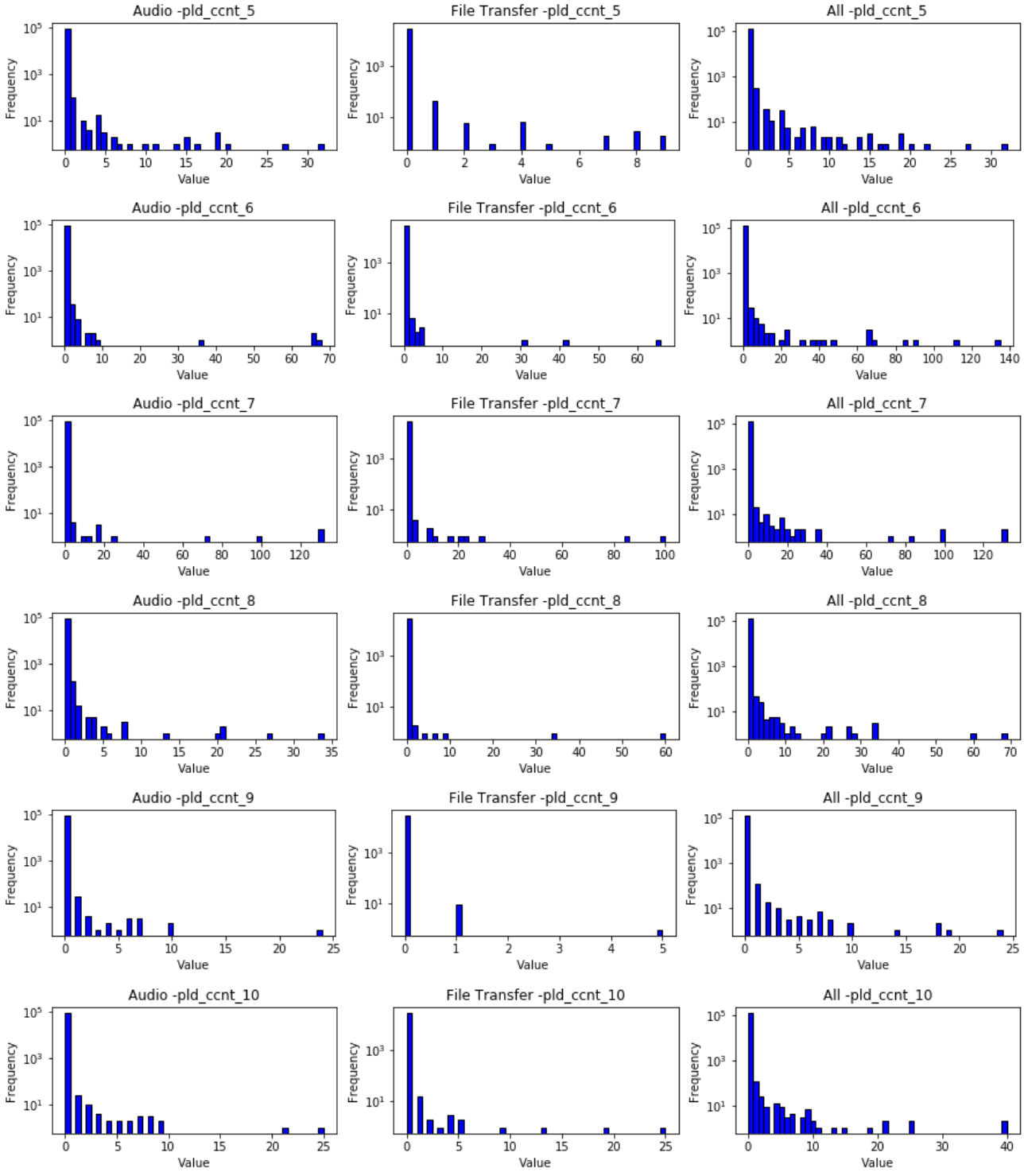

Audio -hdr_ccnt_8
File Transfer -hdr_ccnt_8
All -hdr_ccnt_8
Audio -hdr_ccnt_9
File Transfer -hdr_ccnt_9
All -hdr_ccnt_9
Audio -hdr_ccnt_10
File Transfer -hdr_ccnt_10
All -hdr_ccnt_10
Audio -hdr_ccnt_11
File Transfer -hdr_ccnt_11
All -hdr_ccnt_11
Audio -hdr_distinct
File Transfer -hdr_distinct
All -hdr_distinct
Audio -hdr_mean
File Transfer -hdr_mean
All -hdr_mean
Frequency
Value

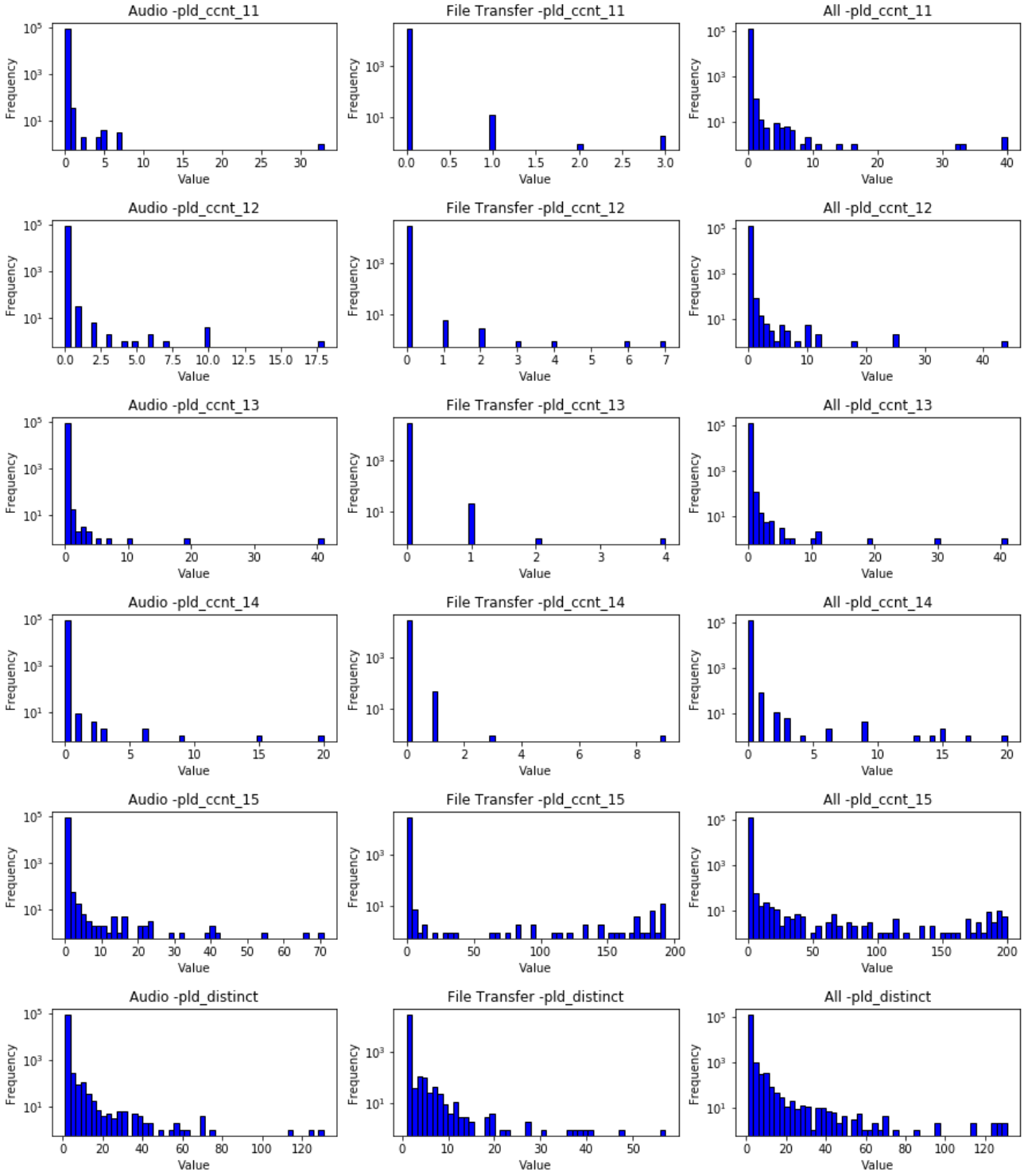

Audio -intervals_ccnt_0
File Transfer -intervals_ccnt_0
All -intervals_ccnt_0
Audio -intervals_ccnt_1
File Transfer -intervals_ccnt_1
All -intervals_ccnt_1
Audio -intervals_ccnt_2
File Transfer -intervals_ccnt_2
All -intervals_ccnt_2
Audio -intervals_ccnt_3
File Transfer -intervals_ccnt_3
All -intervals_ccnt_3
Audio -intervals_ccnt_4
File Transfer -intervals_ccnt_4
All -intervals_ccnt_4
Audio -intervals_ccnt_5
File Transfer -intervals_ccnt_5
All -intervals_ccnt_5
Frequency
Value

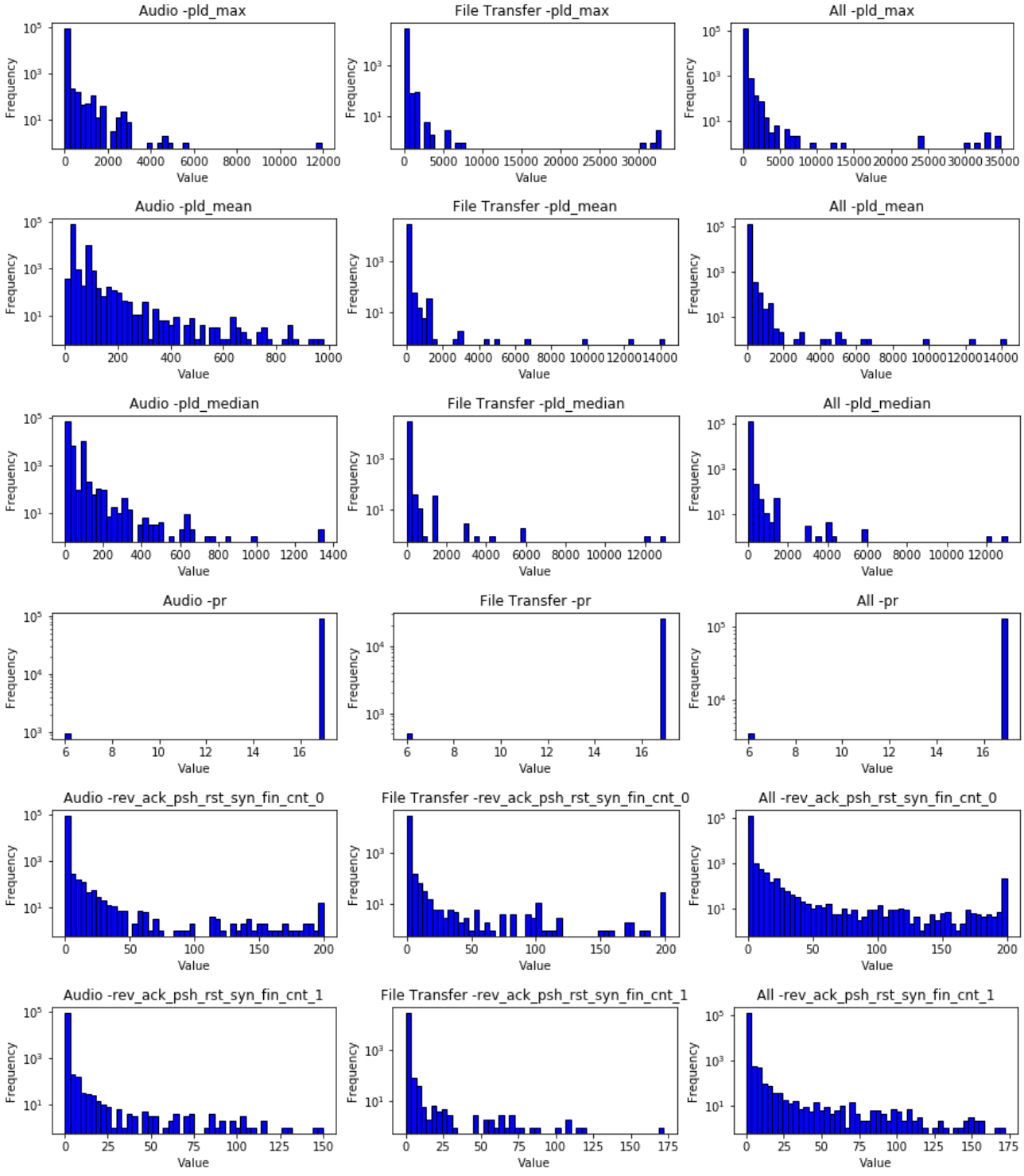

Audio -intervals_ccnt_6
File Transfer -intervals_ccnt_6
All -intervals_ccnt_6
Audio -intervals_ccnt_7
File Transfer -intervals_ccnt_7
All -intervals_ccnt_7
Audio -intervals_ccnt_8
File Transfer -intervals_ccnt_8
All -intervals_ccnt_8
Audio -intervals_ccnt_9
File Transfer -intervals_ccnt_9
All -intervals_ccnt_9
Audio -intervals_ccnt_10
File Transfer -intervals_ccnt_10
All -intervals_ccnt_10
Audio -intervals_ccnt_11
File Transfer -intervals_ccnt_11
All -intervals_ccnt_11
Frequency
Value

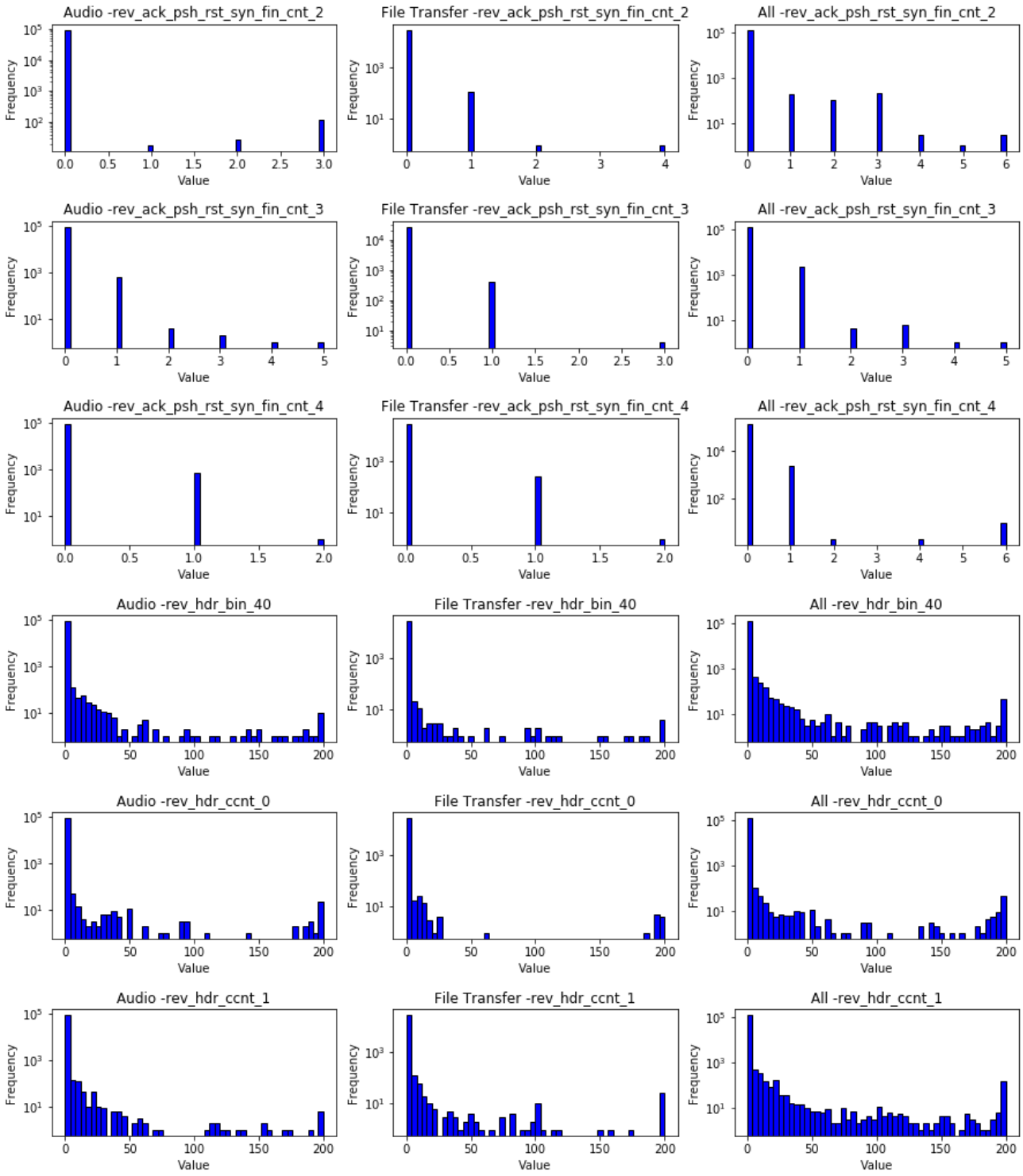

Audio -intervals_ccnt_12
File Transfer -intervals_ccnt_12
All -intervals_ccnt_12
Audio -intervals_ccnt_13
File Transfer -intervals_ccnt_13
All -intervals_ccnt_13
Audio -intervals_ccnt_14
File Transfer -intervals_ccnt_14
All -intervals_ccnt_14
Audio -intervals_ccnt_15
File Transfer -intervals_ccnt_15
All -intervals_ccnt_15
Audio -num_pkts_in
File Transfer -num_pkts_in
All -num_pkts_in
Audio -num_pkts_out
File Transfer -num_pkts_out
All -num_pkts_out
Frequency
Value

