3-2020

Quantitative Analysis of Evaluation Criteria for Generative Models

Marvin W. Newlin

QUANTITATIVE ANALYSIS OF EVALUATION CRITERIA FOR GENERATIVE MODELS

THESIS

Marvin W Newlin, Second Lieutenant, USAF

AFIT-ENG-MS-20-M-048

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-20-M-048

QUANTITATIVE ANALYSIS OF EVALUATION CRITERIA FOR GENERATIVE MODELS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Marvin W Newlin, B.S.C.S., B.S.M.

Second Lieutenant, USAF

March 2020

AFIT-ENG-MS-20-M-048

QUANTITATIVE ANALYSIS OF EVALUATION CRITERIA FOR GENERATIVE MODELS

THESIS

Marvin W Newlin, B.S.C.S., B.S.M.
Second Lieutenant, USAF

Committee Membership:

Mark E DeYoung, Lt Col, Ph.D
Chair

Laurence D Merkle, Ph.D
Member

Clark N Taylor, Ph.D
Member

AFIT-ENG-MS-20-M-048

Abstract

Machine Learning (ML) is rapidly becoming integrated in critical aspects of cybersecurity today, particularly in the area of network intrusion/anomaly detection. However, ML techniques require large volumes of data to be effective. The available data is a critical aspect of the ML process for training, classification, and testing purposes. One solution to the problem is to generate synthetic data that is realistic. With the application of ML to this area, one promising application is the use of ML to perform the data generation. With the ability to generate synthetic data comes the need to evaluate the "realness" of the generated data. This research focuses specifically on the problem of evaluating the evaluation criteria. Quantitative analysis of evaluation criteria is important so that future research can have quantitative evidence for the evaluation criteria they utilize. The goal of this research is to provide a framework that can be used to inform and improve the process of generating synthetic semi-structured sequential data. A series of experiments evaluating a chosen set of metrics on discriminative ability and efficiency is performed. This research shows that the choice of feature space in which distances are calculated in is critical. The ability to discriminate between real and generated data hinges on the space that the distances are calculated in. Additionally, the choice of metric significantly affects the sample distance distributions in a suitable feature space. There are three main contributions from this work. First, this work provides the first known framework for evaluating metrics for semi-structured sequential synthetic data generation. Second, this work provides a "black box" evaluation framework which is generator agnostic. Third, this research provides the first known evaluation of metrics for semi-structured sequential data.

AFIT-ENG-MS-20-M-048

This work is dedicated to my wife and our sons

Acknowledgements

I would like to thank my advisor for guiding me through the research process and continuing to provide input after his deployment. I would also like to thank the members of my committee for their valuable input and guidance throughout my research. I would like to acknowledge my lovely wife for taking care of our children and managing our house through the long nights and weekends of homework and research. Lastly, I would like to thank the members of Dinner Squad for the camaraderie, input, and thought-provoking discussions every Taco Tuesday.

Marvin W Newlin

Table of Contents

List of Figures

Figure Page

Figure Page

List of Tables

QUANTITATIVE ANALYSIS OF EVALUATION CRITERIA FOR GENERATIVE MODELS

I. Introduction

1.1 Problem Background

Machine Learning (ML) is rapidly becoming integrated in critical aspects of cybersecurity today, particularly in the area of network intrusion/anomaly detection. However, ML techniques require large volumes of data to be effective. The availability of data is a critical aspect of the ML process for training, classification, and testing purposes [1]. Network Intrusion Detection System (IDS) are an area where ML and Deep Learning (DL) are being heavily utilized. Network IDS are a critical component of network security as they form the backbone of a network's defense strategy for preventing cyber attacks. The ability of IDS to effectively leverage the capabilities of ML and DL relies heavily upon the data available for training and testing purposes. This reliance on data is negatively impacted by a lack of realistic datasets with which IDS can be trained [2, 3].

One impediment to the availability of datasets is the privacy issues that arise with utilizing real data. Anonymizing real data so that the perpetrators of certain attacks are not revealed or private data is not dispersed is difficult and often leads to only analysing parts of the network data, or removing the actual payload data from the traffic [4]. Another impediment to dataset availability is the general difficulty in obtaining network data. This can be due to agreements set in place to prevent use of real data, which relates back to the privacy issues with real data utilization.

Additionally, some types of network anomalies like certain types of malware do not exist or are very hard to find in real network data so using that data for anomaly detection may not work as intended [5].

One solution to the data availability problem is to generate synthetic data that is realistic [6, 2, 7]. Synthetically generated data that is realistic can improve the training process for IDS. Synthetic data can potentially address privacy concerns since it doesn't come from real sources. Additionally, synthetically generated data can be produced in arbitrary amounts and is not subject to availability constraints like real data.

To this point, much of the work for synthetic data generation, has been done through simulation or emulation. Simulation is where the synthetic data is generated via software such as OPNET where the network exists only in the software. Emulation is a physical network set up where the traffic occurring on it can be captured and used for testing and research purposes [5].

With the application of ML and DL to this area, one promising approach is the use of ML to perform the data generation. The underlying idea is to use real data to train the ML algorithm, then with ML, produce synthetic data that is realistic. This process is inherently challenging and some of the issues are explored in Section 2.2.2.

1.2 Problem Statement and Research Goals

An overarching problem for synthetic data generation is the question of how to evaluate the synthetically generated data. Specifically, how should we evaluate the similarity of generated data and real data? Specific to applying ML to data generation, throughout the generative process, some measurement, i.e. evaluation criteria must be iteratively applied to the data being generated in order to determine how the generative process is progressing.

This research focuses specifically on the problem of characterizing select evaluation criteria. There is not much research on this particular area, however, it is an important one. Quantitatively evaluating the evaluation criteria is important as we desire to use quantitative evaluation instead of expert review to determine the quality of synthetic data (i.e. its similarity to real data).

Thus, the goal of this research is to provide a set of metrics that can be used to inform and improve the process of generating synthetic semi-structured sequential data. Through quantitative evaluation, the generative process can be improved and eventually reduce or remove the need for human validation of results.

1.3 Research Questions and Hypothesis

Our hypothesis is as follows:

Hypothesis: There exist metrics with characteristics that allow for discrimination between real semi-structured sequential data and synthetically generated semi-structured sequential data.

This research seeks to determine what those metrics are by answering three research questions (RQs):

RQ 1 - What methods exist for measuring the "closeness" of real semi-structured sequential data to generated semi-structured sequential data?

RQ 2 - What characteristics should a potential metric possess?

RQ 3 - Given metrics for comparing data and the characteristics we want, what metrics perform best for temporally ordered, semi-structured sequential data?

1.4 Assumptions

The main assumption laid out in the research hypothesis in Section 1.3 is that there exists at least one (or more) metrics whose characteristics allow for the generation of synthetic semi-structured data. This is a hard problem that persists throughout all of synthetic data generation. While there has been much research in some areas of synthetic data generation and some metrics have been found to have nice characteristics, that research has not been applied specifically to semi-structured data.

1.5 Research Contributions

The purpose of this research is to augment the research on synthetic data generation being done in the areas of image generation and unstructured text generation. Several works explore quantitative evaluation of these kinds of data. However, little to no work exists in quantitatively evaluating metrics for semi-structured sequential text generation.

The contributions of this work are as follows:

- Adaptation of framework for evaluating metrics for image generation to semi-structured sequential data generation
- Evaluation of metrics for semi-structured sequential data generation
- "Black box" evaluation framework which is generator agnostic

1.6 Document Overview

In Chapter II, necessary background and related work is discussed. Chapter III lays out the methodology and the work necessary to perform the experiments. In Chapter IV, results and analysis of those results are presented. Chapter V discusses conclusions and future work recommendation.

II. Background and Literature Review

2.1 Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) is a multidisciplinary field generally associated with Computer Science involving elements from mathematics, psychology, philosophy, and several other fields [8]. AI is an umbrella term encompassing many elements, however, the fundamental function of AI can be reduced to two things: search and knowledge representation [9].

Machine Learning (ML) is an area that falls under the umbrella of AI. ML, also called Statistical Learning, is an approach that involves analyzing data to model a function in order to provide a prediction [10]. This process involves performing one of two tasks, regression or classification. The ML task of regression involves predicting a real valued output, for example, predicting income based on years of education [10]. Classification is a task that involves a qualitative or categorical prediction based on input features, for example, predicting whether a person will default on their credit card based on income [11].

Deep Learning (DL) is a class of ML algorithms that utilize multiple layers to extract features from the raw input data. Most DL involves some form of Artificial Neural Network (ANN), usually either a Recurrent Neural Network (RNN) or Convolutional Neural Network (CNN). DL is significantly more powerful for some applications than traditional ML, with the main difference being that no data engineering is required for DL.

2.2 Generative Methods

One application of DL that has emerged is the idea of generating synthetic data, whether it be images, text, or other forms of data. In this section we discuss some of these generative methods.

2.2.1 Generative Adversarial Networks

The concept of the Generative Adversarial Network (GAN) was introduced in 2014 by Ian Goodfellow in [12]. The GAN architecture is a DL architecture designed to generate synthetic images and is composed of a Generator, G, and a Discriminator, D. G takes as input real samples and generates a sample based on the real samples with some noise added in. G and D then play an adversarial game where G passes a sample to D and D must classify whether the sample belongs to the real sample distribution, $\mathbb{P}$, or the generated sample distribution, $\mathbb{G}$. This adversarial minimax game continues with both G and D until D can no longer successfully classify whether a sample belongs to $\mathbb{P}$ or $\mathbb{G}$. Figure 1 below depicts the GAN architecture.

Formally, D seeks to learn the generator's distribution, p_g over the input data x. Additionally, the input noise is defined as $p_z(z)$. G and D are defined as differentiable functions, represented in [12] as multilayer perceptrons, a form of DL architecture. Thus, $D(x)$ can be interpreted as the probability that x came from the real input data rather than p_g [12]. D is then trained to maximize the probability of identifying true positives and true negatives from the input data and samples from G. G is also simultaneously trained to minimize $\log(1 - D(G(z))$ [12]. The GAN architecture can then be defined as a two-player minimax game with value function $V(G, D)$ [12].

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))] \tag{1}$$

Two significant issues that exist in the original GAN architecture are the problems of *mode collapse* and *mode drop*. Generally, the real distribution, $\mathbb{P}_r$, is diverse and inherently multimodal [14]. Many generative methods to include GAN, can have generated distributions $\mathbb{P}_g$ that are less diverse than $\mathbb{P}_r$. Mode collapse and mode

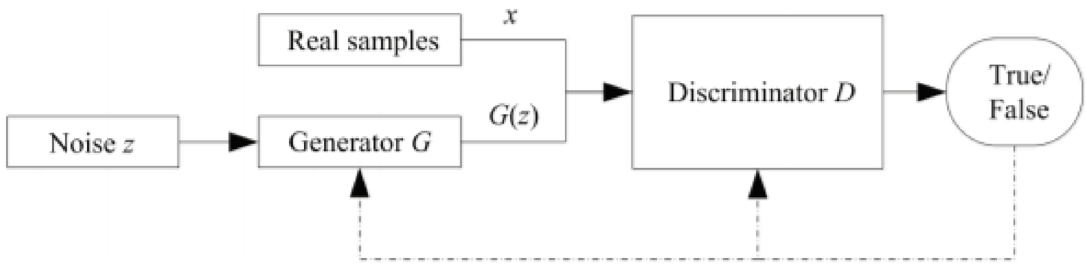

Figure 1: GAN architecture [13]

drop are typically how this lack of diversity gets manifested.

Mode collapse occurs when multiple modes of $\mathbb{P}_r$ are "averaged" into a single mode. This can result in $\mathbb{P}_g$ having modes that do not exist in $\mathbb{P}_r$ and having fewer modes overall. Mode drop occurs when harder to represent modes of $\mathbb{P}_r$ are left out by the generator because it has found certain modes that fool the discriminator [15].

2.2.2 Improvements on Generative Adversarial Networks

There have been many improvements and tweaks made to the original GAN since its introduction in 2014. In this section we describe some of the more significant improvements introduced in the GAN field.

Wasserstein GAN (WGAN)

One of the first major improvements to GAN was introduced by Arjovsky et al. in [16]. The overarching question they sought to answer was how to determine the closeness of the real distribution, $\mathbb{P}_r$ and the generated distribution $\mathbb{P}_g$. The fundamental change they applied was the introduction of the Wasserstein distance as the metric for determining the similarity between $\mathbb{P}_r$ and $\mathbb{P}_g$, hence it was termed WGAN [16]. The WGAN algorithm allows for a more stable learning process than the original GAN. Additionally, the WGAN virtually eliminates the problem of mode collapse, thus producing more stabilized training.

WGAN - Gradient Penalty (WGAN-GP)

Soon after the introduction of WGAN, Gulrajani et al. introduced some solutions to problems that arise from WGAN [17]. In particular, they address the problem of weight clipping in WGAN. WGAN used a process called weight clipping to enforce a Lipschitz condition on the discriminator during the training process. Gulrajani et al. shows that this process can induce undesirable behavior such as vanishing gradients, where the gradients quickly drop to zero and exploding gradients, where the gradients rapidly become very large. To alleviate this issue, they suggest utilizing a gradient penalty and term it WGAN-GP. [16]. Using WGAN-GP, the authors show better performance than weight clipping while also possessing more stable gradients than WGAN.

Sequence GAN (SeqGAN)

One limitation of GAN and its variants as originally presented was that they only worked on images and were not suitable for generating text or other forms of discrete data. This is because images exist in a continuous space so it is easy to tweak images with feedback from a gradient. With text and other discrete data, it is significantly more difficult to tweak the input with the gradient feedback.

Yu et al. present a solution to this problem of applying GAN to text generation with their variant called SeqGAN [18]. SeqGAN works by modelling sentences as numbered sequences and generating synthetic sequences which are then mapped to a corpus based on the numbers in the sequence. To perform this, SeqGAN incorporates Reinforcement Learning using a Monte Carlo (MC) Tree Search. The model for SeqGAN is depicted in Figure 2. Each number of the sequence is modelled as a state of the MC search with the end state reward being applied as the gradient in the transition to the next state. This solution bypasses the issue of gradient updates in the generator which are hard to perform on discrete data. In their results, they outperformed human scoring and had a significant p-value.