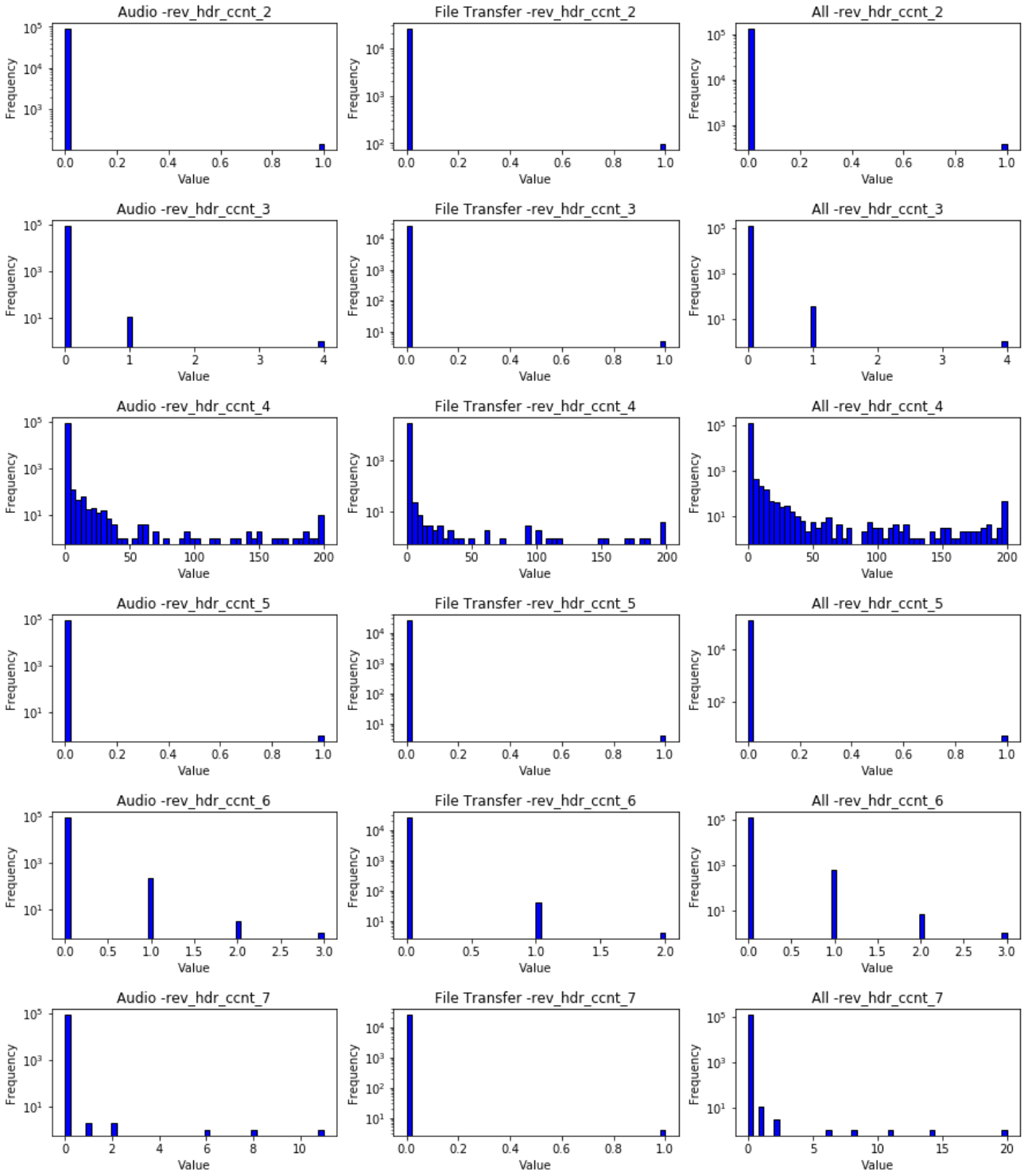

Audio -pld_bin_inf
File Transfer -pld_bin_inf
All -pld_bin_inf
Audio -pld_ccnt_0
File Transfer -pld_ccnt_0
All -pld_ccnt_0
Audio -pld_ccnt_1
File Transfer -pld_ccnt_1
All -pld_ccnt_1
Audio -pld_ccnt_2
File Transfer -pld_ccnt_2
All -pld_ccnt_2
Audio -pld_ccnt_3
File Transfer -pld_ccnt_3
All -pld_ccnt_3
Audio -pld_ccnt_4
File Transfer -pld_ccnt_4
All -pld_ccnt_4
Frequency
Value

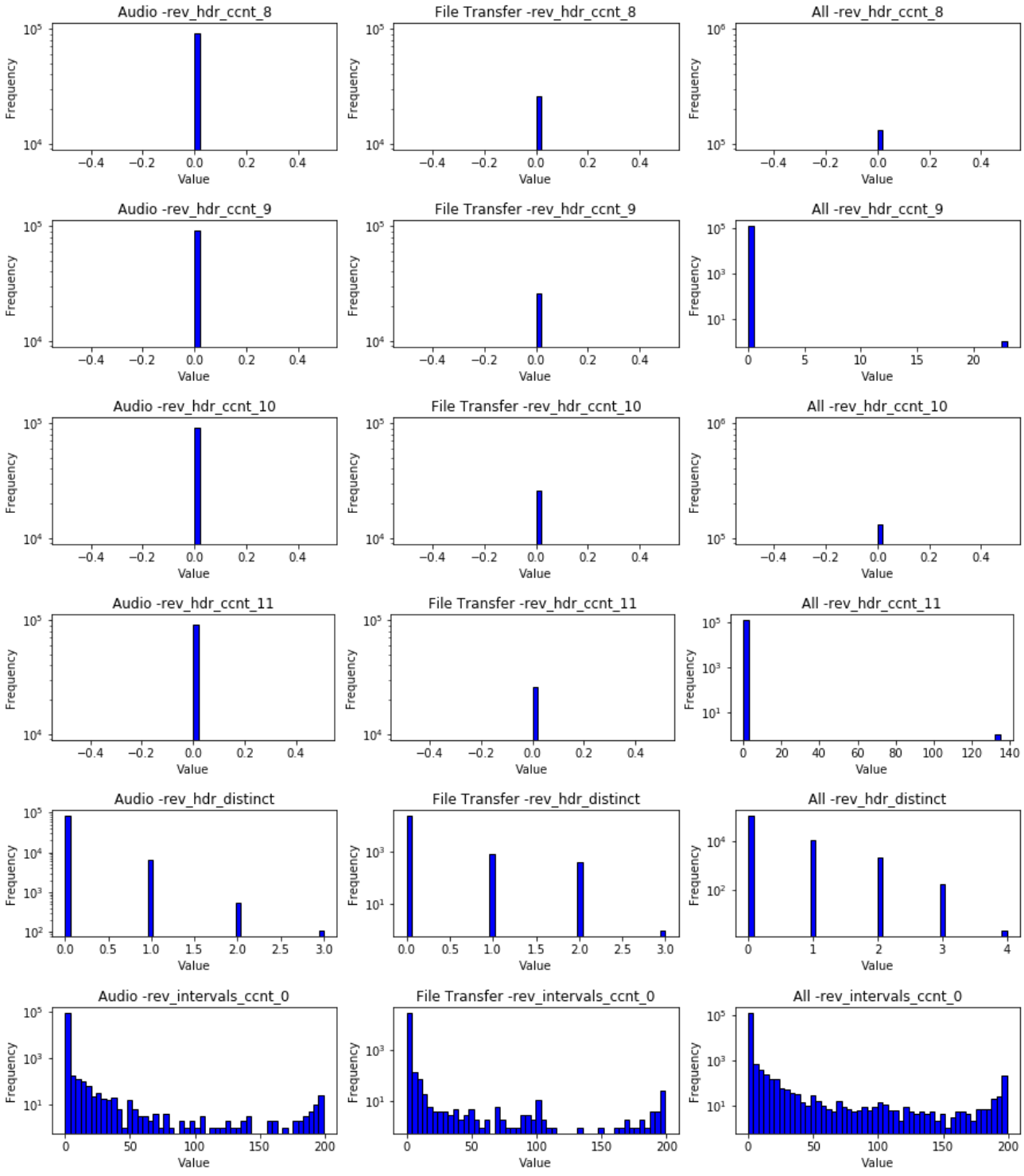

Audio -pld_ccnt_5
File Transfer -pld_ccnt_5
All -pld_ccnt_5
Audio -pld_ccnt_6
File Transfer -pld_ccnt_6
All -pld_ccnt_6
Audio -pld_ccnt_7
File Transfer -pld_ccnt_7
All -pld_ccnt_7
Audio -pld_ccnt_8
File Transfer -pld_ccnt_8
All -pld_ccnt_8
Audio -pld_ccnt_9
File Transfer -pld_ccnt_9
All -pld_ccnt_9
Audio -pld_ccnt_10
File Transfer -pld_ccnt_10
All -pld_ccnt_10
Frequency
Value

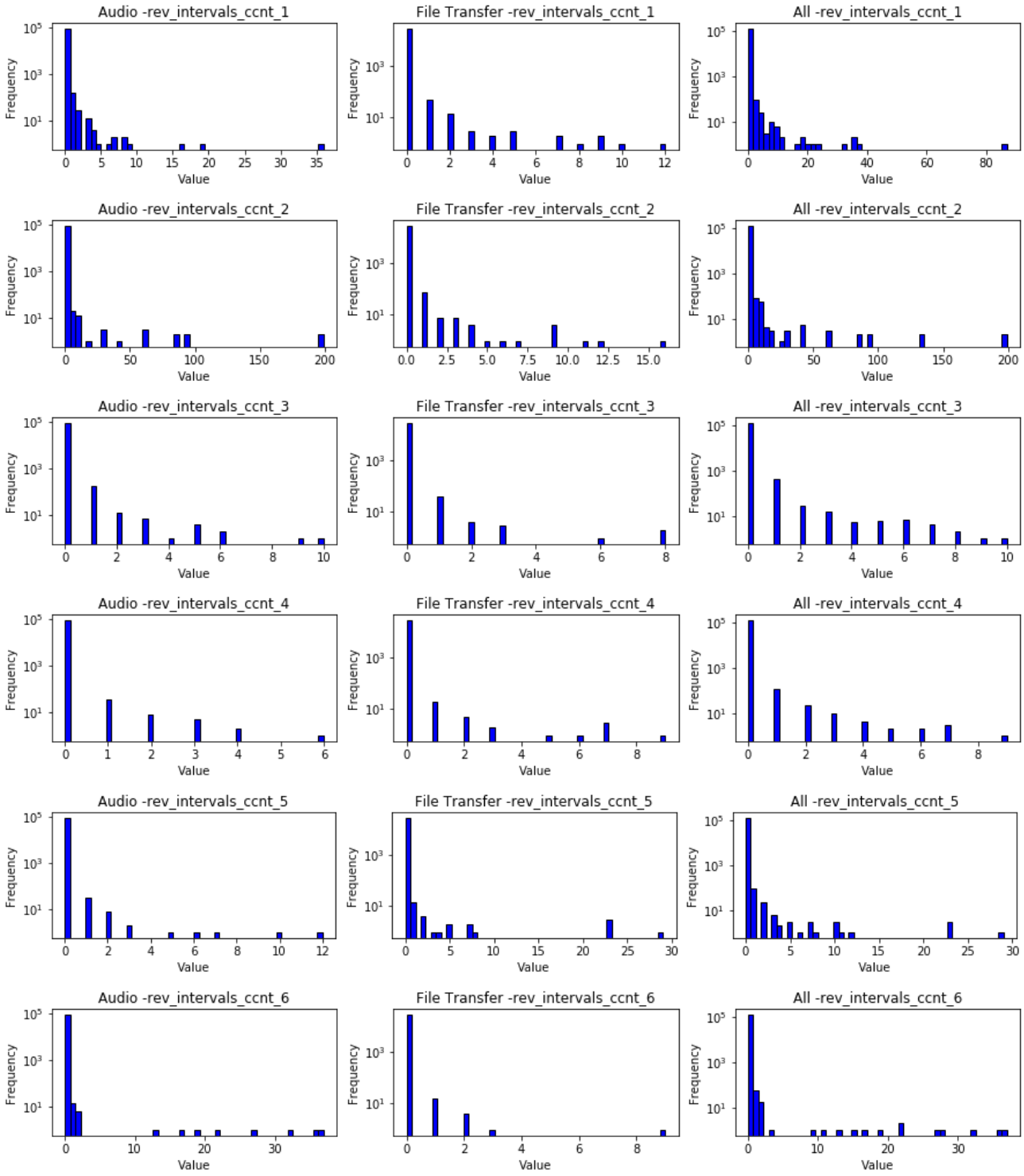

Audio -pld_ccnt_11
File Transfer -pld_ccnt_11
All -pld_ccnt_11
Audio -pld_ccnt_12
File Transfer -pld_ccnt_12
All -pld_ccnt_12
Audio -pld_ccnt_13
File Transfer -pld_ccnt_13
All -pld_ccnt_13
Audio -pld_ccnt_14
File Transfer -pld_ccnt_14
All -pld_ccnt_14
Audio -pld_ccnt_15
File Transfer -pld_ccnt_15
All -pld_ccnt_15
Audio -pld_distinct
File Transfer -pld_distinct
All -pld_distinct
Frequency
Value

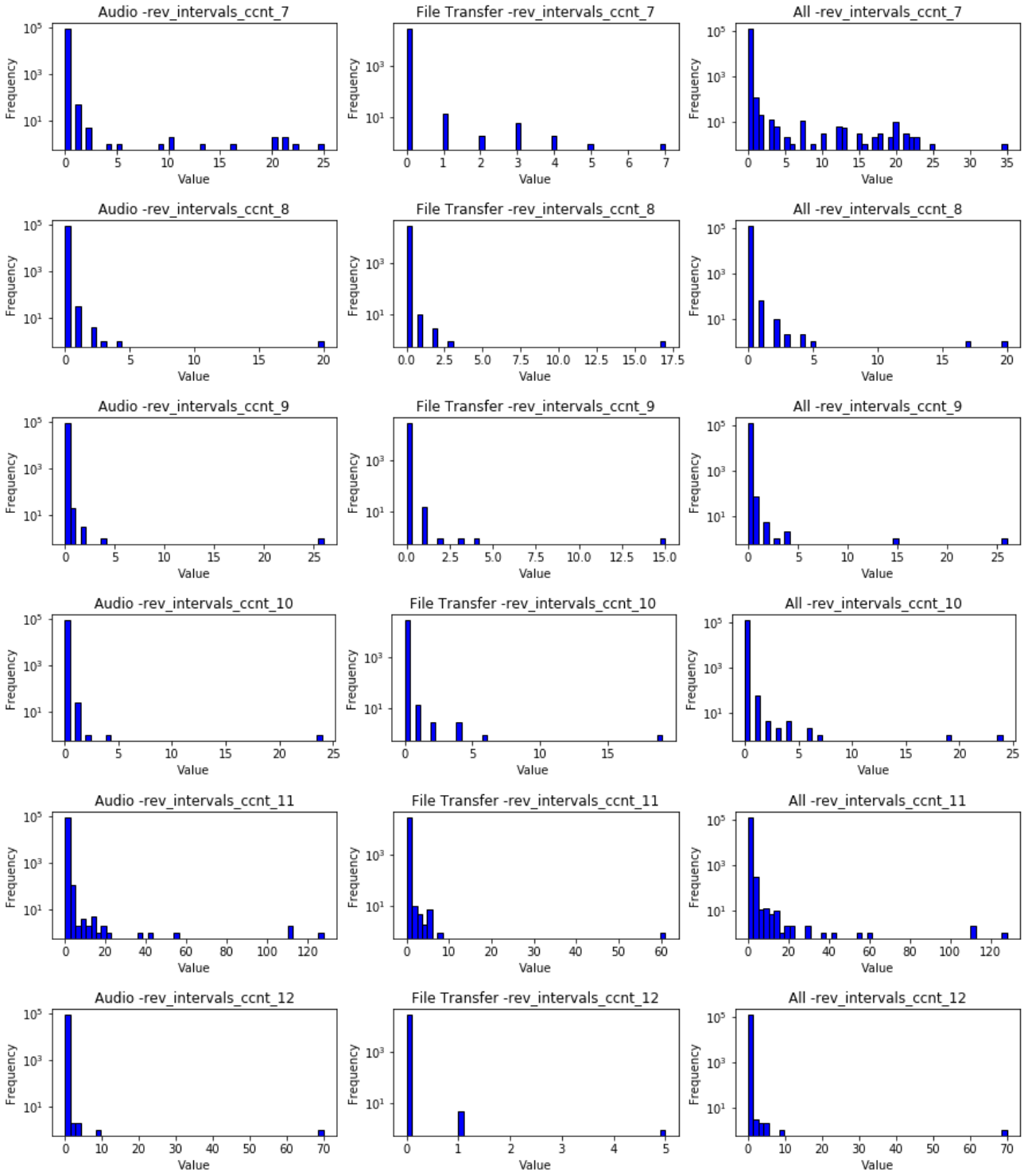

Audio -pld_max
File Transfer -pld_max
All -pld_max
Audio -pld_mean
File Transfer -pld_mean
All -pld_mean
Audio -pld_median
File Transfer -pld_median
All -pld_median
Audio -pr
File Transfer -pr
All -pr
Audio -rev_ack_psh_rst_syn_fin_cnt_0
File Transfer -rev_ack_psh_rst_syn_fin_cnt_0
All -rev_ack_psh_rst_syn_fin_cnt_0
Audio -rev_ack_psh_rst_syn_fin_cnt_1
File Transfer -rev_ack_psh_rst_syn_fin_cnt_1
All -rev_ack_psh_rst_syn_fin_cnt_1
Frequency
Value

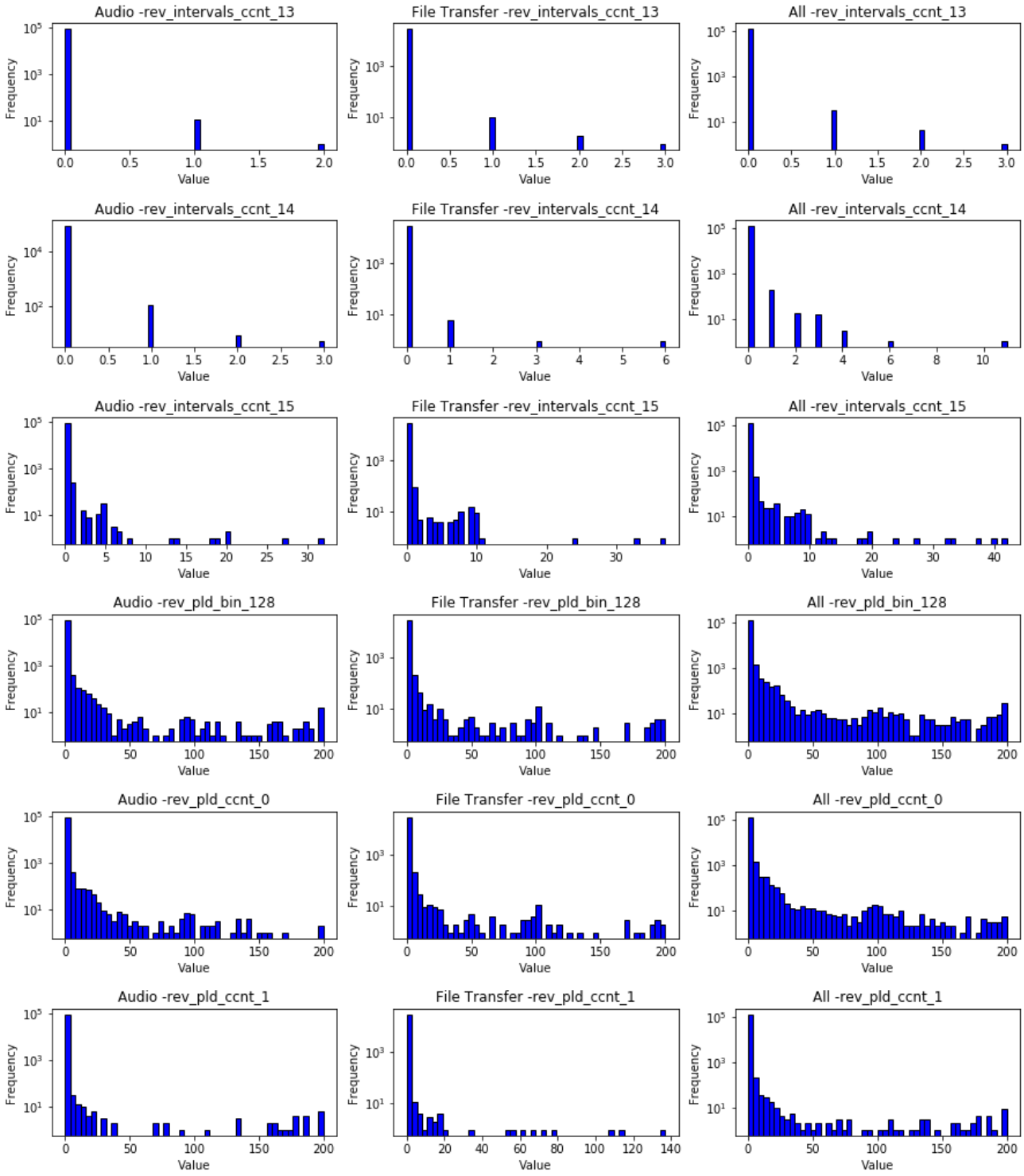

Audio -rev_ack_psh_rst_syn_fin_cnt_2
File Transfer -rev_ack_psh_rst_syn_fin_cnt_2
All -rev_ack_psh_rst_syn_fin_cnt_2
Audio -rev_ack_psh_rst_syn_fin_cnt_3
File Transfer -rev_ack_psh_rst_syn_fin_cnt_3
All -rev_ack_psh_rst_syn_fin_cnt_3
Audio -rev_ack_psh_rst_syn_fin_cnt_4
File Transfer -rev_ack_psh_rst_syn_fin_cnt_4
All -rev_ack_psh_rst_syn_fin_cnt_4
Audio -rev_hdr_bin_40
File Transfer -rev_hdr_bin_40
All -rev_hdr_bin_40
Audio -rev_hdr_ccnt_0
File Transfer -rev_hdr_ccnt_0
All -rev_hdr_ccnt_0
Audio -rev_hdr_ccnt_1
File Transfer -rev_hdr_ccnt_1
All -rev_hdr_ccnt_1
Frequency
Value

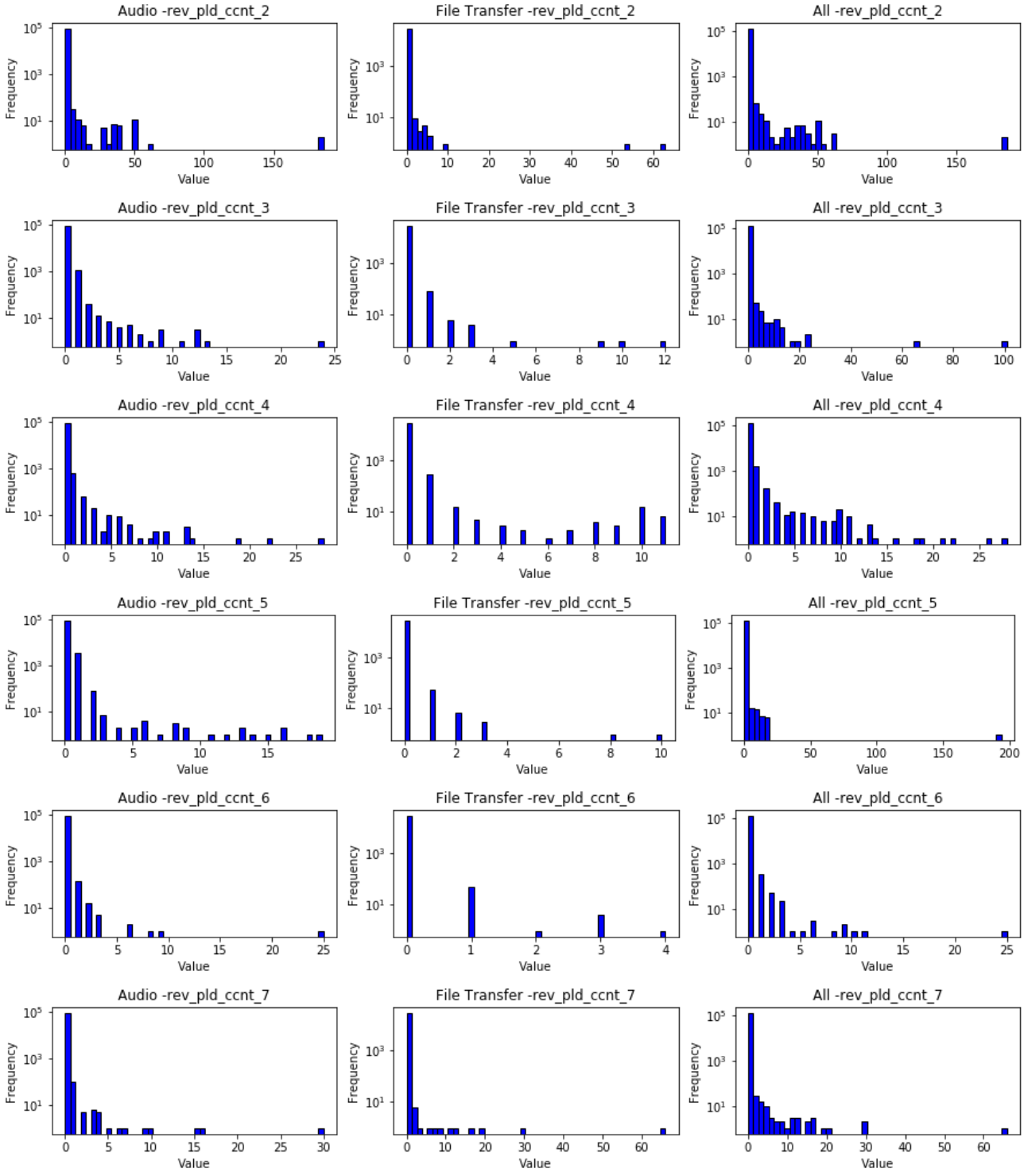

Audio -rev_hdr_ccnt_2
File Transfer -rev_hdr_ccnt_2
All -rev_hdr_ccnt_2
Audio -rev_hdr_ccnt_3
File Transfer -rev_hdr_ccnt_3
All -rev_hdr_ccnt_3
Audio -rev_hdr_ccnt_4
File Transfer -rev_hdr_ccnt_4
All -rev_hdr_ccnt_4
Audio -rev_hdr_ccnt_5
File Transfer -rev_hdr_ccnt_5
All -rev_hdr_ccnt_5
Audio -rev_hdr_ccnt_6
File Transfer -rev_hdr_ccnt_6
All -rev_hdr_ccnt_6
Audio -rev_hdr_ccnt_7
File Transfer -rev_hdr_ccnt_7
All -rev_hdr_ccnt_7
Frequency
Value

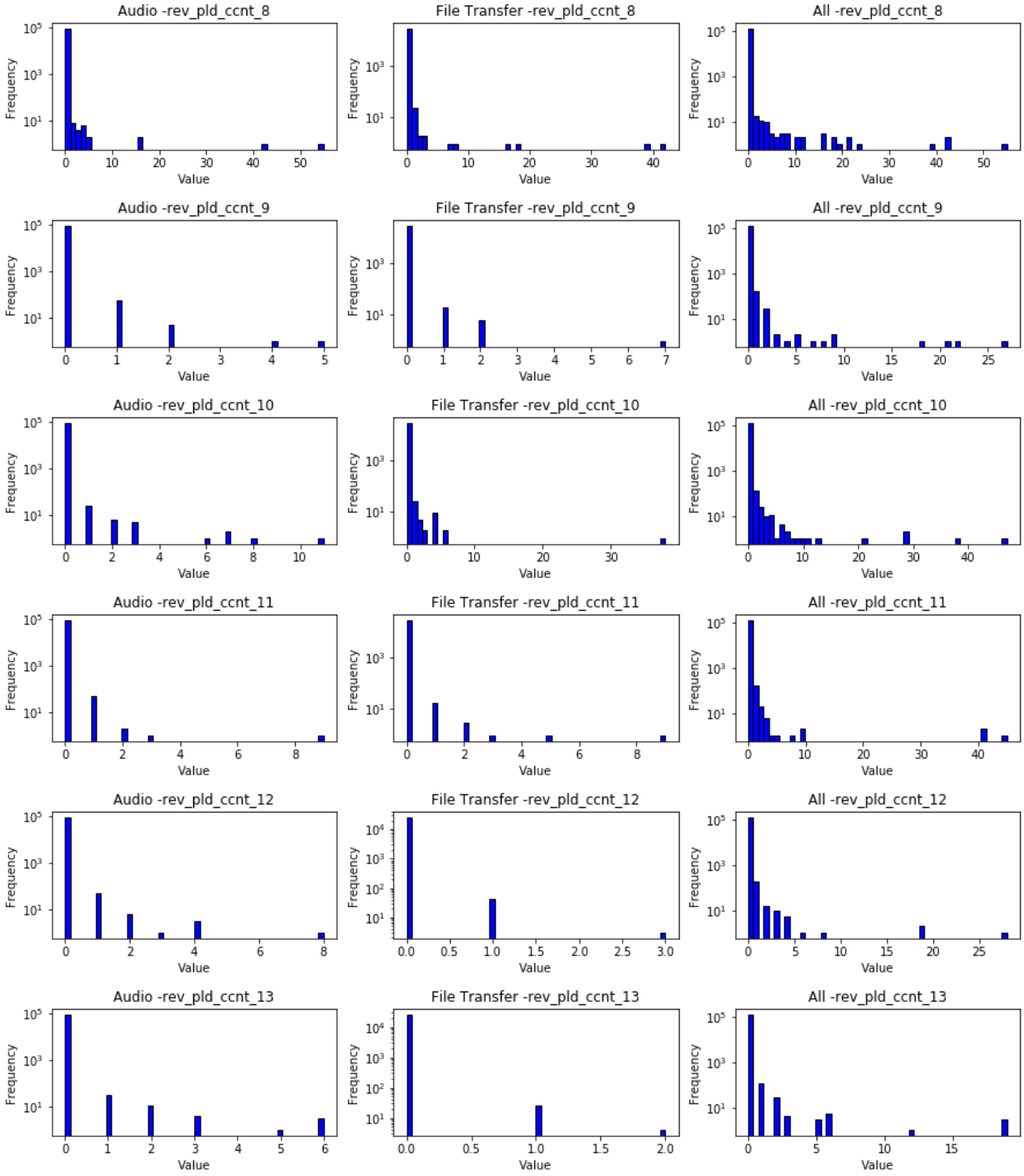

Audio -rev_hdr_ccnt_8
File Transfer -rev_hdr_ccnt_8
All -rev_hdr_ccnt_8
Audio -rev_hdr_ccnt_9
File Transfer -rev_hdr_ccnt_9
All -rev_hdr_ccnt_9
Audio -rev_hdr_ccnt_10
File Transfer -rev_hdr_ccnt_10
All -rev_hdr_ccnt_10
Audio -rev_hdr_ccnt_11
File Transfer -rev_hdr_ccnt_11
All -rev_hdr_ccnt_11
Audio -rev_hdr_distinct
File Transfer -rev_hdr_distinct
All -rev_hdr_distinct
Audio -rev_intervals_ccnt_0
File Transfer -rev_intervals_ccnt_0
All -rev_intervals_ccnt_0
Frequency
Value