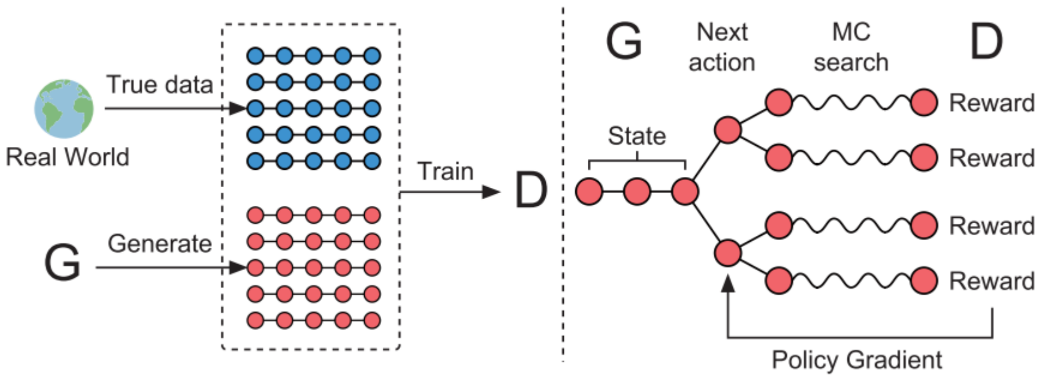

Figure 2: SeqGAN architecture [18]

CycleGAN

CycleGAN was developed to address the problem of unpaired image-to-image translation [19]. As in a typical GAN, the goal is to develop a map from $G : X \to Y$ such that at convergence Y is equivalent to $G(X)$. The novel aspect that Zhu et al. introduce is the idea of coupling this with an inverse mapping. They introduce the notion of pairing an inverse mapping $F : Y \to X$ such that when enforced with cycle consistency loss, $F(G(X)) \approx X$ and $G(F(X)) \approx Y$. As the name indicates, this mapping and inverse mapping creates a cycle between the two images. This approach is incredibly powerful and produces excellent synthetic images.

TreeGAN

TreeGAN, like SeqGAN, addresses the issue of applying GAN to discrete data. [20] However, rather than regular sentences, TreeGAN address the problem of syntax-aware sequence generation. To accomplish this, they couple the SeqGAN architecture with a Context-Free Grammar (CFG) to develop a parse tree, hence the name Tree-GAN. They then are able to translate the generated parse tree into a sequence that is valid according to the CFG. One of the methods of evaluation they demonstrate is the ability to generate 100% syntactically correct SQL queries that are completely synthetic. This beat the existing SeqGAN and other generative methods which could only generate about 70% syntactically correct SQL queries. This variant of GAN takes a large step in the direction of being able to develop discrete synthetic data that also requires syntactic correctness.

Evolutionary GANs

Wang et al. propose the idea of the Evolutionary GAN in [21]. Typical GAN architectures employ only one generator. However, the Evolutionary GAN utilizes and evolves an entire population of generators to play the adversarial game. The mutation operations are applied during the adversarial training process and the generators are updated based upon the mutations. They also introduce some new evaluation mechanisms for the generators in order identify the best performing generators and then select those generators for further training. Overall, the results of Evolutionary GAN are promising and show better training and performance results on most of the common image datasets used in GAN training.

2.2.3 Other Generative Methods

In this section, we describe some other non-GAN methods that have been used for generating synthetic data.

A Variational Auto-Encoder (VAE) is an architecture that can be used to generate a variety of types of data. VAEs work by encoding the input features down to a continuous latent space and then using random sampling to generate a new sample which is then decoded into a generated sample. [22]

Bachman and Precup in [23] present a generalized method for generating synthetic data. Titled *Data Generation as Sequential Decision Making*, their process builds models using neural networks trained with a form of guided policy search. Their models then generate predictions using an iterative process. They show some promising results, however, the quality of their results are lower than certain GAN variants.

Neural Text Generation is another generative method commonly used for text generation problems. Traditionally, neural text generation is built upon a RNN combined with Maximum Likelihood Estimation (MLE) [24]. However, this process can be difficult and requires a large amount of human correction for grammar and other errors.

Hajdik et al. in [25] present a method for neural text generation using minimal recursion semantics. Minimal Recursion Semantics allows for deeper semantic detail which corresponds to a better encoding for the model. Using this, the authors are able to achieve a higher Bilingual Language Evaluation Understudy (BLEU) score on evaluation than typical neural text generation models.

2.3 Applications of Generative Adversarial Networks

Aside from the original image synthesis task, GANs have been applied to many other fields. Image translation [19] and video generation [26] are some applications that have been explored.

In the discrete field, GANs have been applied to generating text [18]. They have also been used to generate syntax aware text such as SQL queries and Python code [20].

GANs have also begun to be applied in cybersecurity contexts. Yin et al. in [27] created a GAN variant called Bot-GAN for the purpose of botnet detection. Bot-GAN is composed of a generator that generates Transmission Control Protocol (TCP) flows which are then fed to the "discriminator", i.e. a botnet detector who classifies the input as real, anomaly, or fake. In their experiments, Bot-GAN has improved detection rate and a lesser false positive rate compared to existing botnet detection methods.

Ring et al. extend the idea of generating discrete data by generating synthetic network flow traffic using WGAN-GP [28]. Their approach uses three variants of WGAN-GP and is able to generate high quality synthetic network flows with two out of three of them. They also introduce a new evaluation method called domain knowledge checks. This approach defines several tests to ensure that the network flow data is high quality. For example, one test is that if the protocol is User Datagram Protocol (UDP) then no TCP flags can be set. While the introduction of the domain knowledge check is useful, the quantitative evaluation of the generated sample distribution is somewhat lacking. The authors default to measuring the difference between the real and generated data with Euclidean distance, which has been shown to not be as informative with higher-dimensional data [29].

2.4 Evaluation of Generative Adversarial Networks

The question of how to evaluate GANs has persisted since its introduction. Theis et al. were some of the first authors to discuss the quantitative evaluation of generative models in [29]. Part of the issue with objective evaluation of GANs is that many different metrics can be used and their use is not standardized. Theis et al. discuss three different evaluation techniques: average log-likelihood, Parzen window estimates, and visual fidelity of samples. One of their key contributions is that they show that these three techniques are largely independent so good performance in one doesn't correspond to good performance in another. they suggest that GANs should be evaluated for their specific application in order to get the best indicator of performance.

One of the first efforts at introducing a standardized evaluation criteria for GANs is the Inception Score, introduced by Salimans et al. in [30]. The Inception Score utilizes $\mathcal{M}$, the Inception Network image classification model [31], which is pre-trained on the ImageNet dataset [32]. The equation for the Inception Score on the generated distribution, $\mathbb{P}_g$ is defined in [30] and given by:

$$IS(\mathbb{P}_g) = \exp(\mathbb{E}_{x \sim \mathbb{P}_g}[KL(p_{\mathcal{M}}(y|x)||p_{\mathcal{M}}(y))]) \tag{2}$$

The Inception Score correlates to human evaluation of the generated images and is a useful tool providing a good initial benchmark for GAN evaluation. However, it falls short in a few aspects. First, the assumption of the model being trained on the ImageNet dataset means that the Inception Score does not generalize well to images that are not part of ImageNet [15]. Additionally, this reliance on ImageNet means that the Inception Score is not sensitive to the existing distribution of the training data labels [33].

The Fréchet Inception Distance (FID) proposed by Heusel et al. in [34] is another effort to provide a standardized evaluation metric for GANs. The FID scores samples by embedding them into a feature space (originally a layer of the Inception model). It then assumes that the embedding can be represented as a continuous multivariate Gaussian. Assuming that X represents $\phi(\mathbb{P}_r)$ and Y represents $\phi(\mathbb{P}_g)$, where ϕ is the embedding into the desired feature space, the FID is defined in [34] by:

$$\text{FID}(X, Y) = \left\|\mu_x - \mu_y\right\| + \mathbf{Tr}(\mathbf{C}_X + \mathbf{C}_Y - 2(\mathbf{C}_X\mathbf{C}_Y)^{\frac{1}{2}}). \tag{3}$$

Where (μ_x, C_x) and (μ_y, C_y) are the means and covariance matrices of X and Y respectively.

Like the Inception Score, the FID has been shown to correlate to human evaluation and is also more robust to noise than the Inception Score [33].

2.5 Evaluation Metrics

In this section, we describe several metrics that are of use in the fields on Network Intrusion Detection, GAN, and other statistical applications.

Mathematically, the term distance metric has a specific definition. According to *The Encyclopedia of Distances* [35], a *distance metric* is a function $d(x,y) : X \times X \to \mathbb{R}$ satisfying the following properties:

1. $d(x,y) \geq 0$ (Non-negativity)

2. $d(x,y) = 0 \Rightarrow x = y$ (Identity of indiscernibles)

3. $d(x,y) = d(y,x)$ (Symmetry)

4. $d(x,y) + d(y,z) \geq d(x,z)$ (Triangle inequality)

Not all of the metrics described in this section satisfy the above definition of *distance metric*, however, we use the term metric for ease of reference.

2.5.1 Power Distances

The first category of commonly used distances is the Power Distances. The Power (p,r)-distance is a distance on $\mathbb{R}^n$ defined by

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{r}}. \tag{4}$$

When $p = r \geq 1$, this distance is the l_p metric and is a proper distance metric. This encompasses the Manhattan ($p = r = 1$) and Euclidean ($p = r = 2$) distances commonly used for many applications (particularly Euclidean since it is the intuitive definition of physical distance in three or less dimensions). When $0 < p = r < 1$, this distance is called the fractional l_p-distance, and is used for high dimensional data [35].

The Mahalanobis distance [36] is defined in [35, 37] as shown in Equation (5). The Mahalanobis distance is a generalization of the power distance to multiple dimensions. Within the equation, x and y are assumed to be of size n, A is a positive-definite matrix (generally the covariance matrix of x and y), $\det A$ is the determinant of A, and ^{T} indicates the transpose of the matrix.

$$\left\|x - y\right\|_A = \sqrt{(x - y)A^{-1}(x - y)^T} \tag{5}$$

When A is the identity matrix, the Mahalanobis distance is the Euclidean distance [35].

2.5.2 Probability Distribution Measures

χ^2-distance is a commonly used distribution distance. Shown in Equation (6), x and y are vectors of length n, $p(x_i)$ is the probability of that the i^{th} element of x occurs and $p(y_i)$ is the probability of that the i^{th} element of y occurs [37].

$$d(x, y) = \sum_{i=1}^{n} \frac{(p(x_i) - p(y_i))^2}{p(y_i)} \tag{6}$$

A simplified χ^2 distance introduced by Wang and Stolfo [38] is a version of the χ^2 distance that is less computationally intensive. The simplified χ^2 distance, $m_{\mu,\sigma}$ is defined in Equation (7). x is a vector containing all of the dimensions of a single observation, μ is a vector that represents the center of mass of all observations, $n = |x| = |\mu|$, and $d(x_i, \mu_i)$ represents the difference between the i^{th} element of x an μ.

$$m_{\mu,\sigma}(x) = \sum_{i=1}^{n} \frac{d(x_i, \mu_i)}{\sigma_i} \tag{7}$$

Entropy is another form of distance measure that falls under probability measures. Entropy of a random variable is calculated as shown in Equation (8), where $p(x)$ is the probability a random variable X takes on the value x [39].

$$H(X) = -\sum_{x \in X} p(x) \log_a p(x). \tag{8}$$

Using the definition of entropy in Equation (8), we can then define Standardized entropy [40] as shown in Equation (9). This form of entropy normalizes the entropy calculation so that the size of the random variable doesn't affect the value of the entropy calculation.

$$H_s(X) = \frac{H(X)}{\log_a m}. \tag{9}$$

Perplexity is another measure related to Entropy and can be interpreted as a measure of how well a probability distribution predicts a given sample [41]. Using the definition of Entropy as the function $H(X)$ in Equation (8), Perplexity is defined as shown in Equation (10).

$$Perp(X) = 2^{H(X)}. \tag{10}$$

The Wasserstein distance [42], also known as Earth Mover's Distance, is another form of probability measure. Analogizing two probability distributions to two piles of dirt, the Wasserstein distance between the two piles can be thought of as the amount of dirt that has to be moved times the distance the dirt is moved to transform one pile into the other. The Wasserstein distance has been used as a GAN evaluation metric, namely in the WGAN and WGAN-GP due its desirable properties of continuity and differentiability everywhere in its domain [16]. The Wasserstein distance is defined in Equation (11), where $\Gamma(u, v)$ is the set of joint probability distributions whose marginals are $u(x)$ and $v(y)$.

$$\inf_{\pi \in \Gamma(u,v)} \int_{\mathbb{R}\times\mathbb{R}} |x - y| d\pi(x, y) \tag{11}$$

Kullback-Leibler Divergence (KLD) or "information gain" between probability distributions is another metric that has been considered for GAN evaluation. As the name implies, it is a measure of dissimilarity between two probability distributions. One issue with the KLD is that it lacks the symmetry and Triangle Inequality properties making it undesirable in some cases. The KLD between P_1 and P_2 over a domain X is defined in Equation (12) [43].

$$KLD(P_1, P_2) = \sum_{x \in X} p_1(x) \log_a \frac{p_1(x)}{p_2(x)} \tag{12}$$

Jensen-Shannon Divergence is a smoothed, well-behaved, symmetric, and bounded version of the KLD [44]. Let P and Q be probability distributions and $D(P||Q)$ be the KLD as shown in Equation (12). The formula for the Jensen-Shannon Divergence is shown in Equation (13).

$$JSD(P, Q) = \frac{1}{2}D(P||M) + \frac{1}{2}D(Q||M) \tag{13}$$

Where $M = \frac{1}{2}(P + Q)$, the arithmetic mean of P and Q. The square root of the Jensen-Shannon Divergence is known as the Jensen-Shannon Distance (JSD) and is a proper distance metric as it obeys the Triangle Inequality [45].

2.5.3 Other Distance Measures

Cosine similarity [46] is another metric commonly used for document similarity and other applications in network intrusion detection. Cosine similarity is defined as shown in Equation (14).

$$\cos\phi = \frac{\langle x, y\rangle}{\sqrt{x^2}\cdot\sqrt{y^2}} \tag{14}$$

Kernel Maximum Mean Discrepancy (MMD) [47] is another metric that has been used as a GAN evaluation metric. The kernel MMD is also a measure of dissimilarity between two probability distributions . Let $X = \{x_1, ..., x_{n_1}\}$ and $Y = \{y_1, ..., y_{n_2}\}$ and let ϕ be a fixed kernel function (typically the Radial Basis Function).

$$\mathrm{MMD}(X, Y) = \frac{1}{n_1}\sum_{i=1}^{n_1}\phi(x_i) - \frac{1}{n_2}\sum_{i=1}^{n_2}\phi(y_i) \tag{15}$$

2.6 Related Work

In this section we detail related work that has been done in the fields of synthetic traffic generation and quantitative evaluation of generative methods. This is not an extensive list of all work in these fields, just a reference of work relative to the research we are conducting.

2.6.1 Synthetic Data Generation

Synthetic Data generation, particularly for cybersecurity purposes is a heavily researched and discussed problem as documented in [6, 2, 7]. Wurzenberger et al. in [48] discuss a method of generating synthetic network log files for Intrusion Detection System training. Their approach includes a combination of log-line clustering and Markov chain simulation to develop synthetic log files. The real value of this approach is that they are able to intake a small amount of real log data and then augment it with synthetically generated data to enhance the training process of the Intrusion Detection System (IDS). This work focuses on evaluating the clustering algorithm rather than the quality of the semi-synthetic logs they generate. Specifically, they focus on the clustering by determining if the log lines pertain to the cluster description.

Kulkarni and Garbinato [49] explored the process of generating synthetic mobility traffic using RNNs. They were interested in generating synthetic location data in order to generate realistic location trajectories since privacy concerns generally prevent the use of actual location data. They utilized an RNN due to its ability to learn long term patterns in sequential data. With this approach they were successfully able to generate synthetic location data. They claim that their synthetic data possessed the same statistical characteristics as the real data, however, they do not specifically say what characteristics. When trained on the synthetic data though, their model predicted the same sleep and wake cycles, movement periodicity, and variance in the movement distance magnitudes as the model did when it was trained on real data.

Garcia-Torres in [7] discusses the idea of generating synthetic network data with a GAN. The goal of this work was to explore the possibility of generating synthetic continuous, discrete, and text network data. The author utilizes two forms of WGAN to carry out the data generation experiments. The generation of continuous and discrete data was overall successful. To determine how well the generated data preserves the real data distribution, the author utilizes the Wasserstein distance. However, in order to evaluate the similarity of the generated data and real data features, the author uses Euclidean distance, which has been shown to not be a very useful or informative metric for higher dimensional data [29].

2.6.2 Quantitative Evaluation of Generative Methods

One important aspect of GAN evaluation is the quantitative evaluation. Human judgement is subjective and not always the best indicator of how good or realistic the synthetically generated data is. To this end, there has been some work within the GAN community focused on developing standardized methods and criteria for evaluating GANs. Arjovsky and Bottou [50] present some approaches for standardized training and evaluation of GAN, but focus mainly on standardized training while briefly mentioning evaluation with the Wasserstein distance.

Kawthekar et al. [41] discuss a framework for evaluating generated text. Their approach is not only limited to GAN as they also evaluate text generated from Scheduled Sampling and RNN text generation. Their framework focuses on three different evaluation metrics: cross-entropy loss, perplexity, and human judgement. In their results, they found that cross-entropy and perplexity tended to underperform on the test set. However, despite the poor performance, they found that the human judgement found the generated text to be more realistic than suggested by the test performance. This suggests that other metrics may work better for demonstrating performance.

Semeniuta et al. [51] explore the problem of GAN evaluation from the angle of evaluating the text generated by the GAN. They discuss how the standard metric for language generation evaluation, the BLEU score [52] falls short in GAN application. They demonstrate that the BLEU scores do not reflect any degradation of semantics in the generated samples. To remedy this, they propose other metrics that better capture the real quality of generated samples. Their work evaluates three metrics, the BLEU score, Language Model score, the FID adapted to use the feature space from a sequence embedding model, and human evaluation. In their evaluations, they found that FID was the best metric for evaluating the generated text, corresponding highly with human evaluation.

The closest related work to this research is the work by Xu et al. in [15]. Xu et al. present a quantitative evaluation of several GAN metrics for image generating GANs. Like many others, they recognized that there was no evaluation of the metrics being used for GAN, other than analysis of the theoretical properties of the metrics themselves. The authors evaluate six commonly used GAN metrics: Inception Score, Mode Score (improved version of the Inception Score), Kernel MMD, Wasserstein distance, FID, and the 1-Nearest Neighbor classifier. One important feature of the metrics that they choose is that all of the metrics are "model agnostic", i.e. they can be calculated by directly inputting the samples into the model like a black box. This allows the framework they present to be applied to more broad generative methods and not just GANs [15].

Xu et al. [15] conduct experiments on the chosen metrics in two different feature spaces. First is "pixel space", a direct comparison pixel-to-pixel of the input images. The second space is termed "convolutional space", the space of the features extracted by their chosen model, a 34-layer ResNet model. The reason the authors include the pixel space is to demonstrate that it is not a suitable space for evaluating the metrics as all of them fail in pixel space.

The experiment setup that Xu et al. utilize evaluates their chosen metrics in several categories. Discriminability, the ability to discriminate between real and generated images is the first and arguably the most important aspect of a GAN evaluation metric. Behaviors under the conditions of mode collapse and mode drop (described in Section 2.2.1) are also evaluated for all of the metrics. The authors also evaluate robustness to transformations by performing random translations to the input images and observing the behavior of the metrics. They also evaluate the efficiency of the metrics in two ways. First, they examine the wall-clock time required for each metrics against an increasing number of evaluated samples. Second, they examine the scores of each metric as the sample size increases to determine how many samples are required for each metric to reach a “good” score. The authors also evaluate each of the metrics in their ability to detect overfitting.

Their findings were that overall, the kernel MMD performed well in the convolutional space with FID also performing well in all categories except that it is unable to detect overfitting. The most important conclusion that the authors make is that the feature space in which the metrics are calculated is the most crucial aspect of metric performance.

2.7 Summary

The work of Xu et al. [15] forms the framework for the research performed in this work. While Xu et al.’s framework applies to image GANs, the goal of this research is to apply this experimental approach to various types of semi-structured sequential data. Conducting this research will lay an empirical base for choosing what metrics are useful for future research seeking to generate synthetic network data. The need for this is made clear by the default reliance on Euclidean distance as the evaluation for measuring how “good” synthetically generated data is [7, 28].

III. Methodology

The focus of this chapter is to outline the experimental methodology for this research. As mentioned in Section 2.7, the research methodology here is based on the research conducted by Xu et al. in [15], with this research seeking to apply their methodology on semi-structured sequential data rather than images.

3.1 Methodology Overview

Our overarching research methodology is based on the Cross-Industry Standard Process for Data Mining (CRISP-DM) [53]. As the name states, CRISP-DM is a general process that can be applied to broad areas of research in order to guide the data mining process. Figure 3 presents a flowchart of the CRISP-DM methodology.

The background and literature review from Chapter II fall into the Business Understanding portion of the CRISP-DM cycle. Details about the dataset described in Section 3.3 fall under the Data Understanding portion of the CRISP-DM cycle. Data pre-processing and data generation (section 3.4 and section 3.5 respectively) fall under the Data Preparation portion of the CRISP-DM cycle. The experiments performed in this research, described in chapter IV fall into the Modeling section of the CRISP-DM cycle. Analysis of the results and suggestions for future work fall into the Evaluation portion of the CRISP-DM cycle.

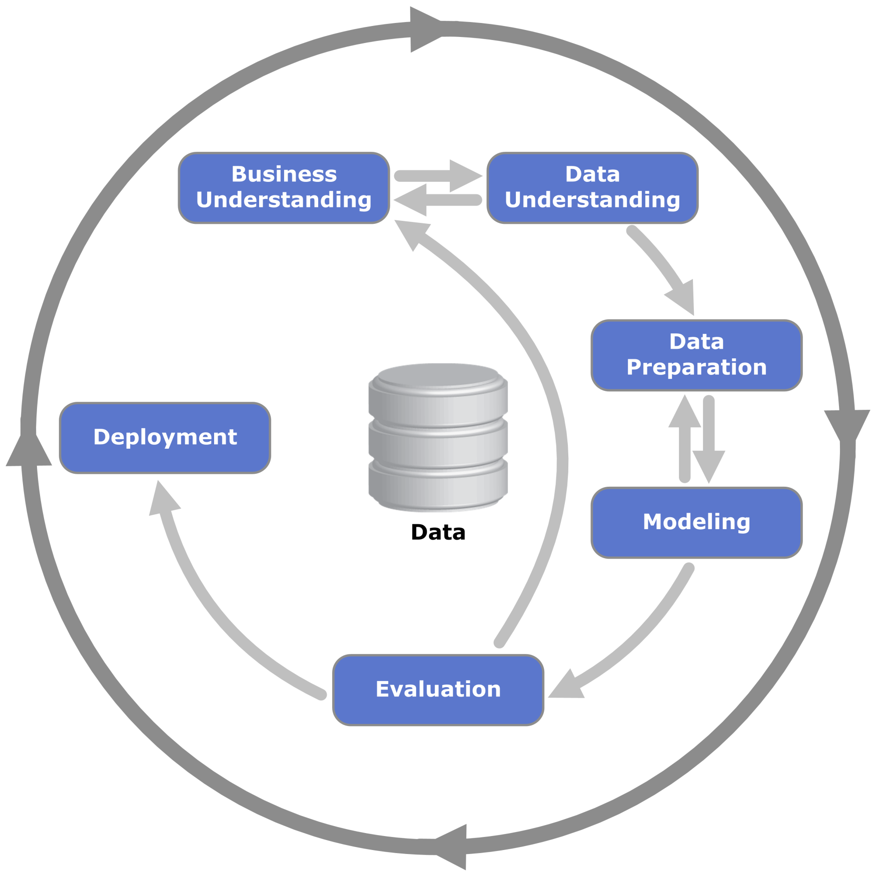

Figure 3: Flowchart of the CRISP-DM process [54].

3.2 Research Questions

The research questions (RQs) that this research seeks to answer are the following:

RQ 1 - What methods exist for measuring the "closeness" of real semi-structured sequential data to generated semi-structured sequential data?

RQ 2 - What characteristics should a potential metric possess?

RQ 3 - Given metrics for comparing data and the characteristics we want, what metrics perform best for temporally ordered, semi-structured sequential data?

In order to answer these questions, we explore the dataset(s) being utilized, the data pre-processing and generation process required for the experiment, the metrics being evaluated, the characteristics we are examining, and overview the experiments themselves. A detailed user guide for reproduction of the data pre-processing, data generation, and experiments described in this research is provided in Appendix A.

3.3 Data Understanding

3.3.1 Network Events

The dataset used in this research is the Unified Host and Network Data Set (UH-NDS) from Los Alamos National Labratory [55]. This dataset is freely available and is also fairly large. This particular dataset was chosen because it contains two different types of data as well as its currency and general representation of semi-structured sequential network data. The dataset consists of two portions: Network Event Data and Host Event data.

The Network Event portion of the dataset contains records and statistics for network connections between different devices. Details about the fields of this portion of the dataset are shown in Table 1. For this research, the *Time* field was removed. The *Duration*, **Packets*, and **Bytes* fields are all 32-bit unsigned integers. The *Protocol* field is typically an unsigned integer with standard transport layer port numbers ranging from 0 - 65,536, however, sometimes the port number is prefaced with the text “Port”. The **Device* fields are typically ASCII text “Comp” followed by a 5 or 6 digit integer. In some cases, the device is identified just as “Mail” as in Figure 4 or “ActiveDirectory”, etc.

An example of the Network Event portion of the dataset is shown in Figure 4. This portion of the dataset is representative of numeric semi-structured sequential network data, which is commonly the type of data in packet capture and NetFlow files. Details about pre-processing of the data for the experiment are described in Section 3.4.

Table 1: **Field names, descriptions, and data formats for features of the UHNDS Network Events dataset.**

Field Name	Description	Format
Time	The start time of the event in epoch time format	`int32`
Duration	The duration of the event in seconds.	`int32`
SrcDevice	The device that likely initiated the event.	ASCII text
DstDevice	The receiving device.	ASCII text
Protocol	The protocol number.	`int32`
SrcPort	The port used by the SrcDevice.	ASCII/`int32`
DstPort	The port used by the DstDevice.	ASCII/`int32`
SrcPackets	The number of packets the SrcDevice sent during the event.	`int32`
DstPackets	The number of packets the DstDevice sent during the event.	`int32`
SrcBytes	The number of bytes the SrcDevice sent during the event.	`int32`
DstBytes	The number of bytes the DstDevice sent during the event.	`int32`

```
761,4434,Comp132598,Comp817788,6,Port12597,22,89159,85257,15495068,69768940
764,13161,Comp178973,Comp164069,17,137,137,325,0,30462,0
765,14369,Comp492856,Mail,6,Port30344,443,227,214,32300,9844
765,14431,Comp782574,Mail,6,Port28068,443,1637,3313,75302,1220077
765,17056,Comp378125,Mail,6,Port28068,443,3848,4096,177008,1441295
765,17087,Comp378125,Mail,6,Port41392,443,571,275,60842,12650
765,18105,Comp492856,Mail,6,Port30344,443,292,298,40698,13708
765,18633,Comp378125,Mail,6,Port41392,443,622,310,70370,20963
765,22042,Comp782574,Mail,6,Port28068,443,2142,4299,98532,1423831
```

Figure 4: Network Events dataset in raw format.