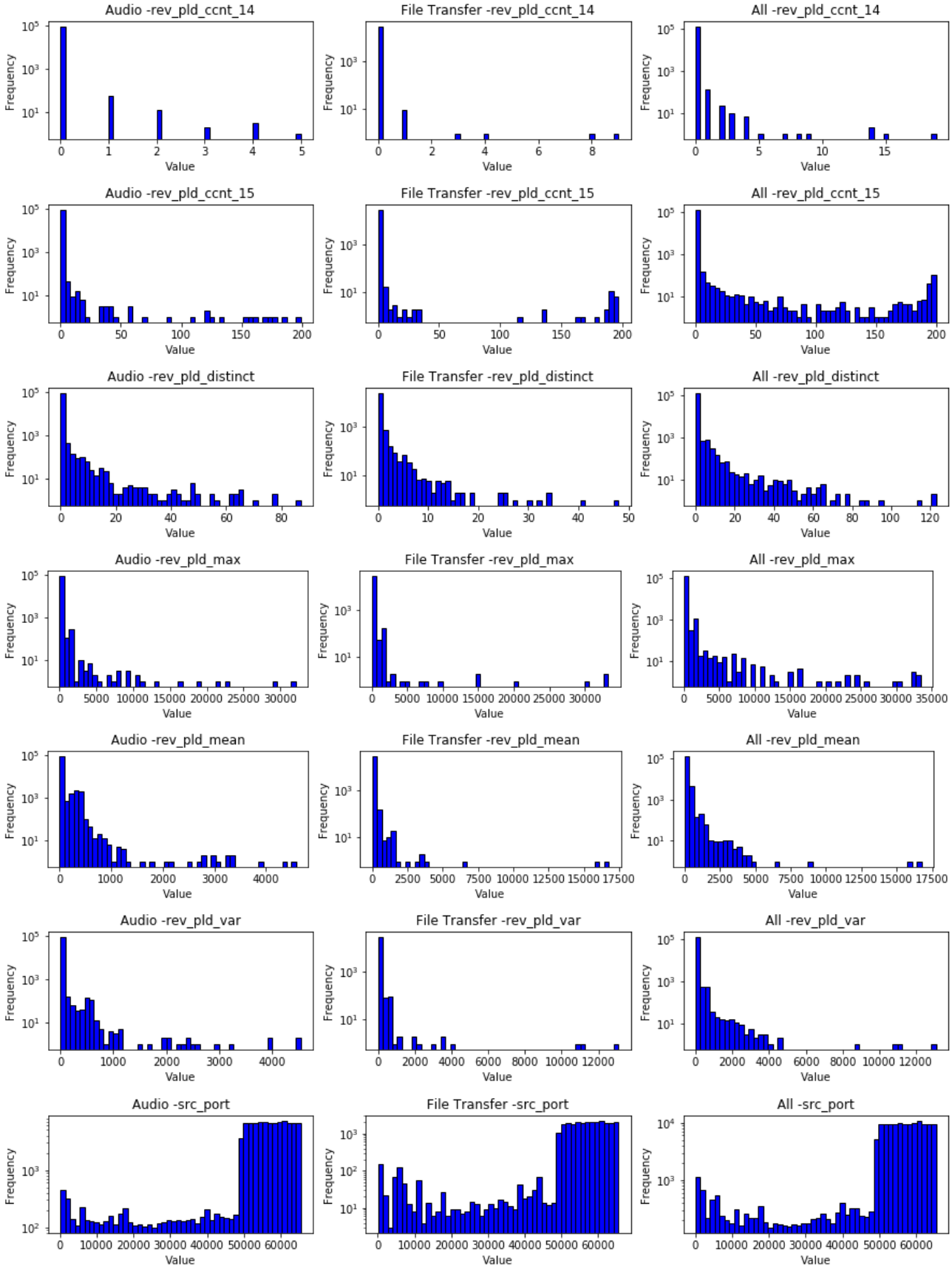

Audio -rev_intervals_ccnt_1
File Transfer -rev_intervals_ccnt_1
All -rev_intervals_ccnt_1
Audio -rev_intervals_ccnt_2
File Transfer -rev_intervals_ccnt_2
All -rev_intervals_ccnt_2
Audio -rev_intervals_ccnt_3
File Transfer -rev_intervals_ccnt_3
All -rev_intervals_ccnt_3
Audio -rev_intervals_ccnt_4
File Transfer -rev_intervals_ccnt_4
All -rev_intervals_ccnt_4
Audio -rev_intervals_ccnt_5
File Transfer -rev_intervals_ccnt_5
All -rev_intervals_ccnt_5
Audio -rev_intervals_ccnt_6
File Transfer -rev_intervals_ccnt_6
All -rev_intervals_ccnt_6
Frequency
Value

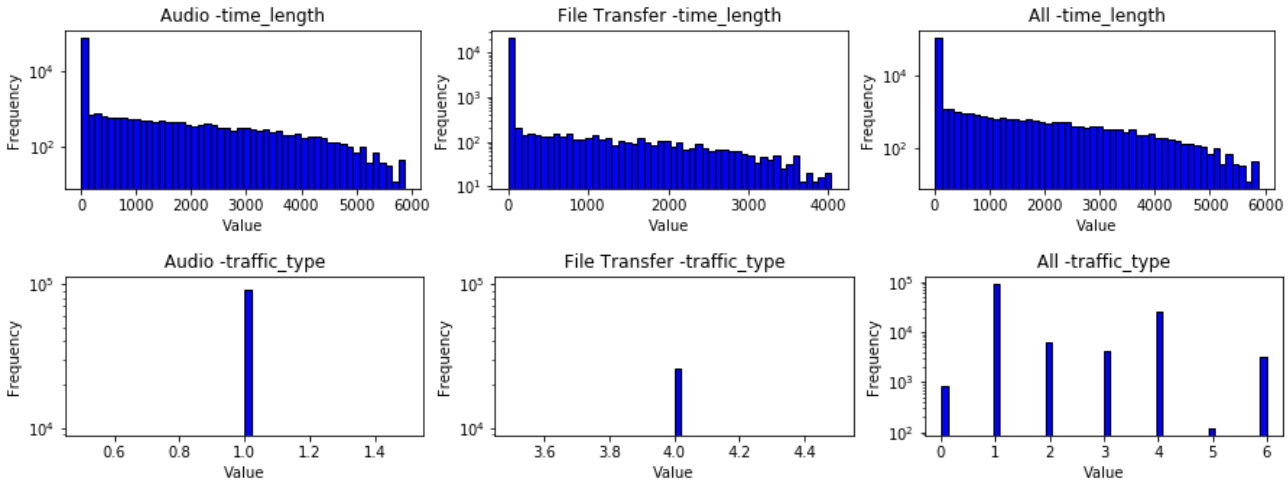

Audio -rev_intervals_ccnt_7
File Transfer -rev_intervals_ccnt_7
All -rev_intervals_ccnt_7
Audio -rev_intervals_ccnt_8
File Transfer -rev_intervals_ccnt_8
All -rev_intervals_ccnt_8
Audio -rev_intervals_ccnt_9
File Transfer -rev_intervals_ccnt_9
All -rev_intervals_ccnt_9
Audio -rev_intervals_ccnt_10
File Transfer -rev_intervals_ccnt_10
All -rev_intervals_ccnt_10
Audio -rev_intervals_ccnt_11
File Transfer -rev_intervals_ccnt_11
All -rev_intervals_ccnt_11
Audio -rev_intervals_ccnt_12
File Transfer -rev_intervals_ccnt_12
All -rev_intervals_ccnt_12
Frequency
Value


Audio -rev_intervals_ccnt_13
File Transfer -rev_intervals_ccnt_13
All -rev_intervals_ccnt_13
Audio -rev_intervals_ccnt_14
File Transfer -rev_intervals_ccnt_14
All -rev_intervals_ccnt_14
Audio -rev_intervals_ccnt_15
File Transfer -rev_intervals_ccnt_15
All -rev_intervals_ccnt_15
Audio -rev_pld_bin_128
File Transfer -rev_pld_bin_128
All -rev_pld_bin_128
Audio -rev_pld_ccnt_0
File Transfer -rev_pld_ccnt_0
All -rev_pld_ccnt_0
Audio -rev_pld_ccnt_1
File Transfer -rev_pld_ccnt_1
All -rev_pld_ccnt_1
Frequency
Value

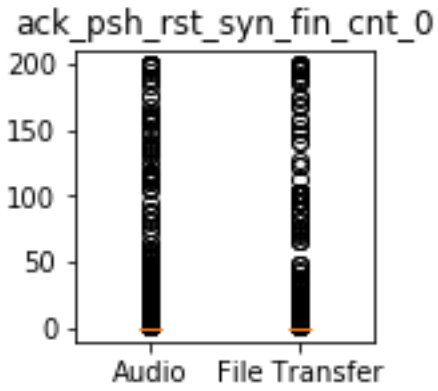

Audio -rev_pld_ccnt_2
File Transfer -rev_pld_ccnt_2
All -rev_pld_ccnt_2
Audio -rev_pld_ccnt_3
File Transfer -rev_pld_ccnt_3
All -rev_pld_ccnt_3
Audio -rev_pld_ccnt_4
File Transfer -rev_pld_ccnt_4
All -rev_pld_ccnt_4
Audio -rev_pld_ccnt_5
File Transfer -rev_pld_ccnt_5
All -rev_pld_ccnt_5
Audio -rev_pld_ccnt_6
File Transfer -rev_pld_ccnt_6
All -rev_pld_ccnt_6
Audio -rev_pld_ccnt_7
File Transfer -rev_pld_ccnt_7
All -rev_pld_ccnt_7
Frequency
Value

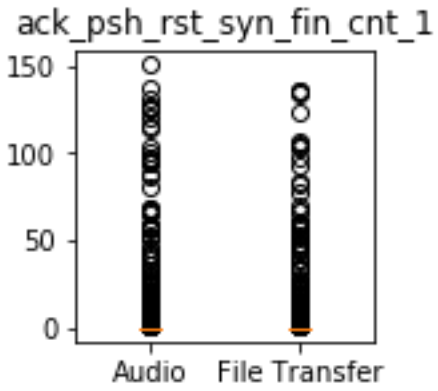

Audio -rev_pld_ccnt_8
File Transfer -rev_pld_ccnt_8
All -rev_pld_ccnt_8
Audio -rev_pld_ccnt_9
File Transfer -rev_pld_ccnt_9
All -rev_pld_ccnt_9
Audio -rev_pld_ccnt_10
File Transfer -rev_pld_ccnt_10
All -rev_pld_ccnt_10
Audio -rev_pld_ccnt_11
File Transfer -rev_pld_ccnt_11
All -rev_pld_ccnt_11
Audio -rev_pld_ccnt_12
File Transfer -rev_pld_ccnt_12
All -rev_pld_ccnt_12
Audio -rev_pld_ccnt_13
File Transfer -rev_pld_ccnt_13
All -rev_pld_ccnt_13
Frequency
Value

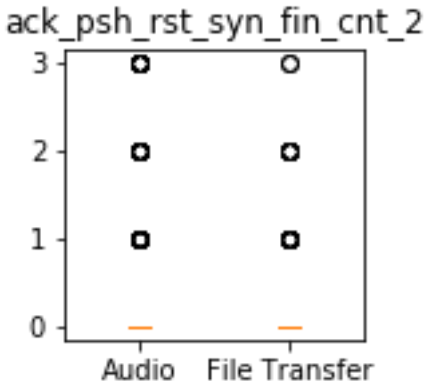

Audio -rev_pld_ccnt_14
File Transfer -rev_pld_ccnt_14
All -rev_pld_ccnt_14
Audio -rev_pld_ccnt_15
File Transfer -rev_pld_ccnt_15
All -rev_pld_ccnt_15
Audio -rev_pld_distinct
File Transfer -rev_pld_distinct
All -rev_pld_distinct
Audio -rev_pld_max
File Transfer -rev_pld_max
All -rev_pld_max
Audio -rev_pld_mean
File Transfer -rev_pld_mean
All -rev_pld_mean
Audio -rev_pld_var
File Transfer -rev_pld_var
All -rev_pld_var
Audio -src_port
File Transfer -src_port
All -src_port
Frequency
Value

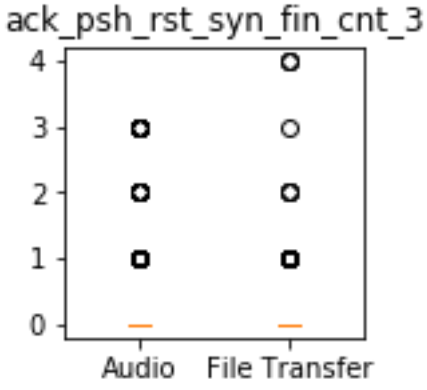


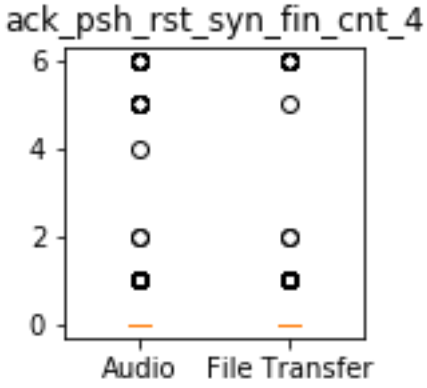
Univariate Analysis - Box Plots

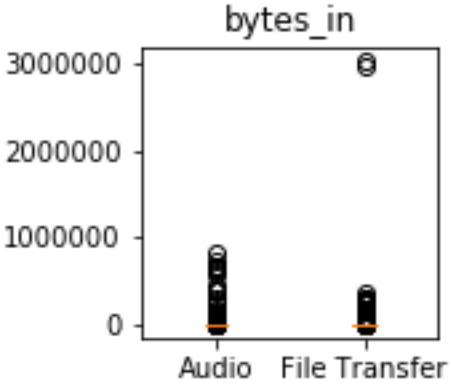


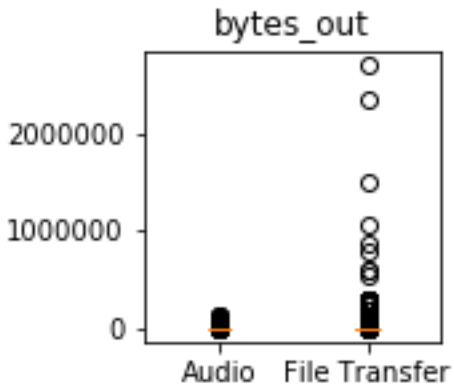


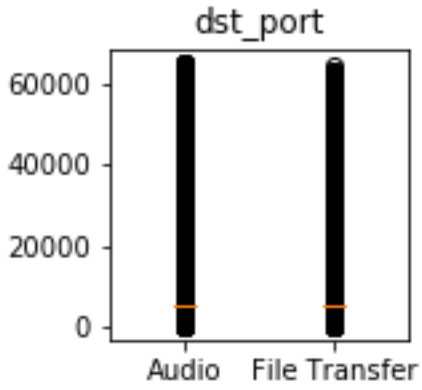

ack_psh_rst_syn_fin_cnt_2
3
2
1
0
Audio
File Transfer

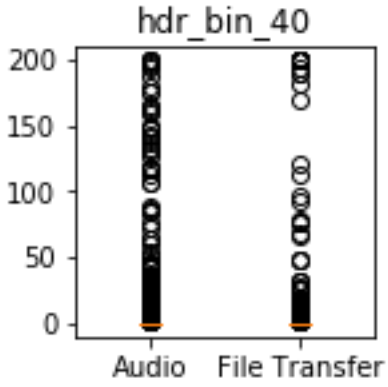

ack_psh_rst_syn_fin_cnt_3
4
3
2
1
0
Audio
File Transfer

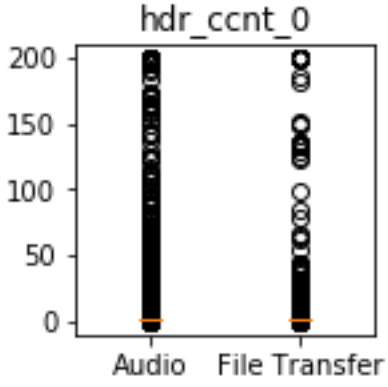

ack_psh_rst_syn_fin_cnt_4
6
4
2
0
Audio
File Transfer

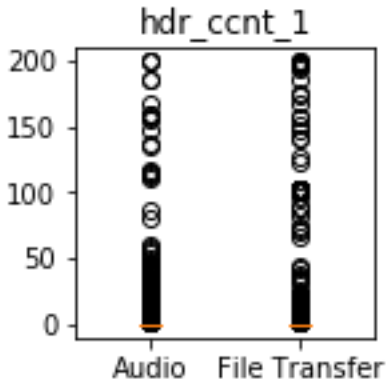

bytes_in
3000000
2000000
1000000
0
Audio
File Transfer

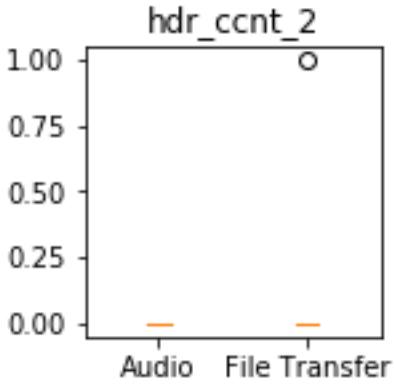

bytes_out
2000000
1000000
0
Audio
File Transfer

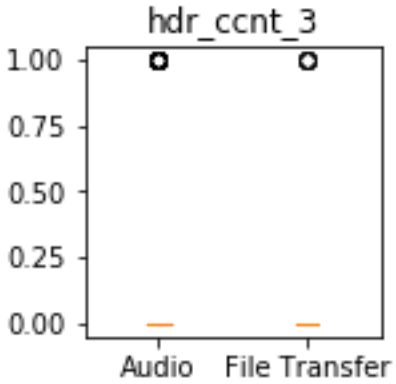

dst_port
60000
40000
20000
0
Audio
File Transfer

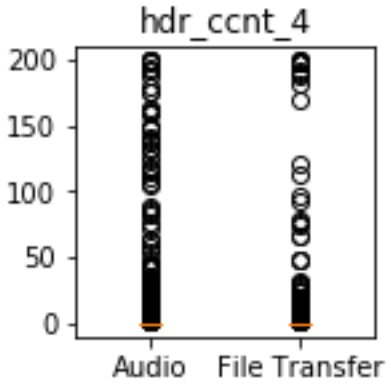

hdr_bin_40
200
150
100
50
0
Audio
File Transfer

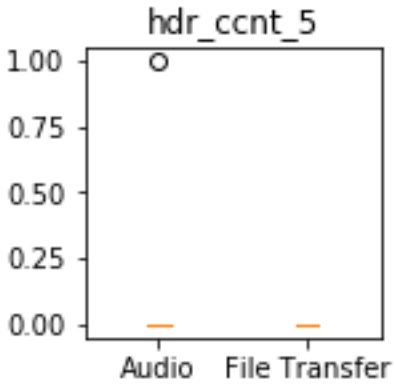

hdr_ccnt_0
200
150
100
50
0
Audio
File Transfer

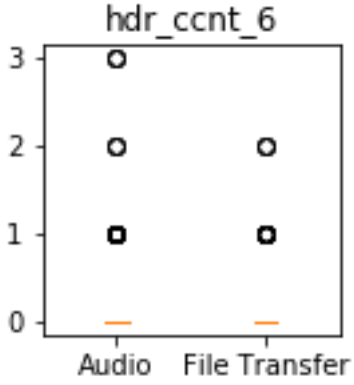

hdr_ccnt_1
200
150
100
50
0
Audio
File Transfer

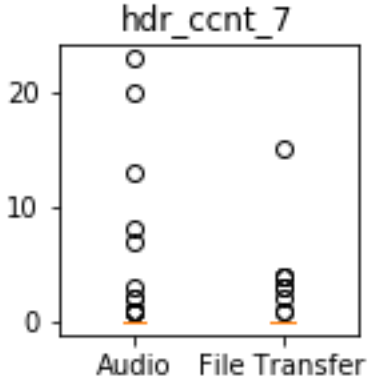

hdr_ccnt_2
1.00
0.75
0.50
0.25
0.00
Audio
File Transfer

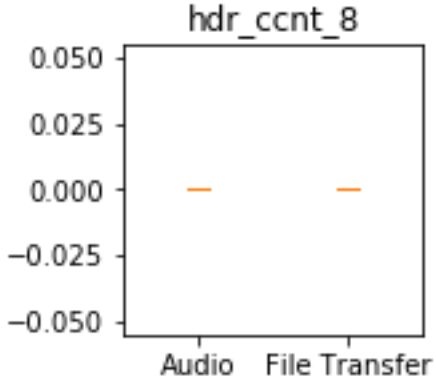

hdr_ccnt_3
1.00
0.75
0.50
0.25
0.00
Audio
File Transfer

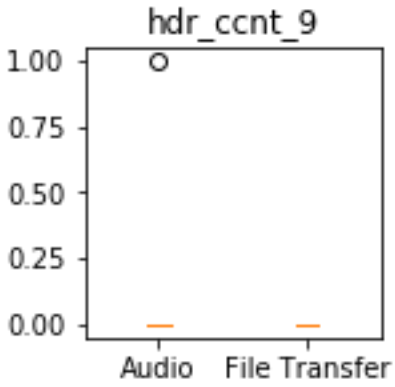

hdr_ccnt_4
200
150
100
50
0
Audio
File Transfer

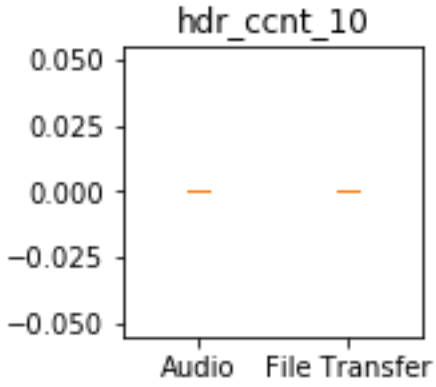

hdr_ccnt_5
1.00
0.75
0.50
0.25
0.00
Audio
File Transfer

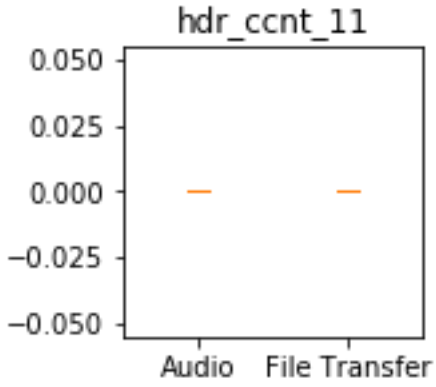

hdr_ccnt_6
3
2
1
0
Audio
File Transfer


hdr_ccnt_7
20
10
0
Audio
File Transfer

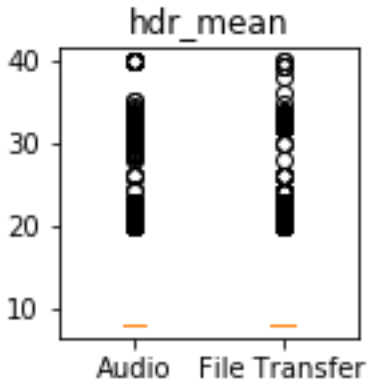

hdr_ccnt_8
0.050
0.025
0.000
−0.025
−0.050
Audio
File Transfer

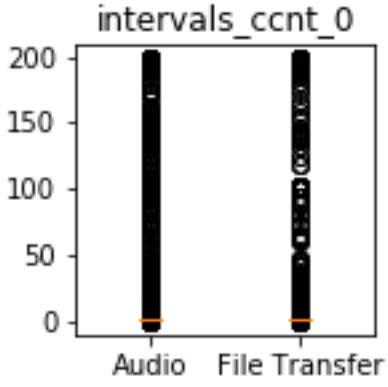

hdr_ccnt_9
1.00
0.75
0.50
0.25
0.00
Audio
File Transfer

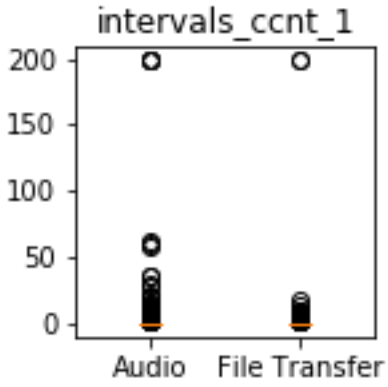

hdr_ccnt_10
0.050
0.025
0.000
−0.025
−0.050
Audio
File Transfer

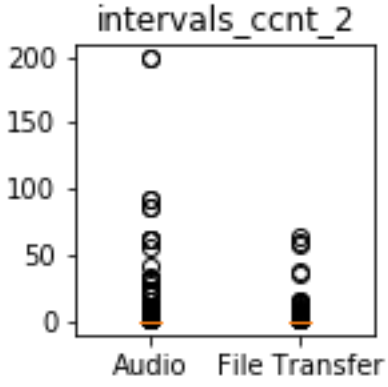

hdr_ccnt_11
0.050
0.025
0.000
−0.025
−0.050
Audio
File Transfer

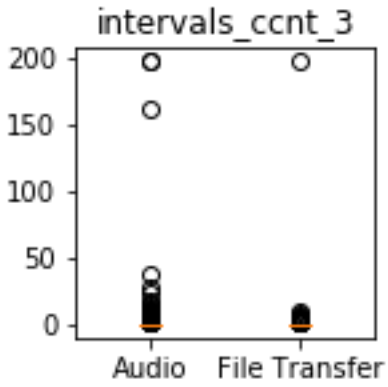

hdr_distinct
4
3
2
1
Audio
File Transfer

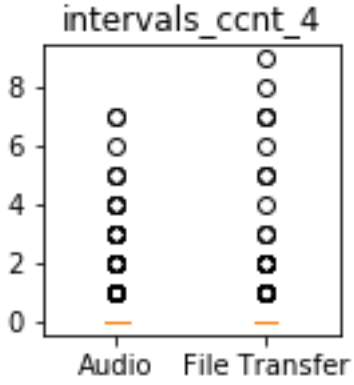

hdr_mean
40
30
20
10
Audio
File Transfer

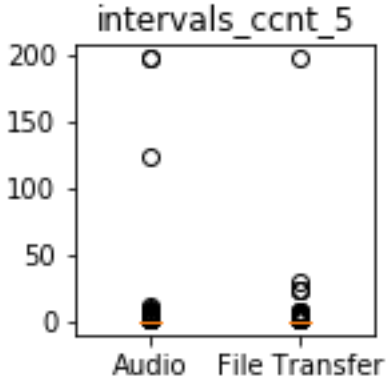

intervals_ccnt_0
200
150
100
50
0
Audio
File Transfer

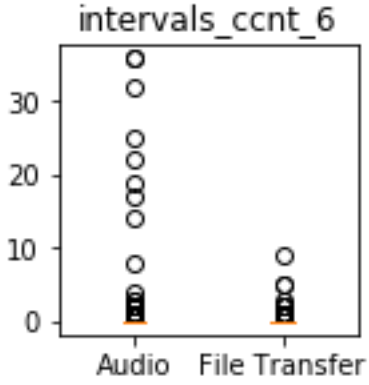

intervals_ccnt_1
200
150
100
50
0
Audio
File Transfer

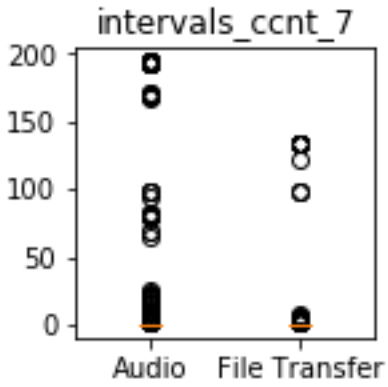

intervals_ccnt_2
200
150
100
50
0
Audio
File Transfer

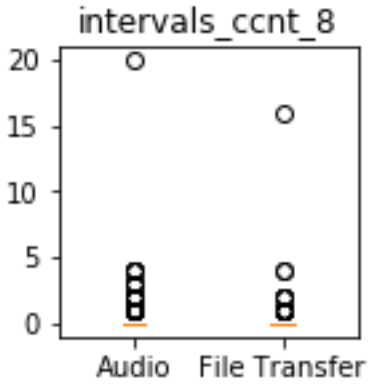

intervals_ccnt_3
200
150
100
50
0
Audio
File Transfer

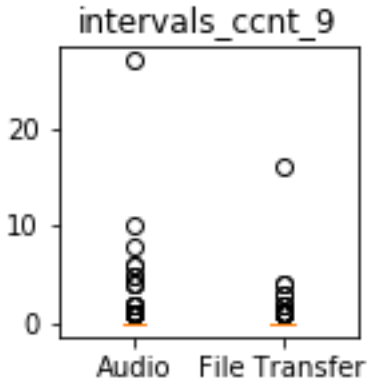

intervals_ccnt_4
8
6
4
2
0
Audio
File Transfer

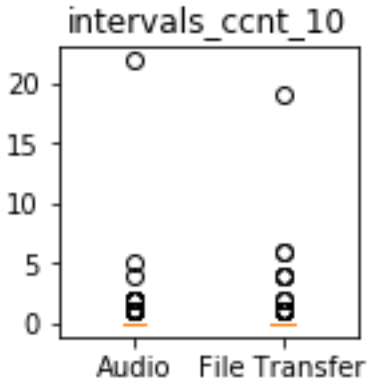

intervals_ccnt_5
200
150
100
50
0
Audio
File Transfer

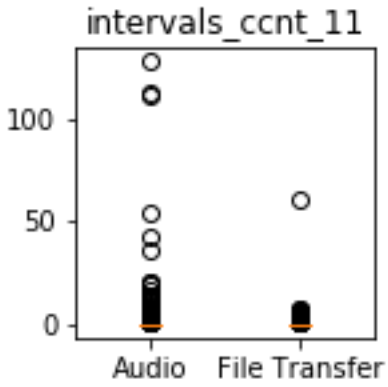

intervals_ccnt_6
30
20
10
0