3.3.2 Host Events

The Host Event Data section of the UHNDS is representative of semi-structured sequential text data. This type of data is commonly seen in system logs or other types of log files where text and numerical data is combined. Formally, semi-structured data is a form of structured data that does not obey the typical structure of relational databases or other data tables. The key element that defines semi-structured data is that it contains tags that separate the semantic elements of the data [56]. Examples of this type of data are Extensible Markup Language (XML), JavaScript Object

Notation (JSON), and email. As can be seen in Figure 4 and Figure 5 both datasets fit the definition of semi-structured data. An example of the Host Events data is shown in Figure 5. The raw data is formatted in JSON. In total, there are 20 different fields of data within the Host Events portion of the dataset. Some of these shown in Figure 5 are: *EventID, UserName, DomainName*, etc. The main difference between the Host Events and Network Events data, aside from the data type, is that the Host Events data describe specific events on the network such as a user log on while the Network Events data describe Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) flows.


```
{"EventID": 4769, "UserName": "User624729", "ServiceName": "Comp883934$", "Domai
nName": "Domain002", "Status": "0x0", "Source": "Comp309534", "Computer": "Activ
eDirectory", "Time": 2}
{"EventID": 4776, "UserName": "Scanner", "DomainName": "Domain002", "Status": "0
x0", "Computer": "ActiveDirectory", "AuthenticationPackage": "MICROSOFT_AUTHENTI
CATION_PACKAGE_V1_0", "Time": 2}
{"EventID": 4672, "UserName": "ActiveDirectory$", "LogonID": "0x2e66398d", "Doma
inName": "Domain002", "Computer": "ActiveDirectory", "Time": 2}
{"EventID": 4624, "UserName": "ActiveDirectory$", "LogonID": "0x2e66398d", "Doma
inName": "Domain002", "LogonTypeDescription": "Network", "Computer": "ActiveDire
ctory", "AuthenticationPackage": "Kerberos", "Time": 2, "LogonType": 3}
{"EventID": 4634, "UserName": "ActiveDirectory$", "LogonID": "0x2e66398d", "Doma
inName": "Domain002", "LogonTypeDescription": "Network", "Computer": "ActiveDire
ctory", "Time": 2, "LogonType": 3}
{"EventID": 4624, "UserName": "User380010", "LogonID": "0x9f17415", "DomainNam
e": "Domain002", "LogonTypeDescription": "Network", "Computer": "Comp966305", "A
uthenticationPackage": "Kerberos", "Time": 2, "LogonType": 3}
{"EventID": 4634, "UserName": "User380010", "LogonID": "0x9f17415", "DomainNam
e": "Domain002", "LogonTypeDescription": "Network", "Computer": "Comp966305", "T
ime": 2, "LogonType": 3}
```


Figure 5: Host Events data in raw JSON format.

3.4 Data Preparation

Since the Network Event data is not all numeric to begin with, some pre-processing is required. The non-numeric data occurs in the Device, Protocol, and Port sections of the data. In cases where the text is followed by a number (e.g. Comp178973), we simply remove the text since the numbers are also unique identifiers. For example, "Comp178973" becomes 178973 after processing. We chose this approach since it was the most straightforward and each of the numbers after the text were unique. For the text data that does not contain numbers, the text is converted to integers by converting the text to hex and then taking the first 5 nibbles and converting that number to a base 10 integer. For example, "EnterpriseAppServer" becomes 456e7 in hex which is 284391 in base 10. We took this approach in an effort to create a generalized conversion method to the data that could be applied without knowing beforehand exactly what text data would show up. Table 2 below contains the full set of mappings between non-numeric and numeric data used in the experiments.

We chose to not apply pre-processing other that removing the "Port" text to the *Port* field for simplicity. One step that is utilized in [28] for pre-processing the *Port* field is to convert the value of the field to an element of the $[0, 1]$ range by dividing the value of the field by 65,536 (e.g. Port 80 becomes $\frac{80}{65536} = 0.00122$). The authors of [28] acknowledge that a field like port number is actually a categorical value, however, the numerical nature of the port number lends itself to normalization in this method. There are many ways to encode categorical variables and a list of them can be found at [57]. Normalization to the $[0, 1]$ range is generally a good thing to apply to data for Machine Learning (ML) approaches. We did not discover this normalization method until the data pre-processing was complete and had started to run experiments so we chose not to implement this. We instead utilize a scaling function from Scikit-learn [58] to perform normalization of the data. Specifically we chose the `RobustScaler()` function [59]. The `RobustScaler()` removes the median and scales the data between the first and third quartiles. Each feature is then centered and scaled using the appropriate statistics [59]. We chose this function over the

`StandardScaler()` because the `RobustScaler()` is more robust to outliers, of which there are many in the UHNDS dataset.

Table 2: Mappings of original values to numeric values for Network Events dataset.

Original Value	Numeric Value
EnterpriseAppServer	284391
ActiveDirectory	267831
VPN	56566
VSCanner	353590
CompXXXXXX	XXXXXX
IPXXXXX	XXXXX
PortXXXX	XXXX

Pre-processing the Host Events data requires more work due to the presence of text data. In order to convert the Host Events data to a numeric form, we utilize Term Frequency - Inverse Document Frequency (TF-IDF) to convert text to numbers.TF-IDF is a method of measuring the importance of a word in a collection of documents [60]. TF-IDF, as the name implies, has two components: Term Frequency (TF) and Inverse Document Frequency (IDF). TF is used within a single "document" and is calculated as shown in Equation (16). First, let N be the total number of documents, let f_{ij} be the number of times word i occurs in document j. The TF of term i in document j is defined in [60] as:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \tag{16}$$

IDF for a given term can be calculated as shown in Equation (17). Suppose that term i appears in n_i out of N documents in the set of documents. The IDF is defined in [60] as:

$$IDF_i = \log(\frac{N}{n_i}) \tag{17}$$

The TF-IDF for term i in document j can then be calculated as $TF_{ij} \times IDF_i$ [60]. This multiplication of TF and IDF produces a balanced representation of the importance of a given term in a document.

TF-IDF is normally applied to unstructured text such as sentences and documents. The results of this is that the resulting size of the TF-IDF matrix can vary from collection to collection. Since the data we utilize is semi-structured, we need the TF-IDF matrix to have a repeatable and constant size. To accomplish this, we adapted the TF-IDF conversion process in the following way. First, define a single "document" to be a single column of the data (e.g. *LogonID*). This then allows us to define the total number of documents in the collection to be the number of columns in the Host Events dataset. Thus, the TF of a term within a single column is the number of occurrences of that term in the column divided by the number of distinct terms in the column. Similarly, we then define IDF of a term as the log of the total number of columns divided by the number of columns that term appears in.

3.5 Modeling

For this experiment, we utilize 1,000 line log samples as the standard size. In initial tests, 1,000 line samples showed best performance for stable calculation of metrics. For both datasets, in order to build our repository of "real" samples, we iterate through the processed real data, partitioning it into contiguous 1,000 line samples. Due to the large size of both portions of the dataset, we are able to build a set of 10,000 real samples from the real data.

In order to construct the "generated" log samples, we iterate through the entire real set of log samples tracking the global minimum and maximum for each column.

Once these values have been determined we then generate 1,000 line samples through uniform random sampling between the global minimum and maximum for each column. In order to have equally sized datasets, we also generate 10,000 samples. We also explored generating random samples from a Normal distribution based on the global mean and variance of the real data and the results of the Discriminative experiment are discussed in Section 4.2. However, the results were not significantly different from the results with the uniform random samples, thus we did not include them in the Efficiency experiment.

In the future, the generated samples would ideally be provided by a Generative Adversarial Network (GAN). However, the focus of this work is on the evaluation framework and not the quality of the generated samples. The real and generated samples are provided via a "black box" so that the evaluation framework is generator agnostic, similar to [15].

3.6 Metrics

For the experiment, eleven metrics have been chosen for evaluation and are listed below. The details of the metrics and their equations are provided in Section 2.5.

- Power distance (Equation (4)): Euclidean ($p = r = 2$), Manhattan ($p = r = 1$), fractional l_p distance ($p = r = 0.5$ and $p = r = 0.75$)
- Mahalanobis distance (Equation (5))
- Cosine similarity (Equation (14))
- Wasserstein Distance (Equation (11))
- Maximum Mean Discrepancy (MMD) (Equation (15))
- Fréchet Inception Distance (FID) (Equation (3))

- Entropy (Equation (8))

- Perplexity (Equation (10))

The Power distance measures are selected as they are representative of general use distance metrics commonly used on network data [37]. The Mahalanobis, Entropy, and Perplexity are selected since they are probability distribution measures. The MMD, FID, and Wasserstein distances are chosen because of their extensive use for GAN evaluation in the image context.

3.7 Evaluation

Following the methodology from [15], there are four categories in which it is useful to evaluate metrics for GAN use: Discriminative ability, efficiency, generative failure detection, and overfitting detection. This research explores the Discriminative Ability and Efficiency experiments and the details of these experiments are laid out in Section 3.9. The Generative Failure Detection and Overfitting Detection experiments are left as future work.

3.8 Data Transformations

In order to fully explore the behavior of the metrics in the experiments, we perform five different transformations on the data. These transformations act as "feature spaces" to calculate the metrics in since we do not have a Deep Learning (DL) model with layers that we can use as the feature space. The five transformations are: untransformed, Square Root (SQRT), logarithm, Principal Component Analysis (PCA), and Fast Fourier Transform (FFT).

As the name implies, for untransformed, we take the original pre-processed data. This transformation is similar to the "pixel space" from [15]. For SQRT and logarithm transformations, we take the SQRT and natural log using the `NumPy` library. For the PCA transformation we conduct a PCA on the samples using the `Scikit-Learn` `PCA()` function. For the FFT transformation we use the `NumPy` `fftn()` function to perform a Discrete Fourier Transform on the data.

3.9 Experiment overview

The purpose of the experiments is the following. Given a class of network data, evaluate and rank the metrics based on performance. Performance of the metrics is evaluated in the areas of discriminative ability (Chapter IV) and two categories of efficiency (Section 4.2).

As mentioned earlier, we utilize 1,000 line log samples as the standard length for all metric evaluations. This line count can be thought of as being analogous to image size when working with images.

3.9.1 Discriminative Behavior

In order to evaluate the discriminative ability of a metric, we use the following approach. A flow diagram of the experiment is shown in Figure 6. We create two sets of $n = 1000$ samples, S_{r_1} and S_{r_2} with S_{r_1} and S_{r_2} both made up of real samples and generate $S_r = d(S_{r_1}, S_{r_2})$ for each metric d. S_r is then composed of 1,000 metric distances between real samples (Real-Real (R-R) samples). We then build two new sets S_{r_3} and S_{g_1}, where S_{r_3} is composed of n real samples and S_{g_3} is composed of n generated (fake) samples. From these sets, we compose a second set, $S_g = d(S_{r_3}, S_{g_1})$ for each metric d. S_g is thus made up of n samples of metric distances between real samples and generated samples (Real-Fake (R-F) samples). Each of the sets is built of randomly chosen samples from a repository of 10,000 samples with no duplicates. Randomness is controlled with a random seed for repeatability.

From the samples we create two discrete probability distributions, $\mathbb{P}_r$ for the R-R samples and $\mathbb{P}_g$ for the R-F samples. To create the distributions we split the values into 100 equally sized bins between $\min(S_r, S_g)$ and $\max(S_r, S_g)$. This way, both distributions are split into equally sized bins. 100 bins was chosen because in pilot tests, 50 bins didn't produce a fine enough distribution and 200 bins was too fine. The number of elements in each bin is used to generate the histogram figures in Chapter IV using the Matplotlib `hist()` function. The counts for the histogram are then normalized by dividing by 1,000 in order create a Probability Mass Function with a sum of 1 for the Jensen-Shannon Distance (JSD) calculation. We use a base 2 calculation for JSD so that the values from JSD are bounded between 0 and 1. Prior to binning for the histograms, we take the natural log of all the metric values to make the Probability Mass Function (PMF)s nicer. After generating $\mathbb{P}_r$ and $\mathbb{P}_g$, we then calculate the JSD between $\mathbb{P}_r$ and $\mathbb{P}_g$, $JSD(\mathbb{P}_r, \mathbb{P}_g)$. If the two sample distributions, $\mathbb{P}_r$ and $\mathbb{P}_g$, are identical, then the JSD between them is zero. We can then judge the discriminative ability of the metric on the JSD score. The closer the JSD score is to 1, the more discriminative the metric. Conversely, the closer the JSD score is to 0, the less discriminative the metric is.

For repeatability, this process is repeated 10 times. The mean JSD score is reported along with the minimum, maximum, and the range (maximum - minimum).

We choose to use JSD over Kullback-Leibler Divergence (KLD) because JSD fits the definition of a distance metric. Since KLD is not symmetric and doesn't follow the Triangle Inequality, ordering of KLD values is not possible. However, since JSD is a metric, we can order JSD values. Additionally, the JSD is bounded between 0 and 1, so interpretation of the JSD is more intuitive than the KLD. The JSD, along with all other code is written in Python. Exact code for the JSD can be found in Appendix B.

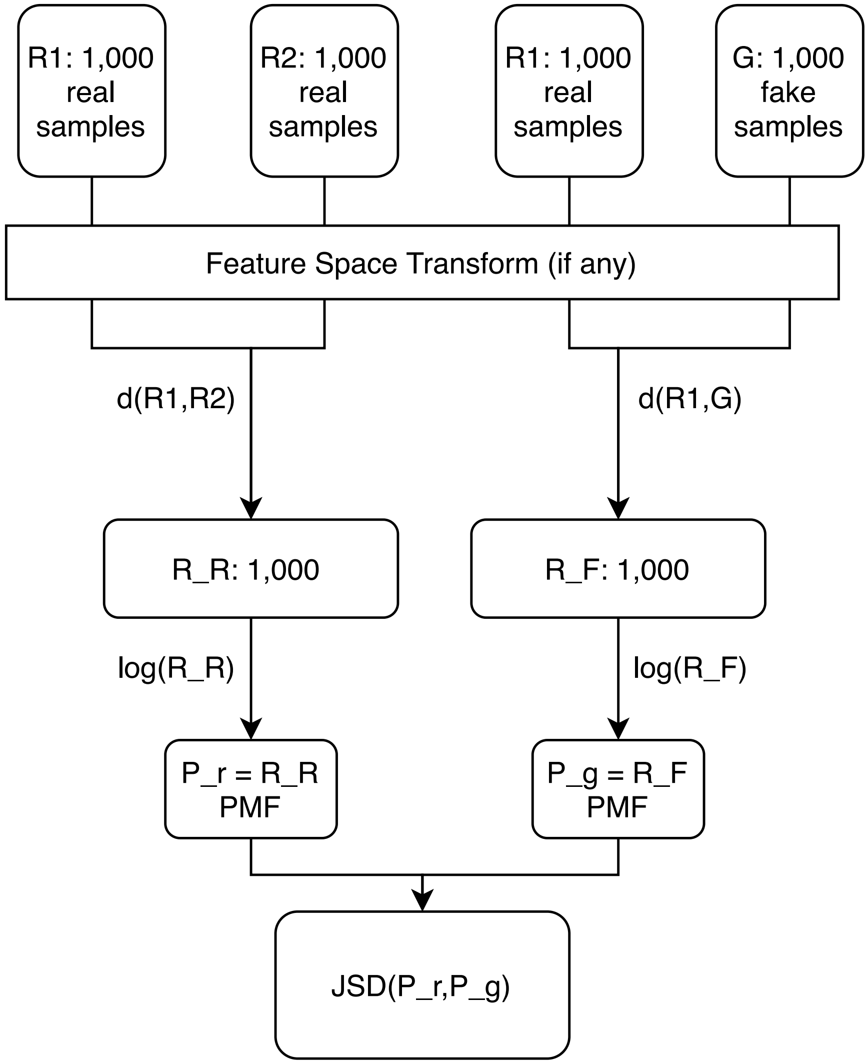

Figure 6: Flow Diagram of Discriminative Experiment

3.9.2 Efficiency

For efficiency, we explore two different categories of efficiency. First we examine time efficiency by examining the wall clock time for metric calculation based on the number of lines in the sample (sample length). Second, we examine the sample efficiency of the metric as we increase the number of samples.

The time efficiency experiment examines the wall-clock time for metric calculation as the size of the sample or number of lines (termed sample length) in the sample increases. For this experiment we use sample lengths of $[100, 500, 1,000, ..., 5,000]$. To calculate these runtimes, we calculate the wall-clock time to score a set of 10 samples of a given length and then take the average to find the average wall-clock time to calculate a given metric on a single sample. This is then repeated 10 times and the average is reported.

For sample efficiency, we explore the behavior of the JSD score by repeating the discriminative behavior experiment for increasing number of samples. For this experiment, we calculate the JSD score between $\mathbb{P}_r$ and $\mathbb{P}_g$ for an increasing number of samples $[100, 500, 1000, ..., 5,000]$.

3.10 Expected Outcomes

Our research hypothesis is the following: There exist metrics with characteristics that allow for discrimination between real semi-structured sequential data and synthetically generated semi-structured sequential data. In this section we detail the expected outcomes of our experiments and how they support our research hypothesis.

The expected outcome for the discriminative experiment is twofold. First, we expect that we will be able to see a difference between the R-R and R-F distributions for some if not all of the metrics. Second, we expect to also see differences in the distributions based on the applied transforms. Being able to quantitatively find differences in the R-R and R-F distributions for certain metrics supports the hypothesis that there exists metrics that allow for discrimination between real and generated data.

For the efficiency experiments, we expect to see increasing time for calculating the metrics as we increase the number of samples in the calculation. For sample efficiency, we expect to see an increase in the JSD score as we increase the number of samples involved in the calculation. This supports the hypothesis because efficiency is an important aspect of being able to practically use a possible metric for discrimination between real and generated data. Actual results and analysis are detailed in Chapter IV.

IV. Results and Analysis

Discriminative Results

In this section, we detail the results of the discriminative ability experiment. We present the results for all five of the transforms on both datasets. For each transform, a box-and-whisker plot of the Jensen-Shannon Distance (JSD) for each metric over the 10 runs is shown. We also present a table with the results ordered by decreasing mean JSD score and a histogram plot of the Real-Real (R-R) and Real-Fake (R-F) distributions for all of the metrics are presented. Note that the histogram plots all represent a single one of the ten runs. The histograms for each run look fairly similar so a single one was chosen to be a visual representative.

JSD values are bounded between 0 and 1. A 0 JSD indicates that the two distributions are identical and a JSD of 1 indicates that the two distributions are completely dissimilar. Based on visual inspection of the histogram plots in Figures 7, 9, 11, 13, 15, 17, 20, 23, 26 and 29, we noticed that for JSD scores between 0 and 0.5, little difference is noticeable in the R-R and R-F distributions, with both having similar shapes and lots of overlap. For JSD scores between 0.5 and 1.0 significant differences in distribution shape are noticeable with some overlap between the R-R and R-F distributions. A JSD score of 1 means that the R-R and R-F distributions are completely disjoint. To gauge the overall performance of each transform, we report how many of the metrics reach the aforementioned thresholds.

4.1 Network Events Data

Untransformed

Here we present the results of the discriminative experiment on the untransformed Network Events data. In Table 3 the results of the 10 runs are presented in order of decreasing JSD score. Entropy, Perplexity, and Cosine are the top three performing metrics and examining the range of the values we see that these numbers are fairly consistent. Additionally, we see that 3 of the 10 metrics in this space reach the first JSD threshold of 0.5. None of the metrics reach a JSD of 1.0 indicating that there is still some overlap between the distributions for the three metrics.

Cross referencing the results in Table 3 with Figure 7, we see that Entropy, Perplexity, and Cosine produce the most significant differences in the R-R and R-F distributions which corresponds with these three being the only ones to reach the previously defined thresholds.

Examining the boxplot of the JSD scores for each metric in Figure 8, we see the same results as in Table 3. Entropy, Perplexity, and Cosine are the only metrics which have a mean above 0.5 JSD. All of the metrics have fairly small ranges as evidenced by the small size of all the boxes and caps. Entropy has a very tight range with the edges of the box almost indistinguishable from the median and mean lines. Additionally we see for all of the metrics that the mean and median are very close together.

Square Root (SQRT) Transform

Examining the results from the SQRT transform on the Network Events data, we see similar orderings to the untransformed Network Events results with different magnitudes of JSD score. Table 4 shows the same top three performing metrics of Entropy, Perplexity, and Cosine. Examining Figure 9 we can verify that Entropy and Perplexity show the largest difference in the R-R and R-F distributions. Cosine and Mahalanobis also exhibit some differences but also have a large overlap in the two distributions, which corresponds with them being right on the threshold of 0.5 JSD.

Examining the boxplot of the JSD scores over the 10 runs of the experiment in

Table 3: Results of the discriminative experiment on the untransformed Network Events data. Only 3 of the 10 metrics reach the initial JSD threshold of 0.5. Corresponding with Figure 7, we see that Entropy, Perplexity, and Cosine are the only metrics with significant differences in the R-R and R-F distributions.

Metric	Mean	Min	Max	Range
Entropy	0.9342	0.9231	0.9428	0.0197
Perplexity	0.8501	0.8308	0.8651	0.0343
Cosine	0.6259	0.6039	0.6601	0.0562
Mahalanobis	0.4245	0.4058	0.4462	0.0404
Wasserstein	0.4097	0.3971	0.4285	0.0314
l_p: $p = r = 0.5$	0.346	0.3193	0.368	0.0487
l_p: $p = r = 0.75$	0.346	0.3166	0.3661	0.0495
Manhattan	0.3379	0.31	0.361	0.051
Euclidean	0.3246	0.3084	0.3342	0.0258
MMD	0.2338	0.2004	0.2572	0.0568

Figure 10, we see that several of the metrics have larger boxes than in Figure 8, particularly Entropy and Perplexity and this is confirmed by the larger ranges we see in Table 4. The Mahalanobis scores increase from the untransformed space, and it is the only metric to experience an increase.

Log Transform

The log transform results on the Network Events data show a similar overall degradation in the JSD scores to the SQRT transform. Table 5 shows the log transform results and in this case, Mahalanobis, l_p: $p = r = 0.5$, and Entropy are the top performers. None of the metrics reach the 0.5 JSD threshold. This indicates that the log transform produces very poor results for being able to discriminate between the R-R and R-F distributions. Figure 11 confirms this as there is very little visual difference in the two distributions for all of the metrics and lots of overlap is visible between them.

Table 4: JSD results from the SQRT transform on the Network Events data. The results have similar orderings to the untransformed results in Table 3 with lower overall scores. This time however, 4 of the 10 metrics reach the 0.5 JSD threshold. Cross referencing with Figure 9, we see that Entropy, Perplexity, Cosine, and Mahalanobis exhibit visible differences in the distributions.

Metric	Mean	Min	Max	Range
Entropy	0.7383	0.7047	0.7727	0.068
Perplexity	0.6454	0.6147	0.6762	0.0615
Cosine	0.5038	0.4849	0.5298	0.0449
Mahalanobis	0.502	0.4557	0.5246	0.0689
Wasserstein	0.3867	0.369	0.4133	0.0443
Manhattan	0.3356	0.3085	0.3566	0.0481
Euclidean	0.3342	0.3141	0.3485	0.0344
l_p: $p = r = 0.75$	0.322	0.2939	0.3389	0.045
l_p: $p = r = 0.5$	0.3045	0.2574	0.335	0.0776
MMD	0.2137	0.1831	0.2397	0.0566

The boxplot of the JSD scores in Figure 12 tells the same story as Table 5. Overall poor performance for this transform. With Mahalanobis as the top performer, we also see that it has a single outlier that is very low, pulling the mean outside of the box. We also see that the fractional l_p distances have larger ranges than on the other transforms.

Principal Components Analysis

The results of the Principal Component Analysis (PCA) transform on the Network Events data are displayed in Table 6 and a boxplot of the JSD scores is shown in Figure 14. Here we see that there is much better overall performance than from any of the other transforms with 8 of 11 metrics above 0.5 JSD. Additionally, we see that the fractional l_p and Wasserstein distances are the best performers.

The ranges for these metrics are also relatively small as well and this is confirmed by the small boxes for the high performing metric in Figure 14. Figure 13 verifies

Table 5: Log transform results on the Network Events data. Overall very poor results with none of the metrics reaching the 0.5 JSD threshold. Mahalanobis comes close and has a max value of 0.5108, indicating that during one of the runs it did reach the 0.5 threshold.

Metric	Mean	Min	Max	Range
Mahalanobis	0.4762	0.3099	0.5108	0.2009
l_p: $p = r = 0.5$	0.4061	0.3622	0.4522	0.09
Entropy	0.3702	0.3554	0.3816	0.0262
Euclidean	0.3333	0.3226	0.343	0.0204
Wasserstein	0.3324	0.3119	0.3534	0.0415
Cosine	0.3116	0.2981	0.3293	0.0312
Perplexity	0.3094	0.2928	0.3293	0.0365
l_p: $p = r = 0.75$	0.2996	0.2567	0.3315	0.0748
Manhattan	0.2873	0.2643	0.3044	0.0401
MMD	0.2326	0.1877	0.2537	0.066

these higher scores with clear differences in the metrics with JSD scores above 0.5.

It is important to note that with the PCA transform we are able to evaluate the Fréchet Inception Distance (FID). This is because the FID involves calculating a matrix square root, which can only be performed on a square matrix. The output of the PCA transform is an $n \times n$ matrix, where n is the number of features of the input sample. For all of the other transforms the input size is $1,000 \times n$ meaning that we cannot calculate the FID in those spaces.

Fast Fourier Transform

The Fast Fourier Transform (FFT) on the Network Events data is wholly ineffective. Table 7 shows the results for the metrics and none of the mean JSD scores are greater than 0.4. Examining Figure 15 we see that the R-R and R-F distributions are very similar with few noticeable differences.

Examining the boxplot in Figure 16 we confirm the results from Table 7. Interestingly, many of the metrics have very low outliers that pull down the mean scores.

Table 6: JSD results from the PCA transform on the Network Events data. 8 of 11 metrics reach the 0.5 JSD threshold. Cross referencing with Figure 13, we see significant differences in the R-R and R-F distributions for these 8 metrics. This indicates that this is a good transform to use for this dataset.

Metric	Mean	Min	Max	Range
l_p: $p = r = 0.5$	0.9594	0.9513	0.9669	0.0156
l_p: $p = r = 0.75$	0.9391	0.9308	0.946	0.0152
Wasserstein	0.9031	0.8923	0.9212	0.0289
Manhattan	0.8878	0.8732	0.8965	0.0233
Entropy	0.8369	0.8158	0.8649	0.0491
Perplexity	0.8049	0.7775	0.8313	0.0538
MMD	0.5662	0.5415	0.5873	0.0458
FID	0.5544	0.5301	0.5779	0.0478
Cosine	0.395	0.3707	0.4192	0.0485
Euclidean	0.3875	0.3605	0.4056	0.0451
Mahalanobis	0.3806	0.3506	0.4048	0.0542

However, these low outliers don't matter much because the entire box and whiskers for all metrics is below 0.5 JSD.

Table 7: JSD results from the FFT transform on Network Events data. Very poor results for all metrics with all metrics falling under 0.4 JSD indicating an inability to distinguish between the R-R and R-F distributions in this space.