Audio
File Transfer

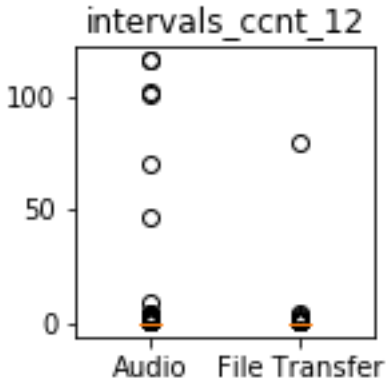

intervals_ccnt_7
200
150
100
50
0
Audio
File Transfer

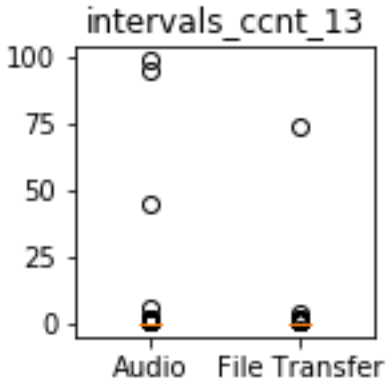

intervals_ccnt_8
20
15
10
5
0
Audio
File Transfer

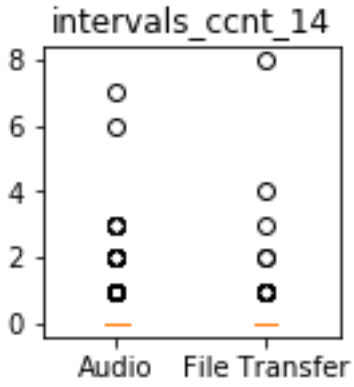

intervals_ccnt_9
20
10
0
Audio
File Transfer

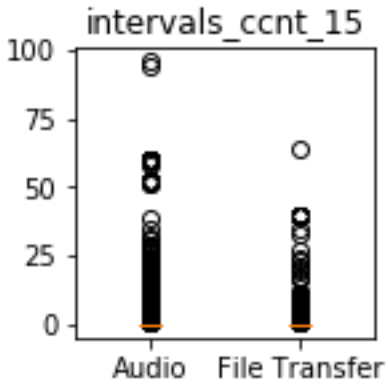

intervals_ccnt_10
20
15
10
5
0
Audio
File Transfer

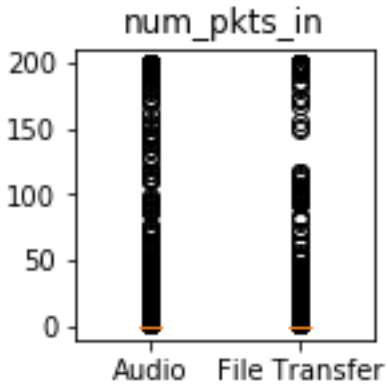

intervals_ccnt_11
100
50
0
Audio
File Transfer

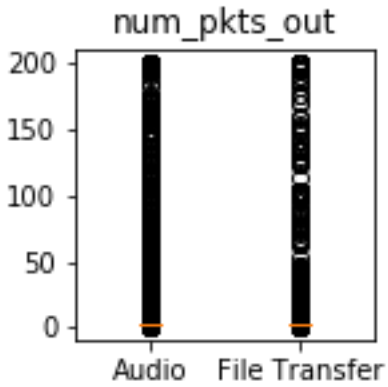

intervals_ccnt_12
100
50
0
Audio
File Transfer

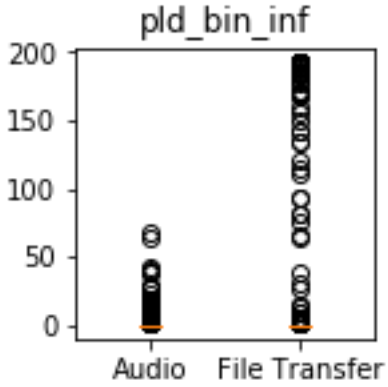

intervals_ccnt_13
100
75
50
25
0
Audio
File Transfer

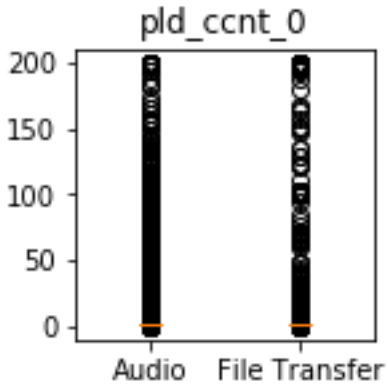

intervals_ccnt_14
8
6
4
2
0
Audio
File Transfer

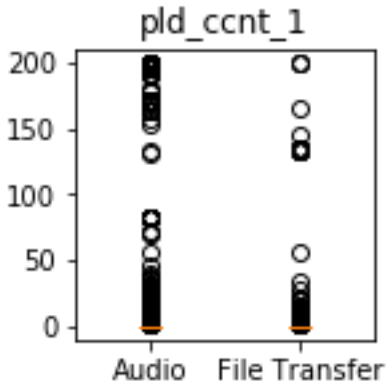

intervals_ccnt_15
100
75
50
25
0
Audio
File Transfer

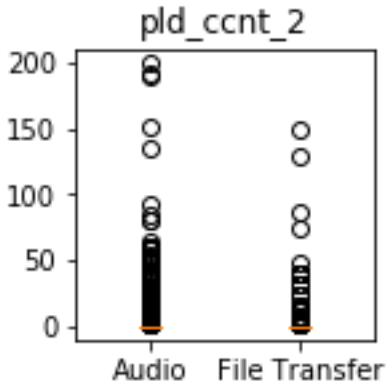

num_pkts_in
200
150
100
50
0
Audio
File Transfer

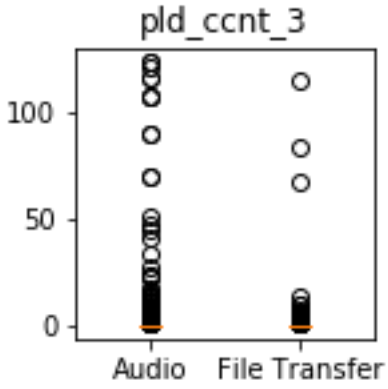

num_pkts_out
200
150
100
50
0
Audio
File Transfer

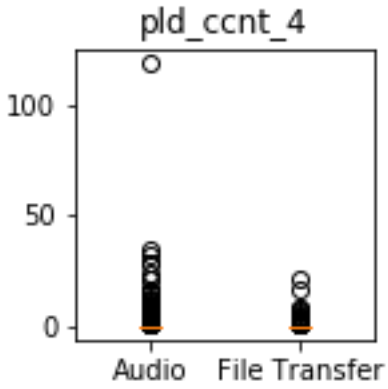

pld_bin_inf
200
150
100
50
0
Audio
File Transfer

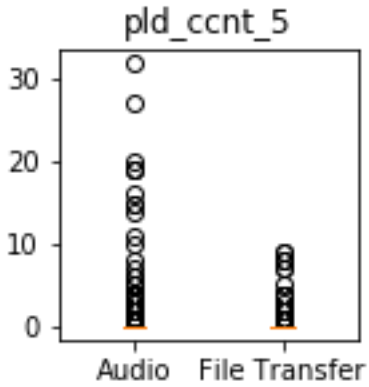

pld_ccnt_0
200
150
100
50
0
Audio
File Transfer

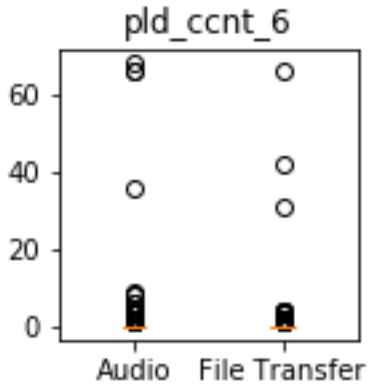

pld_ccnt_1
200
150
100
50
0
Audio
File Transfer

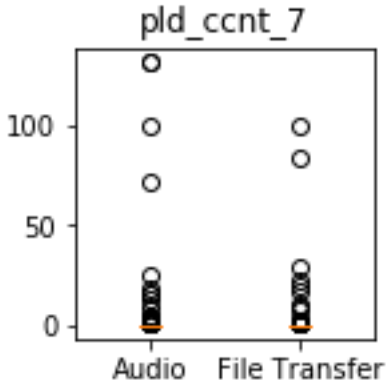

pld_ccnt_2
200
150
100
50
0
Audio
File Transfer

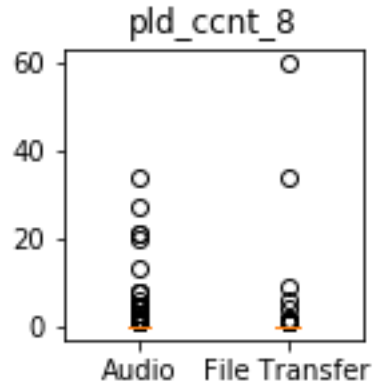

pld_ccnt_3
100
50
0
Audio
File Transfer

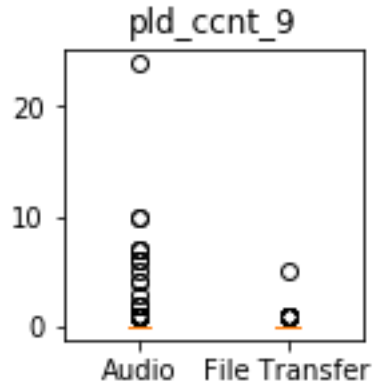

pld_ccnt_4
100
50
0
Audio
File Transfer

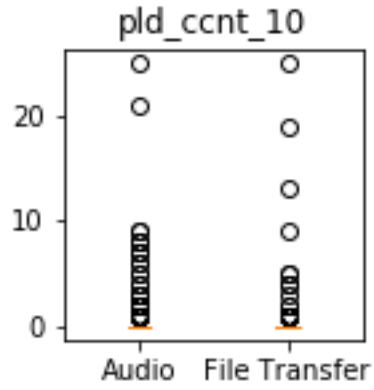

pld_ccnt_5
30
20
10
0
Audio
File Transfer

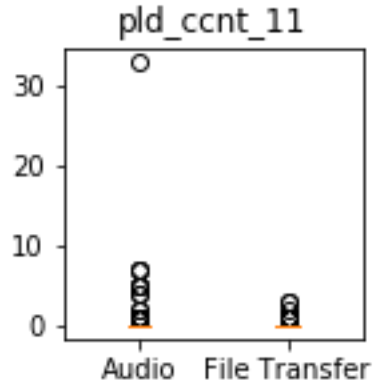

pld_ccnt_6
60
40
20
0
Audio
File Transfer

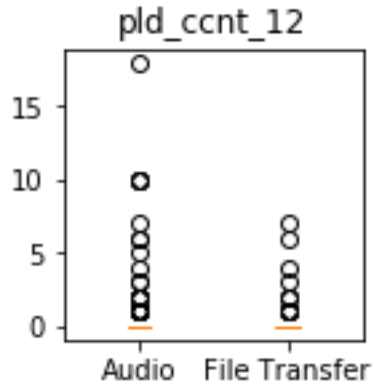

pld_ccnt_7
100
50
0
Audio
File Transfer

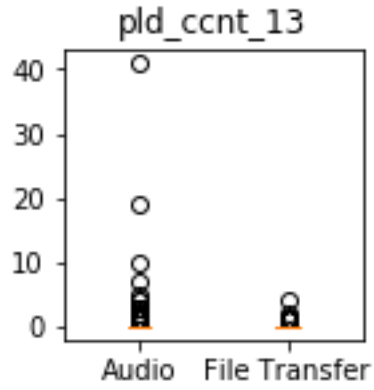

pld_ccnt_8
60
40
20
0
Audio
File Transfer

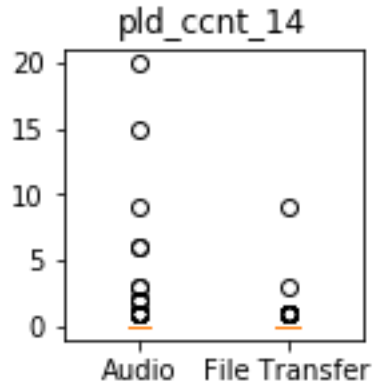

pld_ccnt_9
20
10
0
Audio
File Transfer

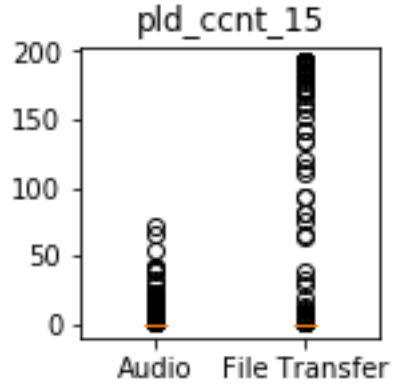

pld_ccnt_10
20
10
0
Audio
File Transfer

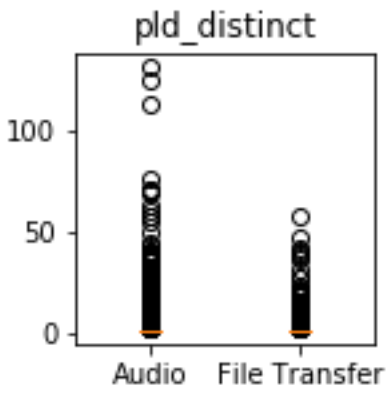

pld_ccnt_11
30
20
10
0
Audio
File Transfer

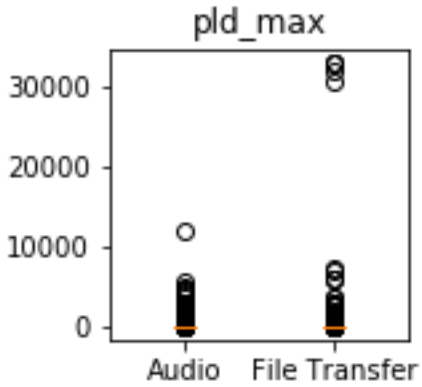

pld_ccnt_12
15
10
5
0
Audio
File Transfer

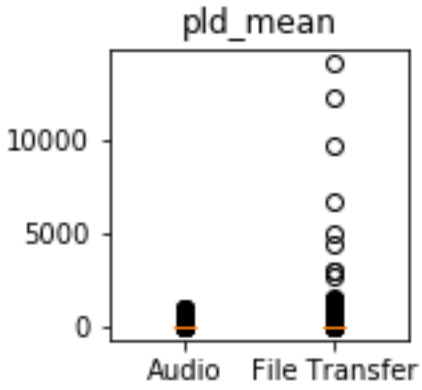

pld_ccnt_13
40
30
20
10
0
Audio
File Transfer

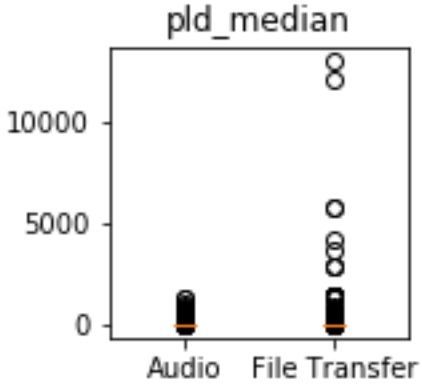

pld_ccnt_14
20
15
10
5
0
Audio
File Transfer

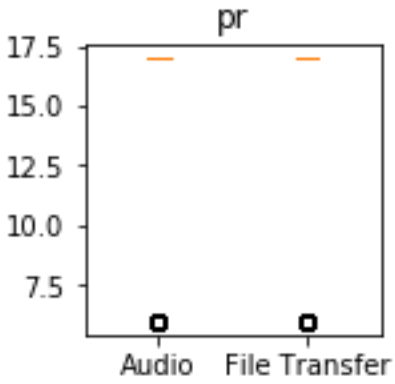

pld_ccnt_15
200
150
100
50
0
Audio
File Transfer

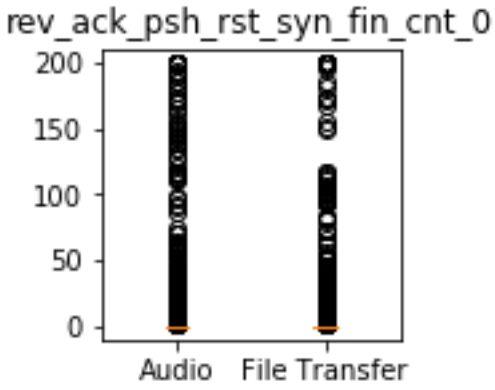

pld_distinct
100
50
0
Audio
File Transfer

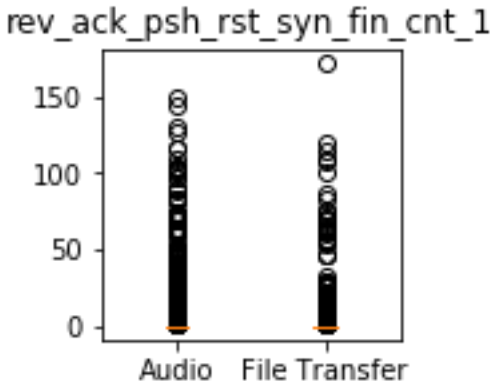

pld_max
30000
20000
10000
0
Audio
File Transfer

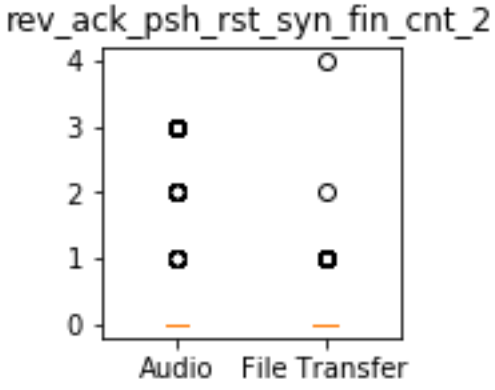

pld_mean
10000
5000
0
Audio
File Transfer