Metric	Mean	Min	Max	Range
Euclidean	0.3994	0.3799	0.4436	0.0637
Cosine	0.3374	0.3193	0.3597	0.0404
Mahalanobis	0.3135	0.2274	0.3381	0.1107
Entropy	0.3052	0.2838	0.3255	0.0417
Perplexity	0.3049	0.2748	0.3238	0.049
Manhattan	0.2998	0.233	0.3179	0.0849
l_p: $p = r = 0.75$	0.286	0.1863	0.3113	0.125
Wasserstein	0.2807	0.2441	0.3048	0.0607
l_p: $p = r = 0.5$	0.2662	0.13	0.292	0.162
MMD	0.259	0.2329	0.3014	0.0685

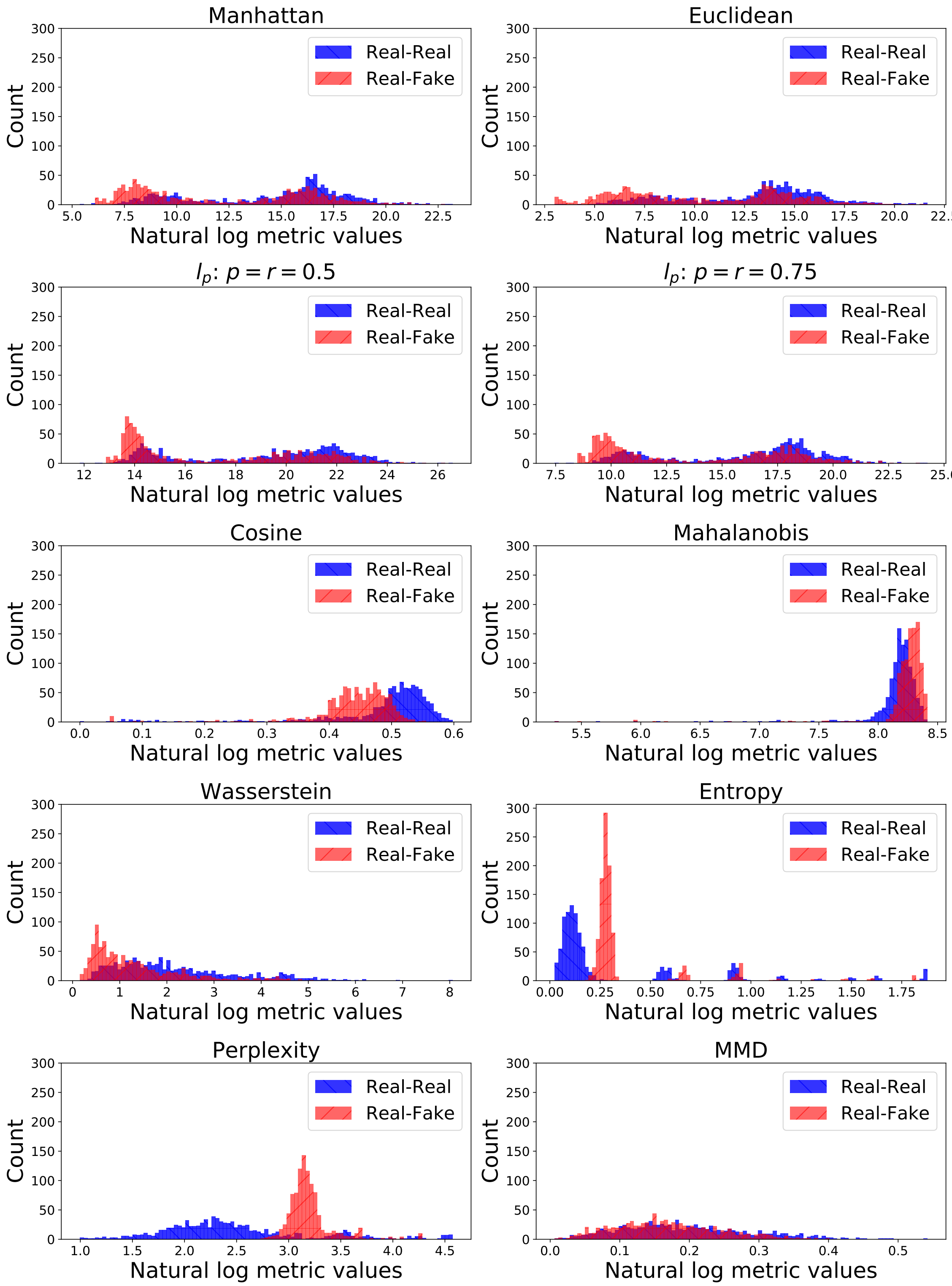

Figure 7: Discriminative results on untransformed Network Events data. Corresponding to the results in Table 3, only Cosine, Entropy, and Perplexity produce a noticeable difference in the R-R and R-F distributions with only Entropy and Perplexity being significantly different.

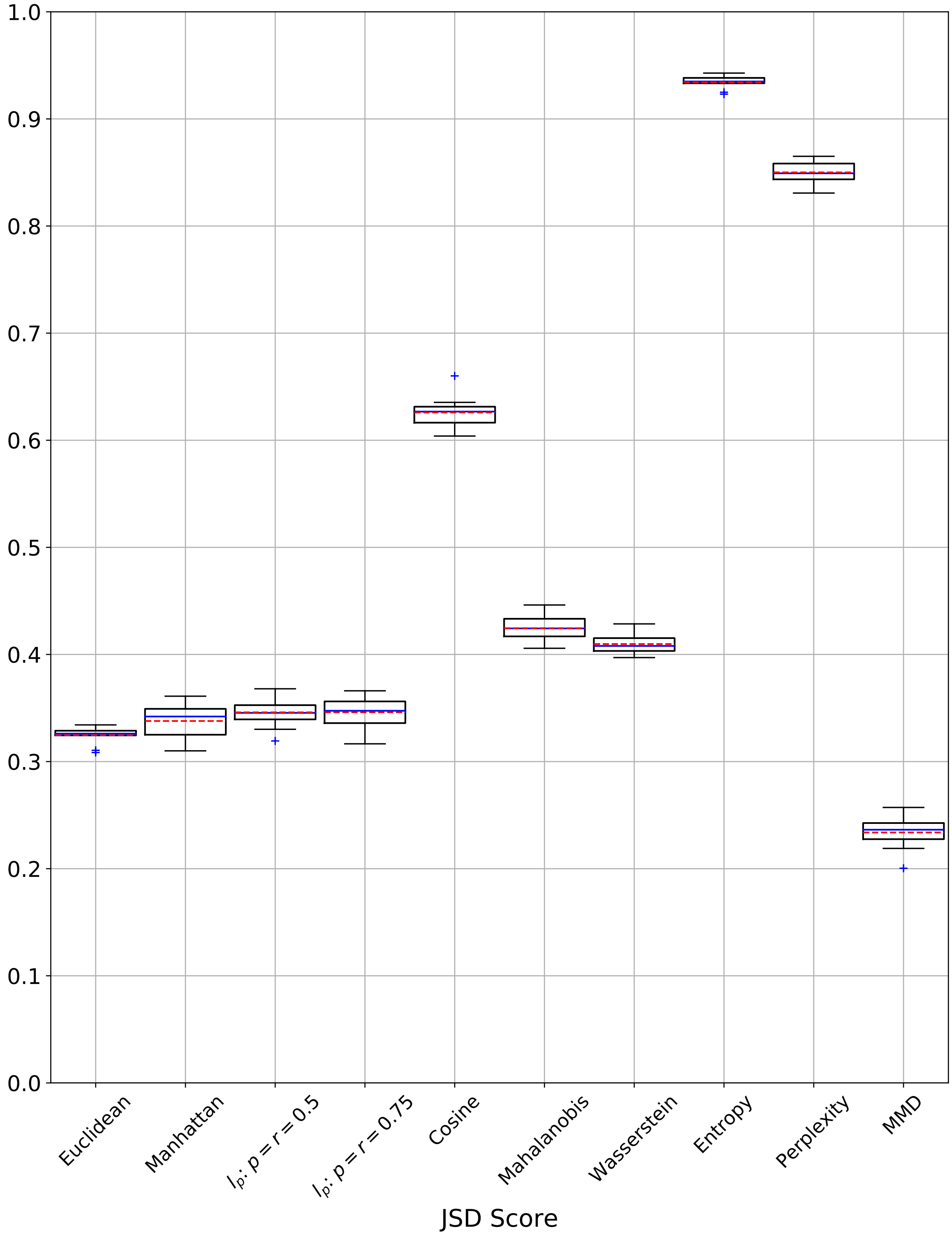

Figure 8: Untransformed Network Events boxplot of JSD scores for 10 runs of the experiment. 3 of the 10 metrics reach the 0.5 JSD threshold, indicating low overall performance for this transform.

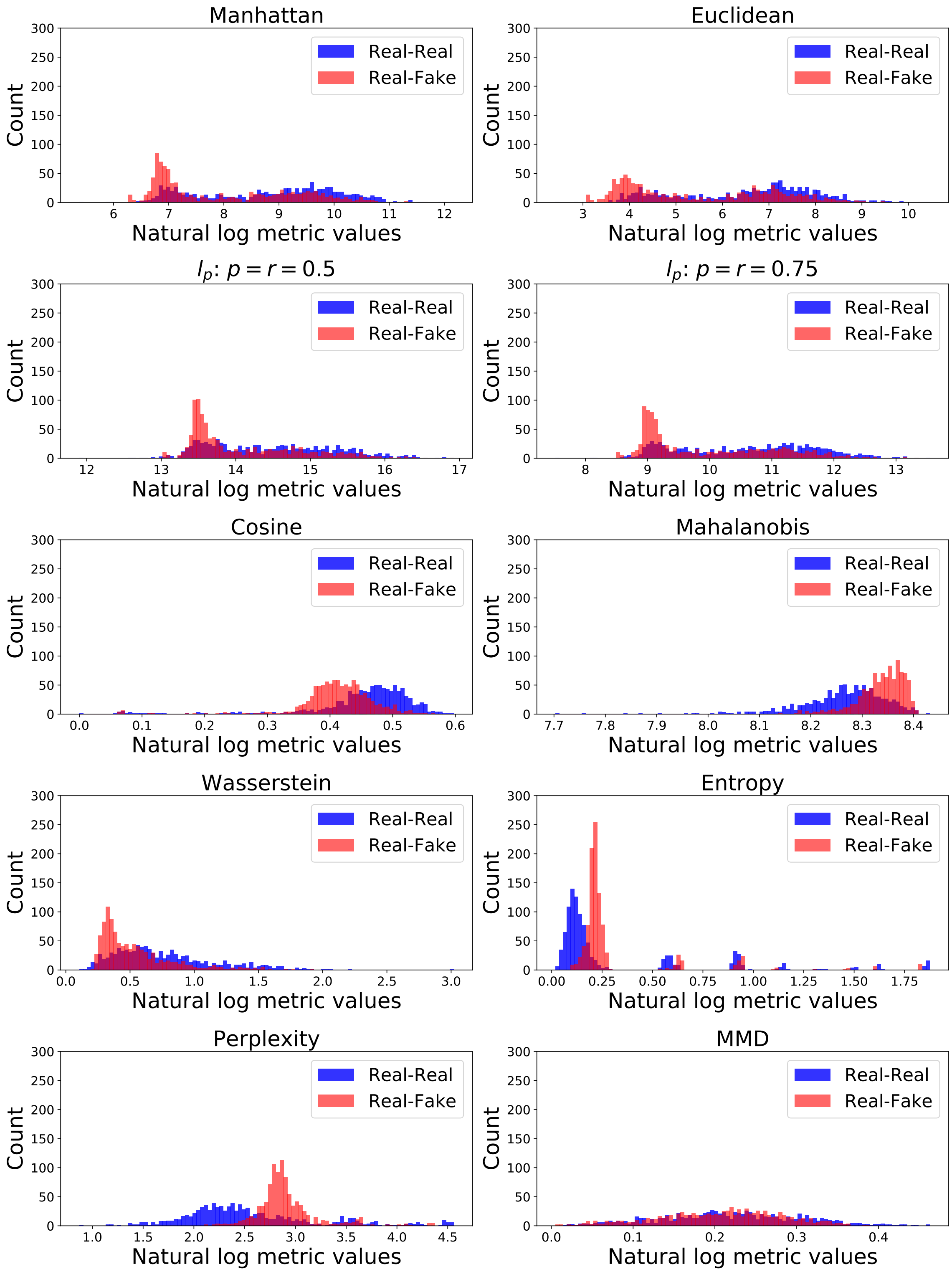

Figure 9: Discriminative results from SQRT transform on Network Events data. Corresponding to the results in Table 4 there is lots of overlap between the R-R and R-F distributions. Only Entropy and Perplexity are noticeably different, with some difference visible in Cosine and Mahalanobis.

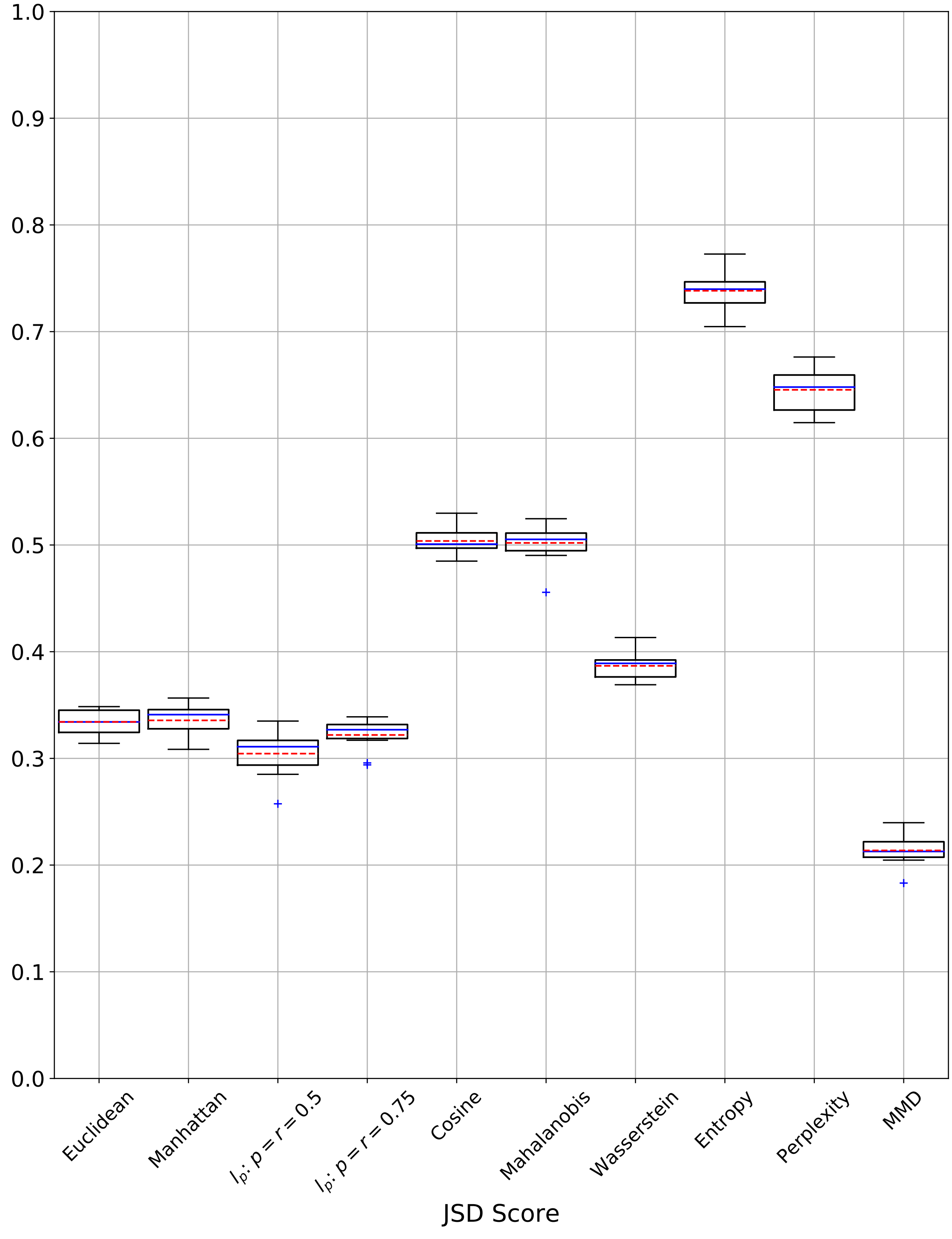

Figure 10: SQRT Network Events boxplot of JSD scores for 10 runs of the experiment. Here we get 4 of 10 metrics with a mean above 0.5 JSD. We also see larger boxes for Entropy and Perplexity compared to the boxes in Figure 8.

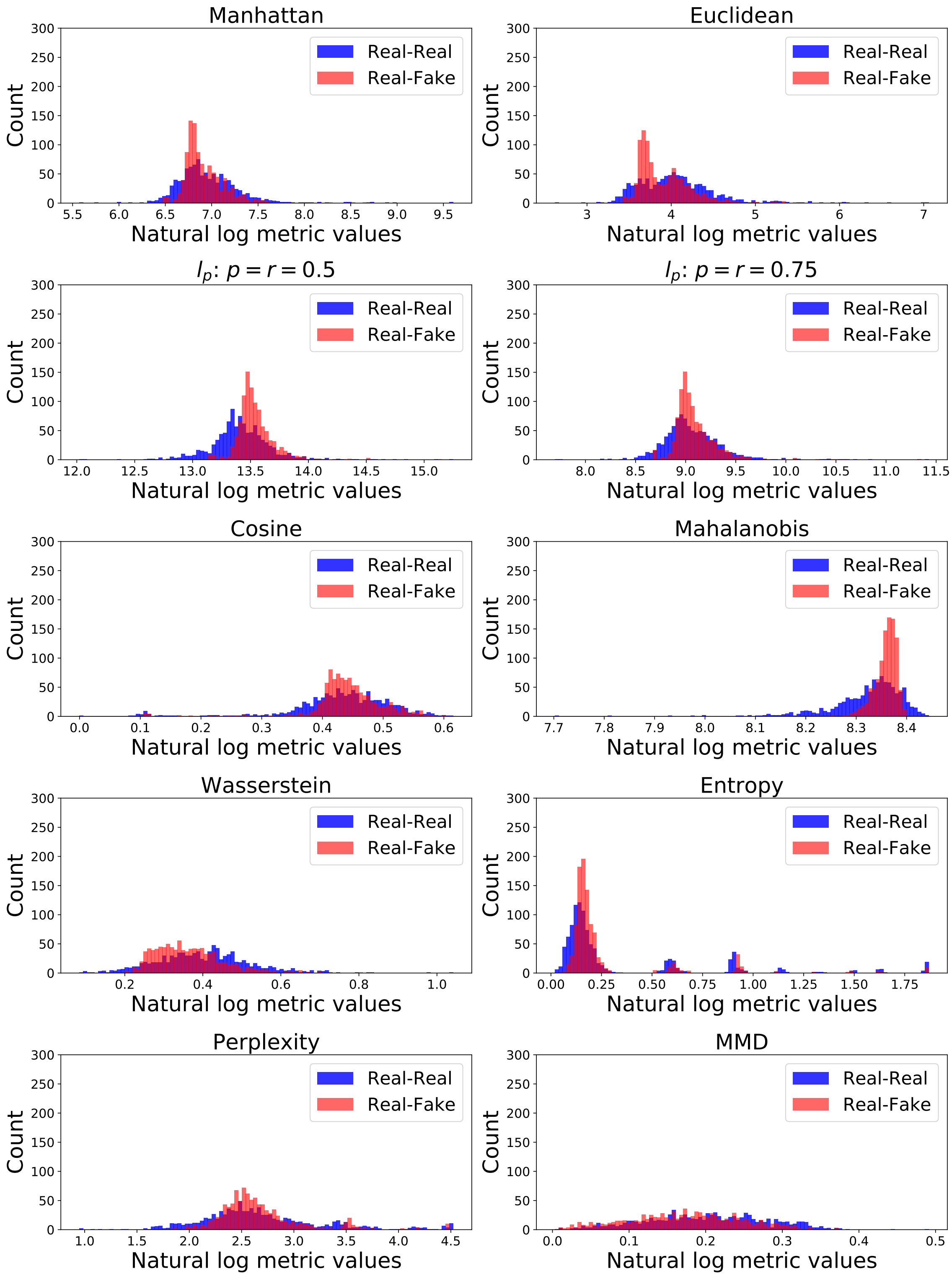

Figure 11: Discriminative results from log transform on Network Events data. There is very little difference in the R-R and R-F distributions for all of the metrics which corresponds to Table 5. This transform on this data produces very poor results.

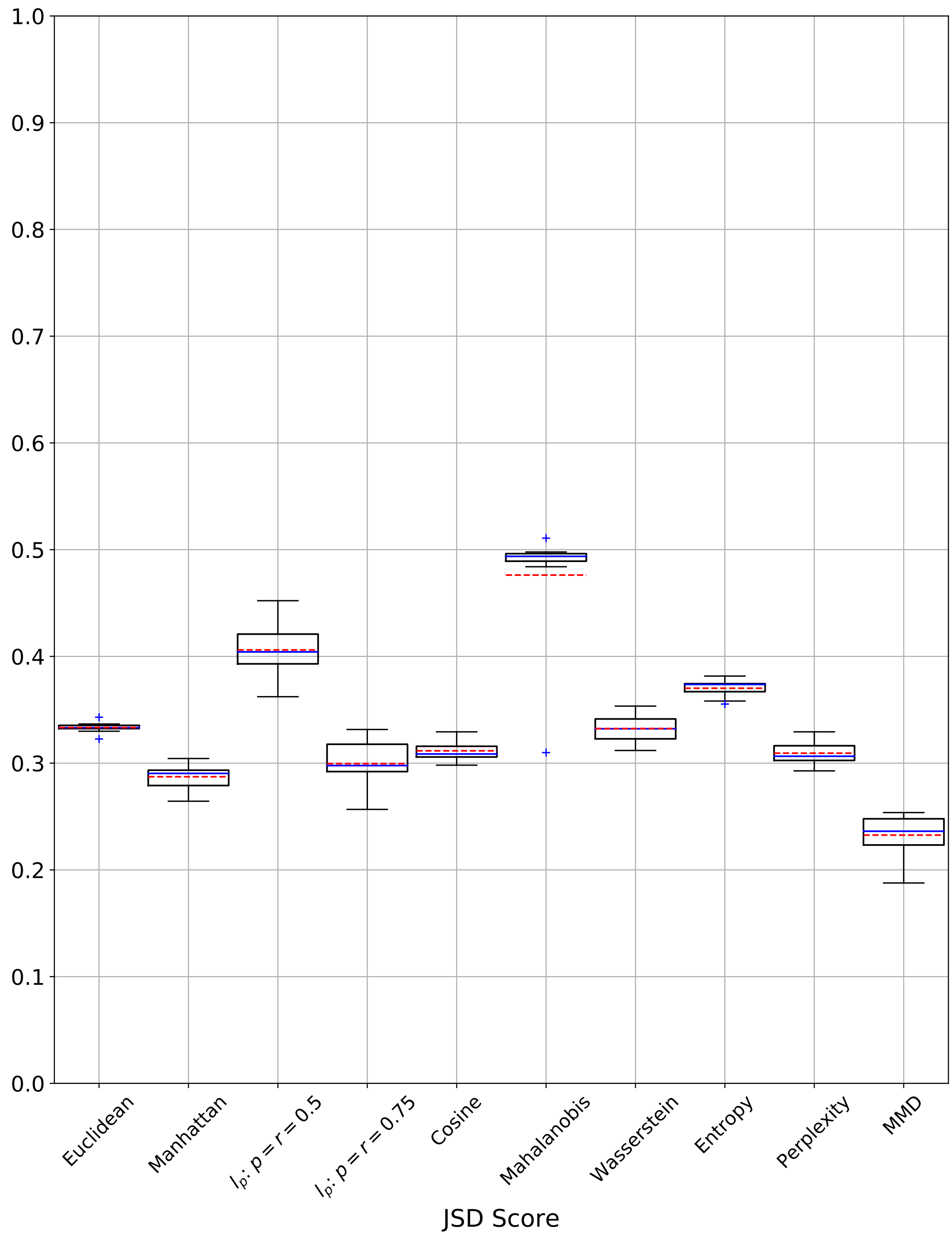

Figure 12: Log transform Network Events boxplot of JSD scores for 10 runs of the experiment. All metrics suffer a drop in JSD with none of the metrics having a mean JSD over 0.5, which is also reflected in Table 5.

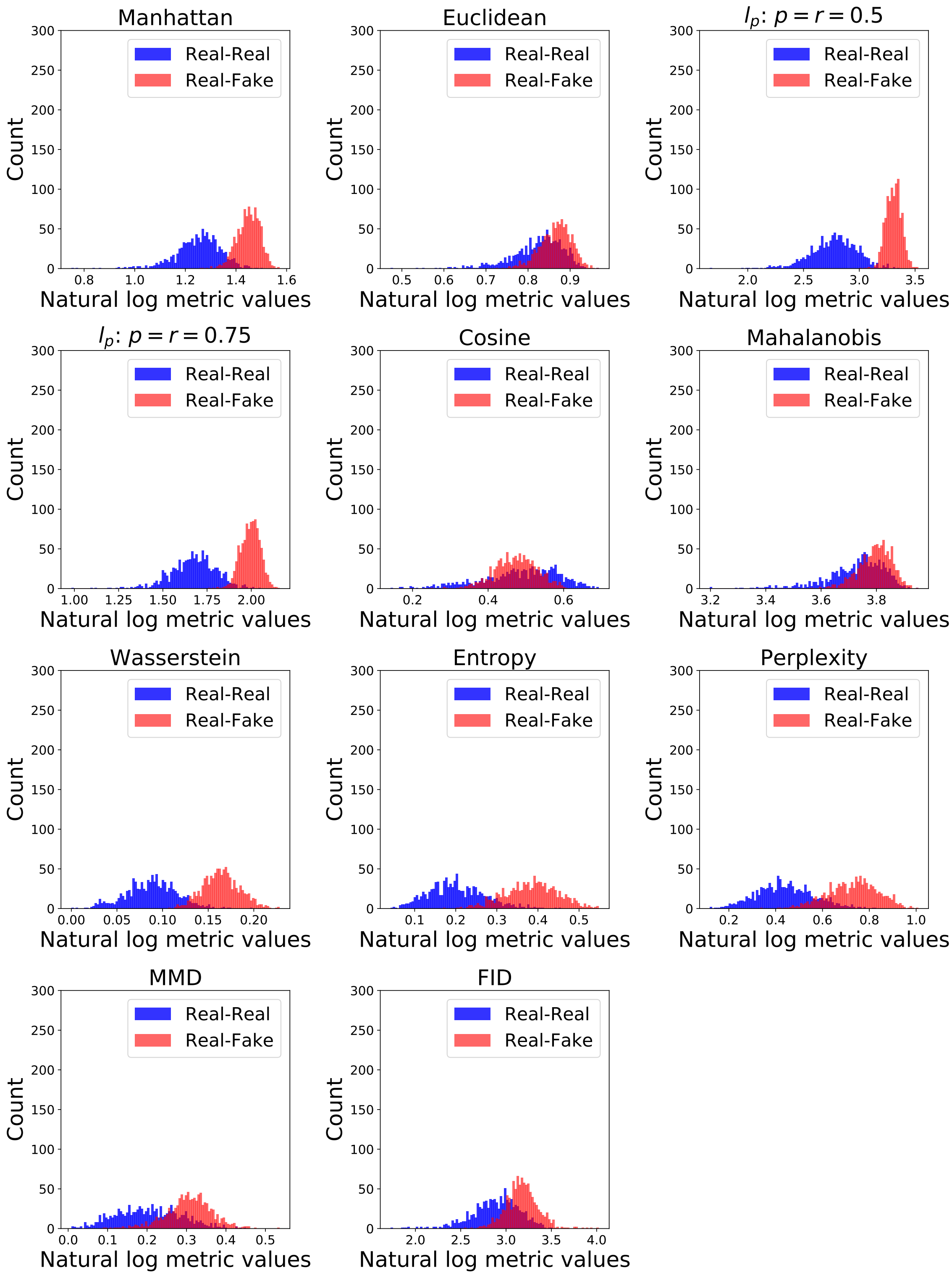

Figure 13: Discriminative results from PCA transform on Network Events data. As seen in Table 6, there are visible differences in the R-R and R-F distributions for most of the metrics. The most noticeable difference is between the fractional l_p distances and Wasserstein distances which have almost no overlap.