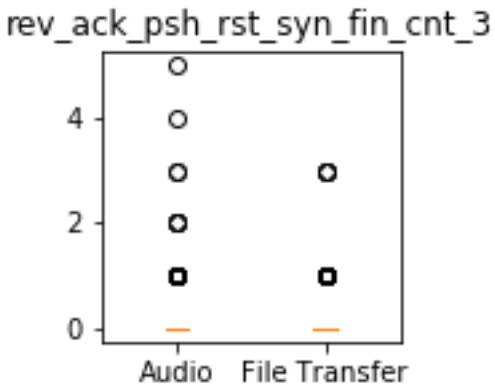

pld_median
10000
5000
0
Audio
File Transfer

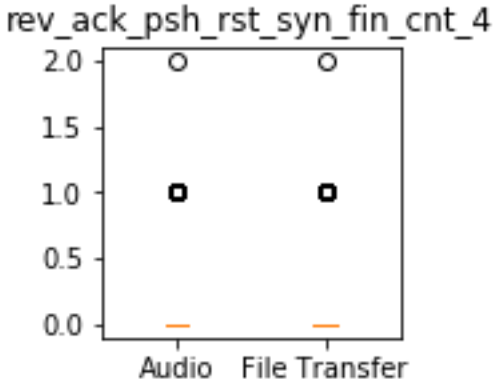

pr
17.5
15.0
12.5
10.0
7.5
Audio
File Transfer

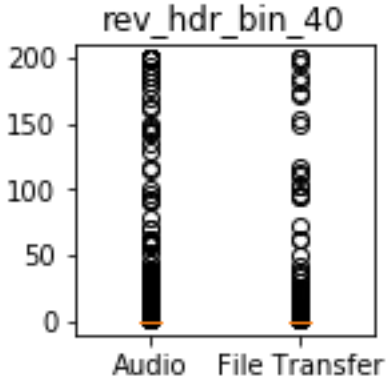

rev_ack_psh_rst_syn_fin_cnt_0
200
150
100
50
0
Audio
File Transfer

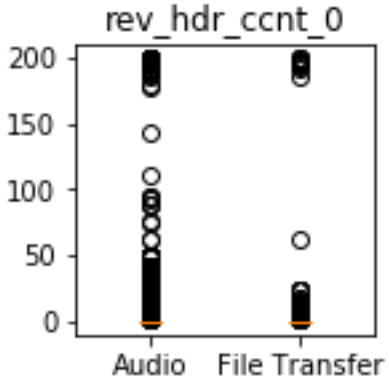

rev_ack_psh_rst_syn_fin_cnt_1
150
100
50
0
Audio
File Transfer

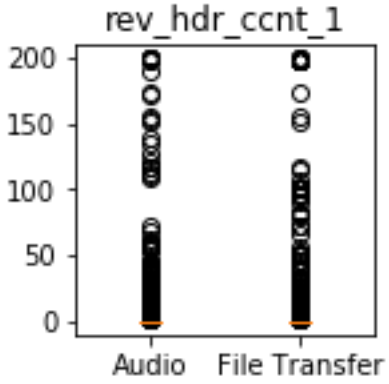

rev_ack_psh_rst_syn_fin_cnt_2
4
3
2
1
0
Audio
File Transfer

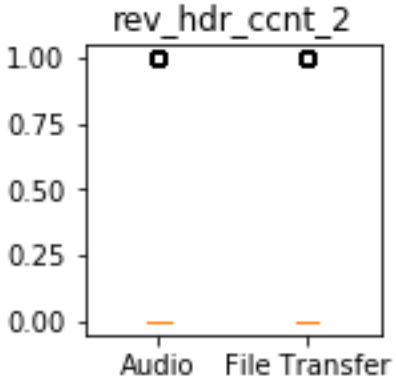

rev_ack_psh_rst_syn_fin_cnt_3
4
2
0
Audio
File Transfer

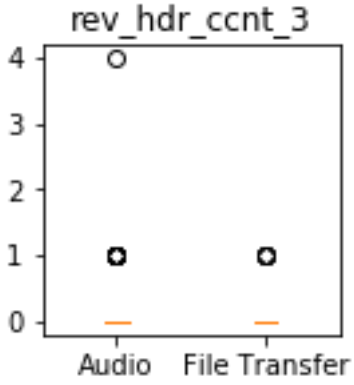

rev_ack_psh_rst_syn_fin_cnt_4
2.0
1.5
1.0
0.5
0.0
Audio
File Transfer

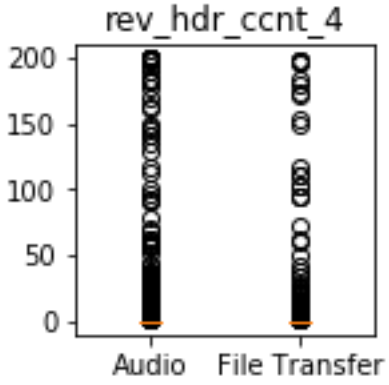

rev_hdr_bin_40
200
150
100
50
0
Audio
File Transfer

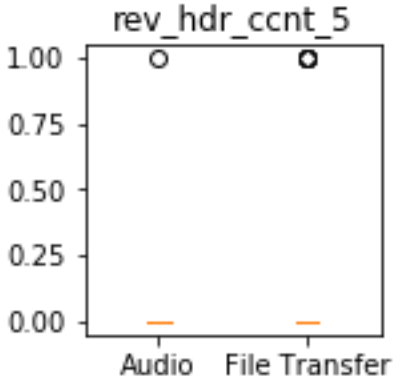

rev_hdr_ccnt_0
200
150
100
50
0
Audio
File Transfer

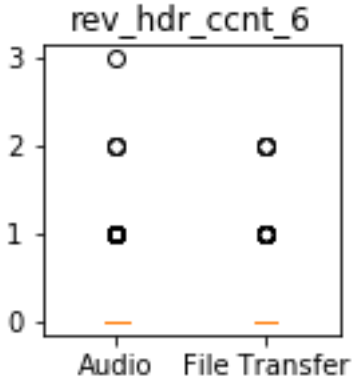

rev_hdr_ccnt_1
200
150
100
50
0
Audio
File Transfer

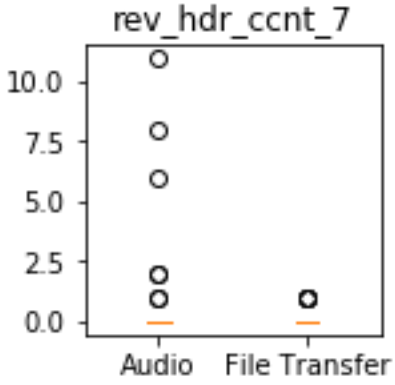

rev_hdr_ccnt_2
1.00
0.75
0.50
0.25
0.00
Audio
File Transfer

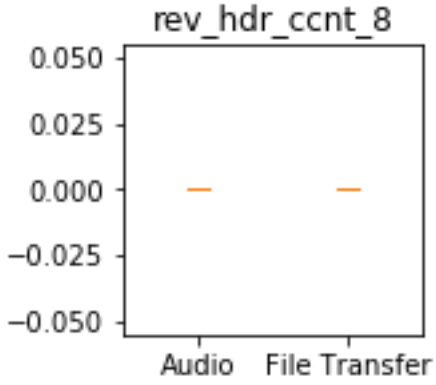

rev_hdr_ccnt_3
4
3
2
1
0
Audio
File Transfer

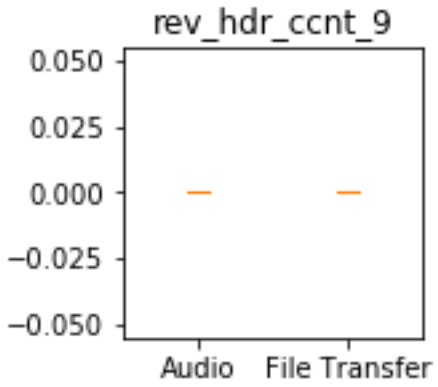

rev_hdr_ccnt_4
200
150
100
50
0
Audio
File Transfer

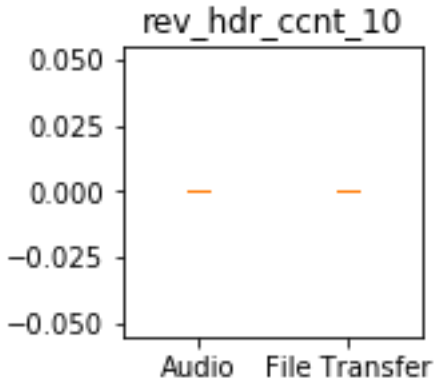

rev_hdr_ccnt_5
1.00
0.75
0.50
0.25
0.00
Audio
File Transfer

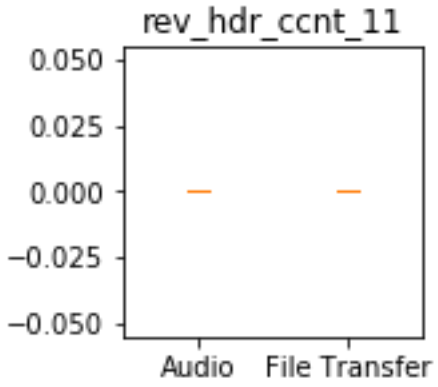

rev_hdr_ccnt_6
3
2
1
0
Audio
File Transfer

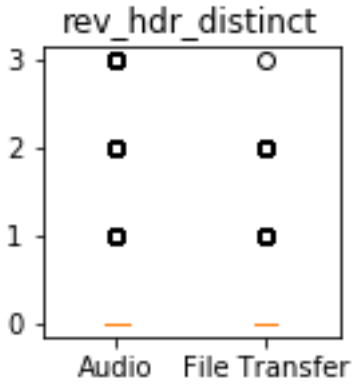

rev_hdr_ccnt_7
10.0
7.5
5.0
2.5
0.0
Audio
File Transfer

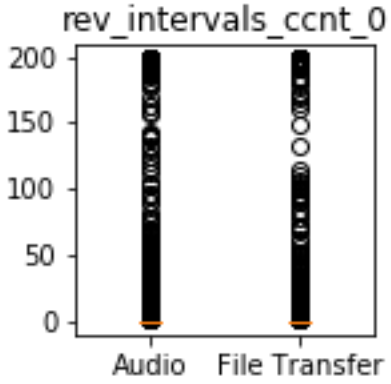

rev_hdr_ccnt_8
0.050
0.025
0.000
−0.025
−0.050
Audio
File Transfer

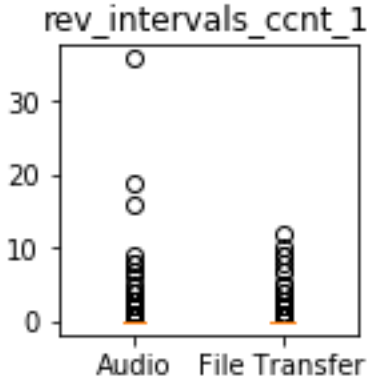

rev_hdr_ccnt_9
0.050
0.025
0.000
−0.025
−0.050
Audio
File Transfer

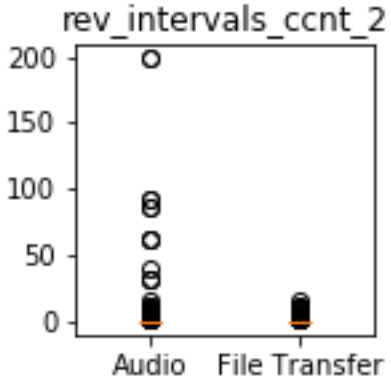

rev_hdr_ccnt_10
0.050
0.025
0.000
−0.025
−0.050
Audio
File Transfer

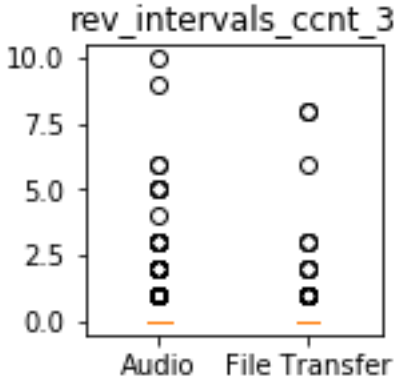

rev_hdr_ccnt_11
0.050
0.025
0.000
−0.025
−0.050
Audio
File Transfer

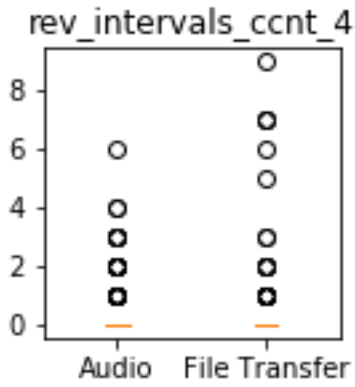

rev_hdr_distinct
3
2
1
0
Audio
File Transfer

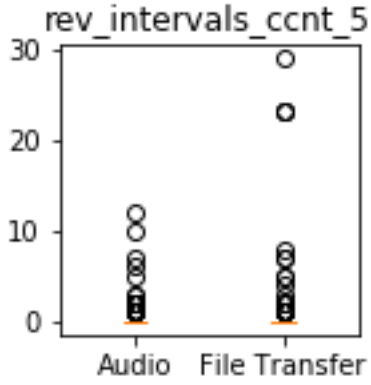

rev_intervals_ccnt_0
200
150
100
50
0
Audio
File Transfer

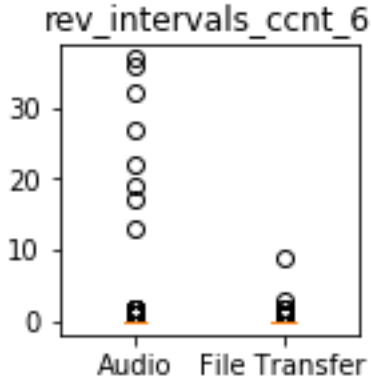

rev_intervals_ccnt_1
30
20
10
0
Audio
File Transfer

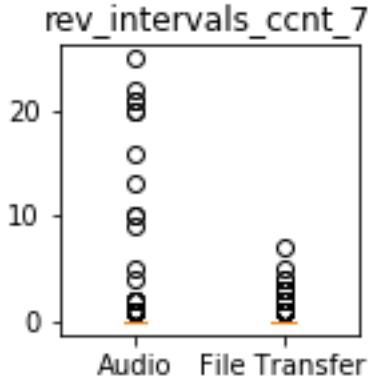

rev_intervals_ccnt_2
200
150
100
50
0
Audio
File Transfer

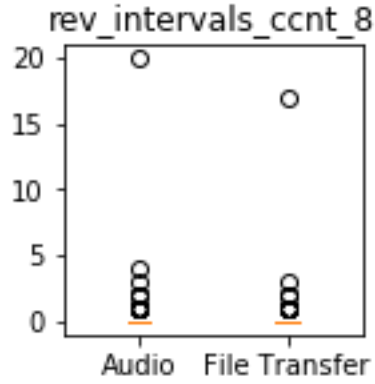

rev_intervals_ccnt_3
10.0
7.5
5.0
2.5
0.0
Audio
File Transfer

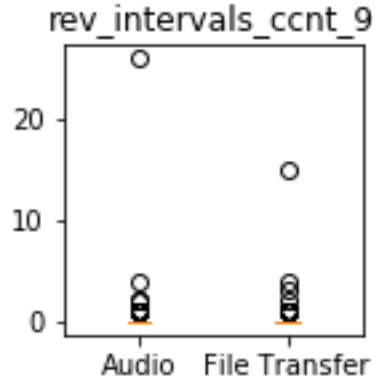

rev_intervals_ccnt_4
8
6
4
2
0
Audio
File Transfer

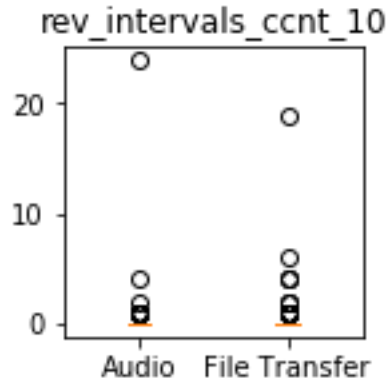

rev_intervals_ccnt_5
30
20
10
0
Audio
File Transfer

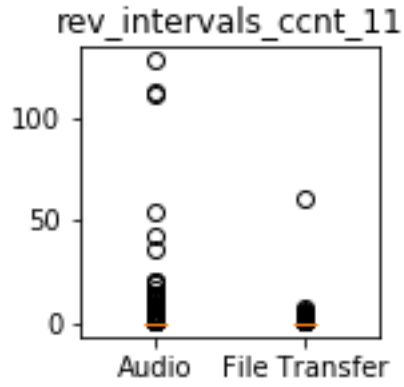

rev_intervals_ccnt_6
30
20
10
0
Audio
File Transfer

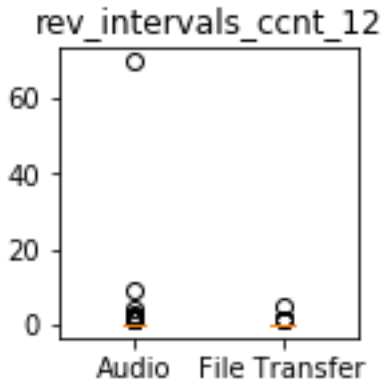

rev_intervals_ccnt_7
20
10
0
Audio
File Transfer

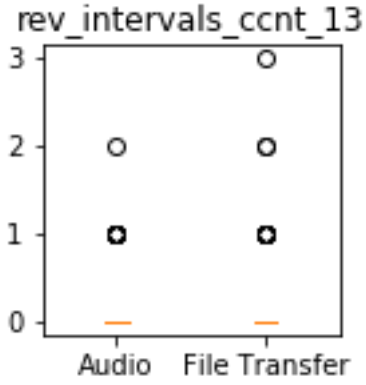

rev_intervals_ccnt_8
20
15
10
5
0
Audio
File Transfer

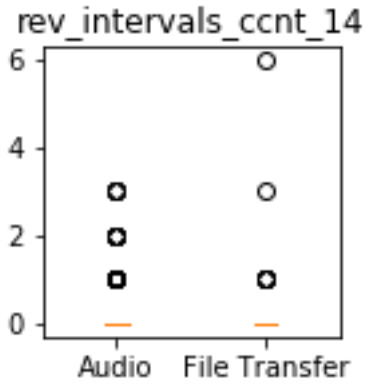

rev_intervals_ccnt_9
20
10
0
Audio
File Transfer

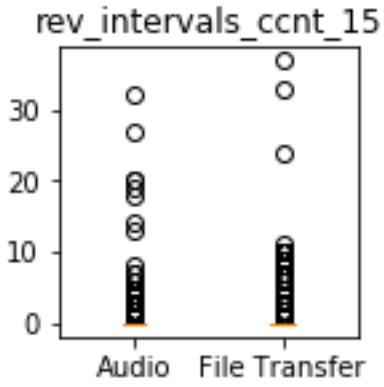

rev_intervals_ccnt_10
20
10
0
Audio
File Transfer

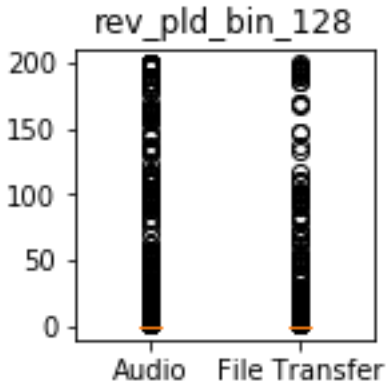

rev_intervals_ccnt_11
100
50
0
Audio
File Transfer

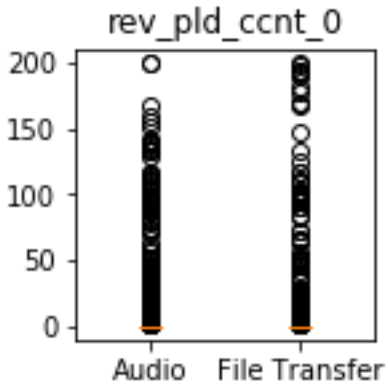

rev_intervals_ccnt_12
60
40
20
0
Audio
File Transfer

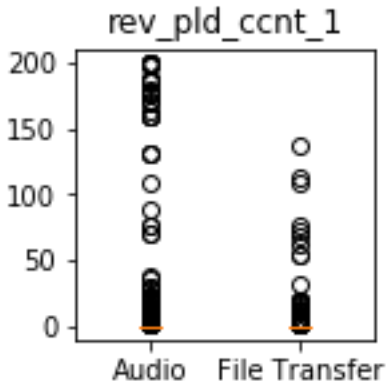

rev_intervals_ccnt_13
3
2
1
0
Audio
File Transfer

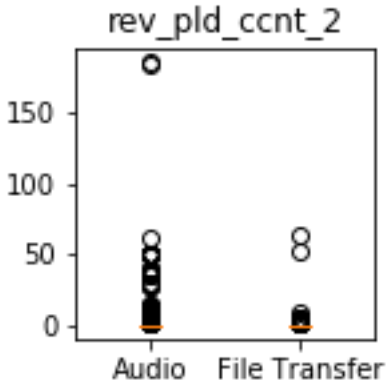

rev_intervals_ccnt_14
6
4
2
0
Audio
File Transfer

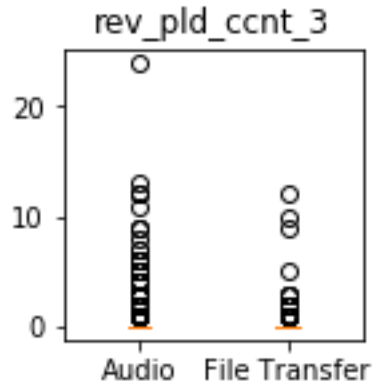

rev_intervals_ccnt_15
30
20
10
0
Audio
File Transfer

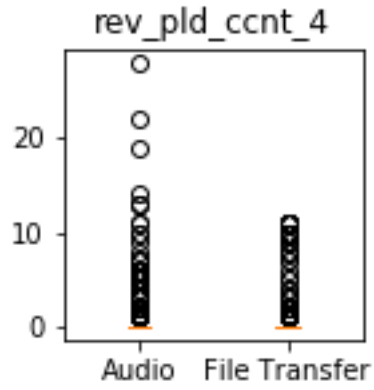

rev_pld_bin_128
200
150
100
50
0
Audio
File Transfer

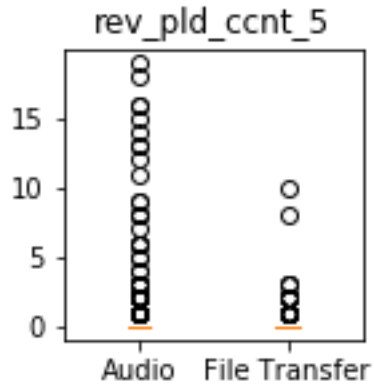

rev_pld_ccnt_0
200
150
100
50
0
Audio
File Transfer

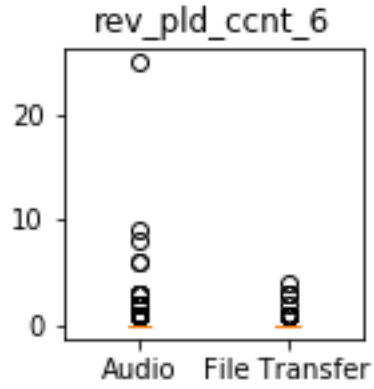

rev_pld_ccnt_1
200
150
100
50
0
Audio
File Transfer

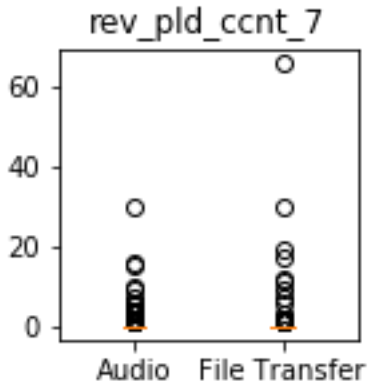

rev_pld_ccnt_2
150
100
50
0
Audio
File Transfer

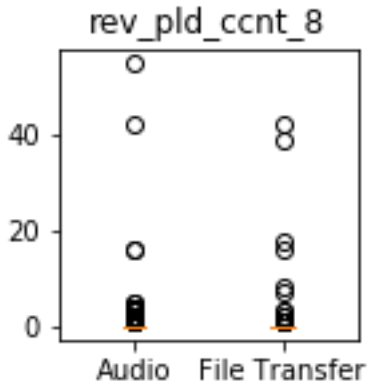

rev_pld_ccnt_3
20
10
0
Audio
File Transfer

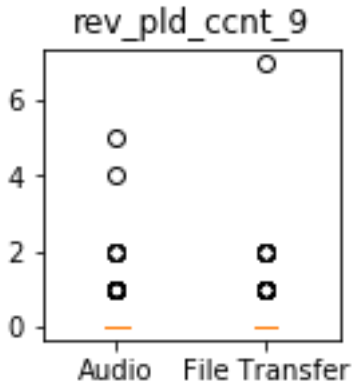

rev_pld_ccnt_4
20
10
0
Audio
File Transfer

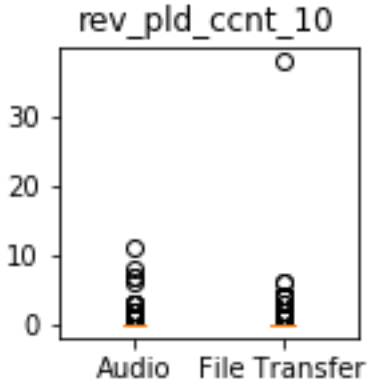

rev_pld_ccnt_5
15
10
5
0
Audio
File Transfer

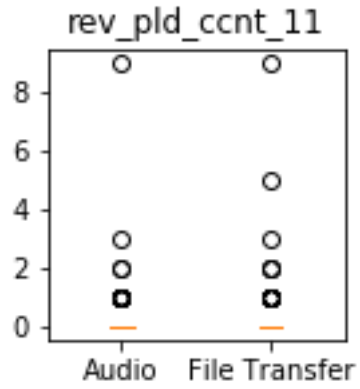

rev_pld_ccnt_6
20
10
0
Audio
File Transfer

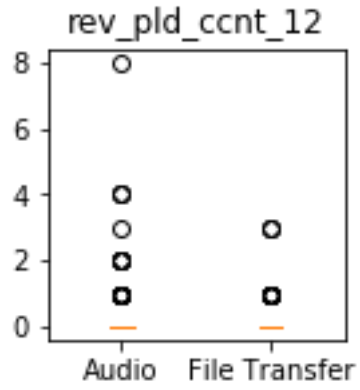

rev_pld_ccnt_7
60
40
20
0
Audio
File Transfer

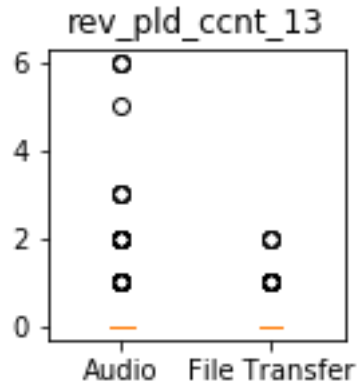

rev_pld_ccnt_8
40
20
0
Audio
File Transfer

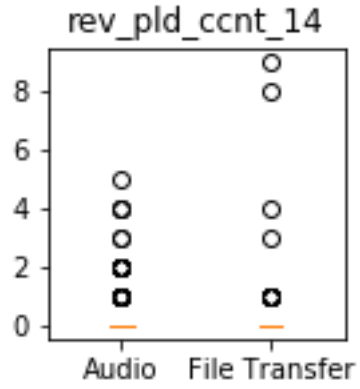

rev_pld_ccnt_9
6
4
2
0
Audio
File Transfer

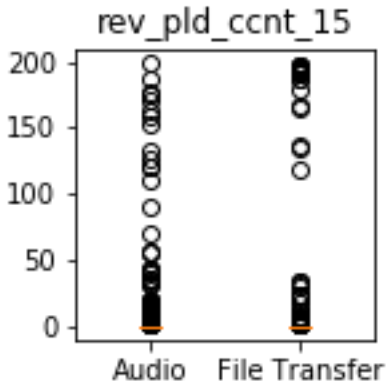

rev_pld_ccnt_10
30
20
10
0
Audio
File Transfer

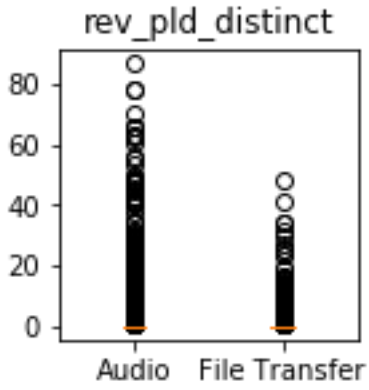

rev_pld_ccnt_11
8
6
4
2
0
Audio
File Transfer

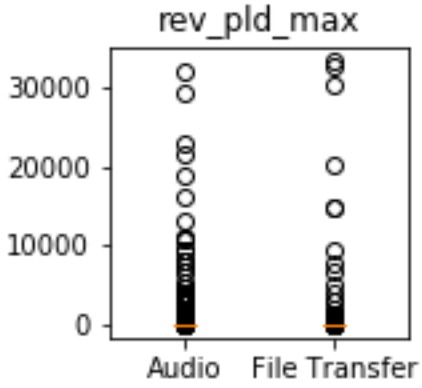

rev_pld_ccnt_12
8
6
4
2
0
Audio
File Transfer

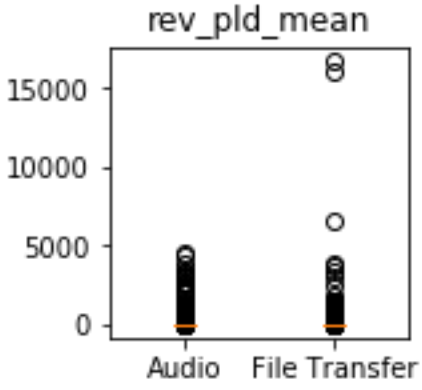

rev_pld_ccnt_13
6
4
2
0
Audio
File Transfer

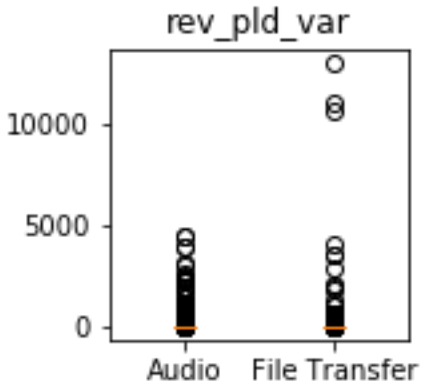

rev_pld_ccnt_14
8
6
4
2
0
Audio
File Transfer

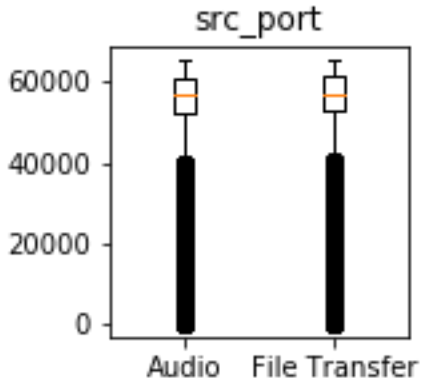

rev_pld_ccnt_15
200
150
100
50
0
Audio
File Transfer

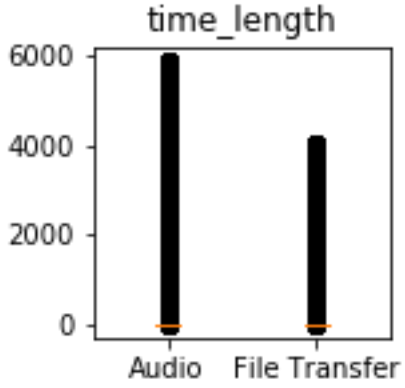

rev_pld_distinct
80
60
40
20
0
Audio
File Transfer

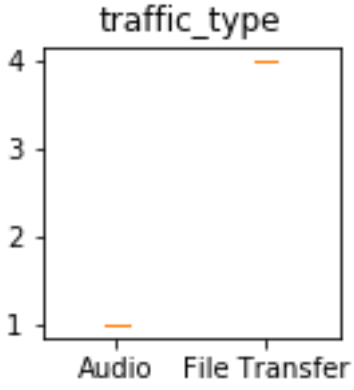

rev_pld_max
30000
20000
10000
0
Audio
File Transfer


rev_pld_mean
15000
10000
5000
0
Audio
File Transfer


rev_pld_var
10000
5000
0
Audio
File Transfer


src_port
60000
40000
20000
0
Audio
File Transfer


time_length
6000
4000
2000
0
Audio
File Transfer


traffic_type
4
3
2
1
Audio
File Transfer

Bivariate Analysis