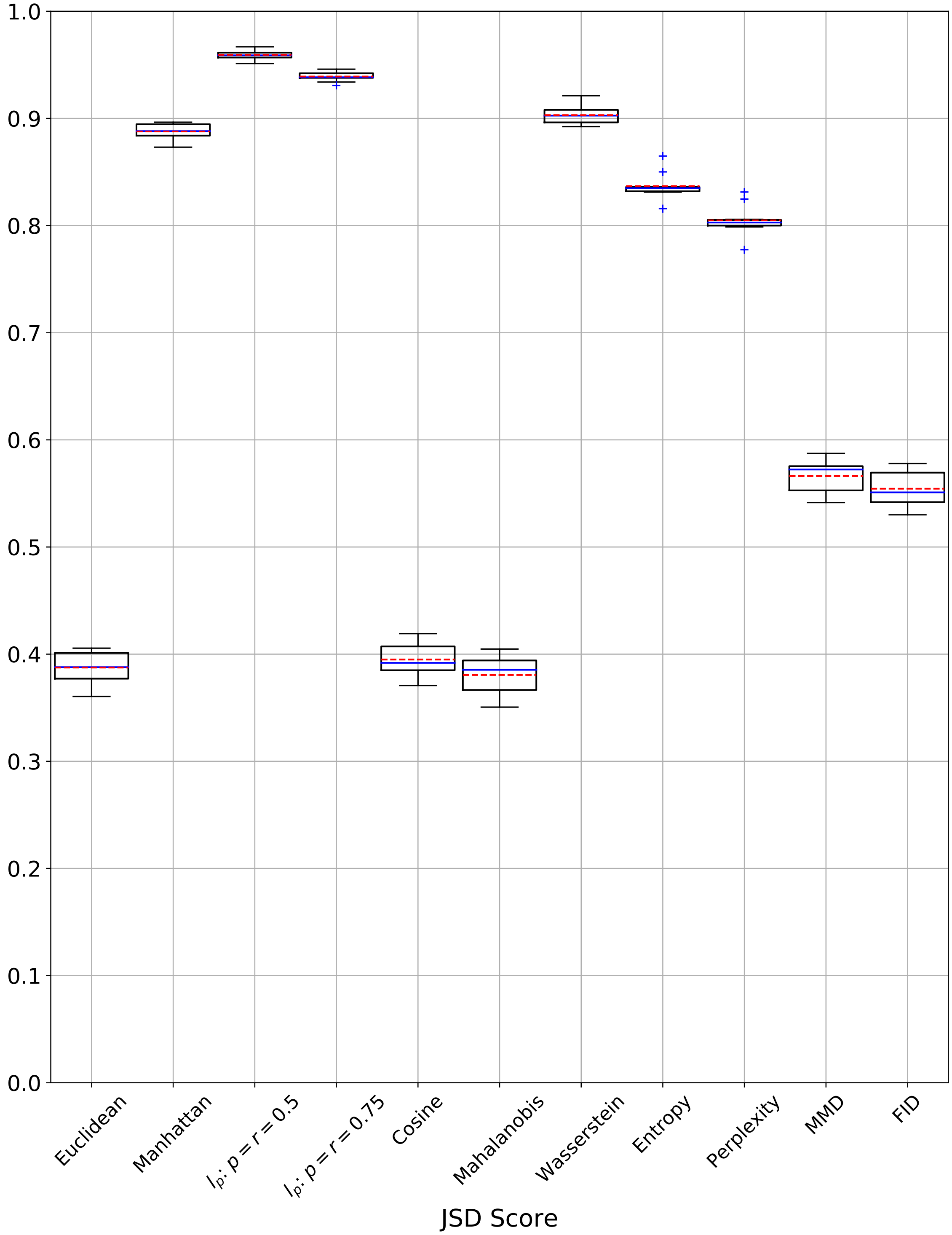

Figure 14: PCA transform Network Events boxplot of JSD scores for 10 runs of the experiment. Much better performance with 8 of 11 metrics reaching the 0.5 JSD threshold. The highest performing metrics also have very small boxes and whiskers indicating good repeatability for these scores.

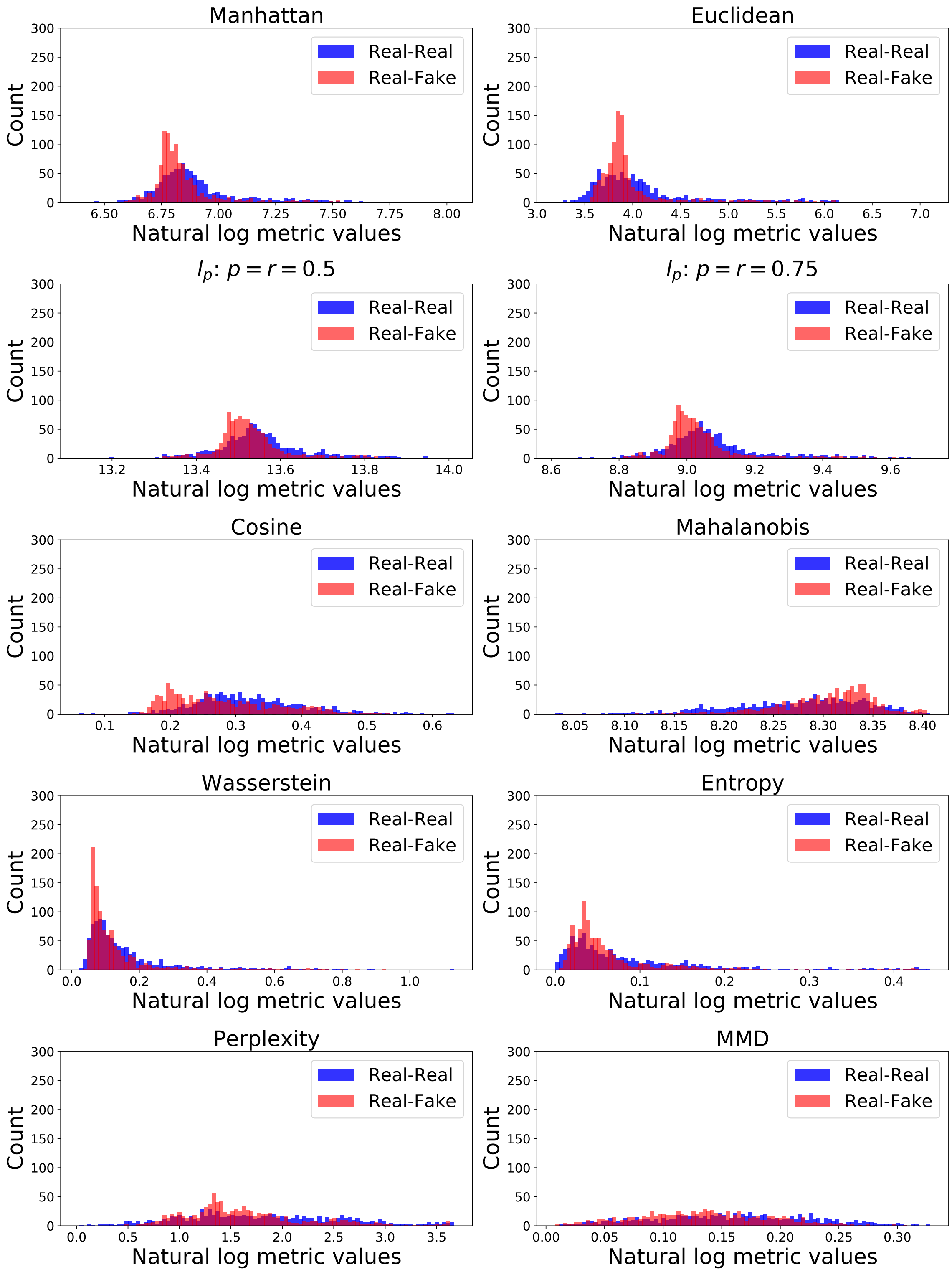

Figure 15: Discriminative results from FFT transform on Network Events data. Examining the histograms we see that there is lots of overlap between the R-R and R-F distributions for all metrics. This corresponds to the extremely low JSD scores (< 0.4) seen in Table 7.

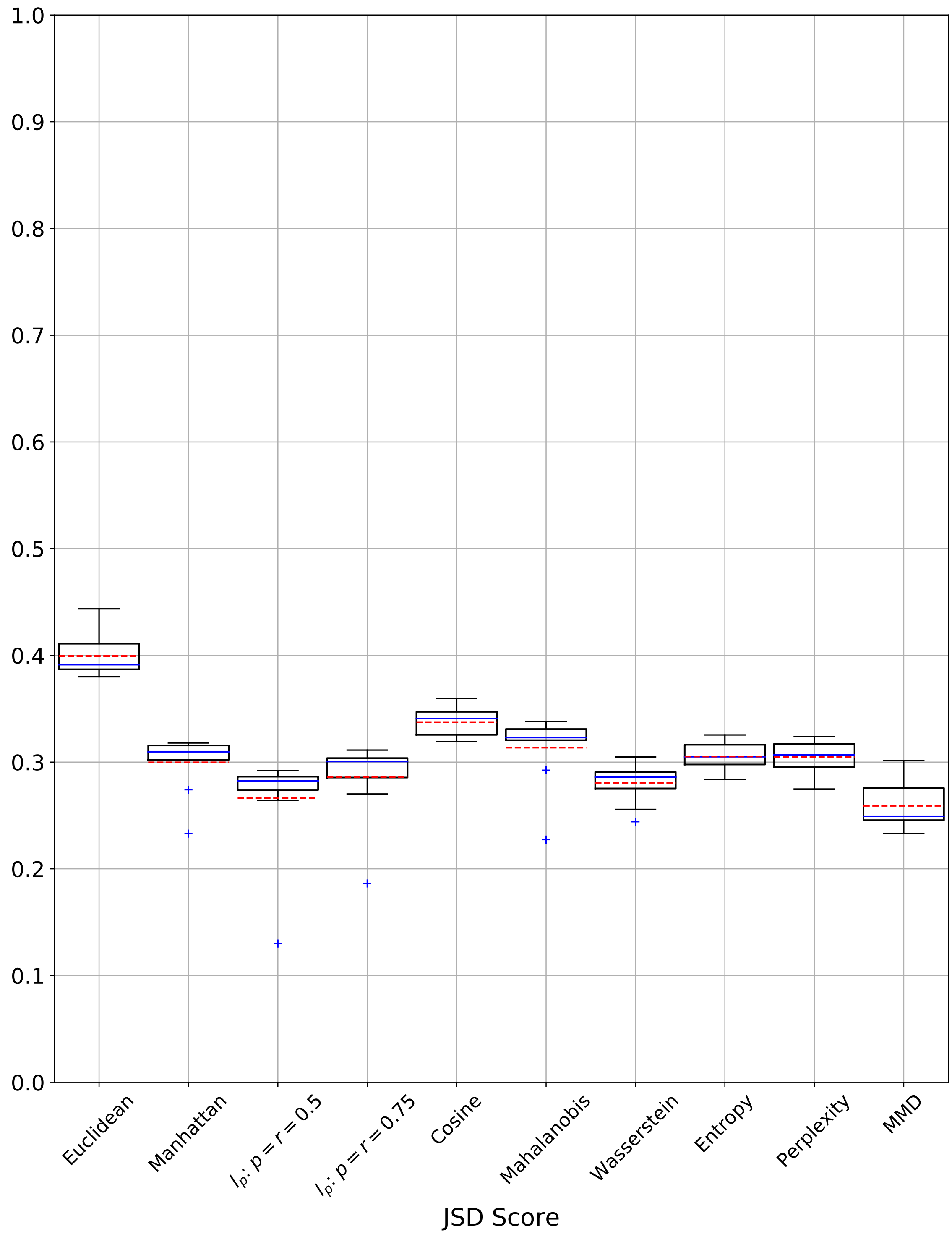

Figure 16: FFT transform network events boxplot of JSD scores for 10 runs of the experiment. Overall results for the FFT transform are very poor with all metrics failing to reach 0.5 JSD. May metrics have low outliers indicated by the blue pluses, however they do not greatly affect the position of the boxes.

4.2 Host Events Data

For the host events data, we experimented with two different methods of generating the fake data. First, like the Network Events data, samples were generated from a Uniform distribution between the global minimum and maximum between each feature. For reference, we say that these samples are from the uniform distribution. Second, we generated a set of fake data from a Normal distribution based on the global mean and global variance for each feature. For reference we say that these samples are from the normal distribution.

For each transform, we show the mean JSD results in tables from both the uniform and normal distributions. We also display box-and-whisker plots for the 10 runs of JSD scores for both the uniform and normal data to correspond with the data displayed in the tables. However, due to the overall similarity, we only report the histogram figures based on the uniform data. Additionally, due to the overall similarity between the results on the uniform and normal data, we only use the uniform data for the Efficiency experiments described in Section 4.2.

Untransformed

Table 8 contains the results of the discriminative experiment on the Host Events data with the generated sample from the uniform distribution while Table 9 contains the JSD results for generated samples from the normal distribution. There is a large difference is the overall JSD scores from the Network Events untransformed data to the Host Events untransformed data, with the Host Events JSD scores being higher across the board. Wasserstein, l_p: $p = r = 0.5$, and Mahalanobis distance exhibit the best performance. 8 of the 10 metrics reach the 0.5 JSD threshold.

Additionally, Wasserstein distance reaches a JSD of 1.0 for all 10 runs as indicated by the min and max being 1 as well. These results show an ability to distinguish between the R-R and R-F distributions which can be seen in Figure 17. For this dataset, not performing any transform on the data produces surprisingly good results.

Examining the boxplots in Figure 18 and Figure 19 gives us some insights. For the uniform and normal data,the Wasserstein distance is also confirmed to have a JSD of 1.0 for all 10 runs because the entire box, whiskers, and mean and median line are all on 1.0. Mahalanobis comes close, however, it appears that an outlier just above 0.8 skews the mean lower than the rest of the runs for both the normal and uniform data. We also see the drop in JSD for Perplexity as the JSD box shifts from the 0.8 range to just above 0.5.

Despite the better results on the Host Events data, we do see a larger range over all of the runs for many of the metrics, particularly Mahalanobis distance at 0.177. However, examining the minimum and maximum values we see that the JSD scores are still rather high, so the JSD score for this metric is still a meaningful value.

Similarly, with the JSD results from the normal data in Table 9, we see that 8 of the 10 metrics reach the 0.5 JSD threshold with Wasserstein again reaching a JSD of 1.0 for all 10 runs. The main difference from the uniform to normal results is the drop in JSD for the Perplexity metric from 0.7999 on the uniform data to 0.5338 on the normal data.

SQRT Transform

The SQRT transformed Host Events data also shows much better results overall than the Network Events data. Examining Table 10 and Table 11 we see that 9 of the 10 metrics reach the 0.5 JSD. The Wasserstein, l_p: $p = r = 0.5$, and Mahalanobis metrics come in as the top three performing metrics again, with Wasserstein maintaining a JSD of 1.0 for all 10 runs, indicating that the distributions are completely separate. This is an interesting development on this data since the JSD scores don't

Table 8: JSD results on uniform untransformed Host Events data. 8 of 10 metrics reach the 0.5 JSD threshold with Wasserstein also reaching a JSD of 1 for all 10 runs as indicated by the Min and Max being 1.

Metric	Mean	Min	Max	Range
Wasserstein	1.0	1.0	1.0	0.0
Mahalanobis	0.9815	0.823	1.0	0.177
Cosine	0.964	0.9483	0.9751	0.0268
l_p: $p = r = 0.5$	0.9331	0.9236	0.9435	0.0199
Perplexity	0.7999	0.7813	0.8117	0.0304
l_p: $p = r = 0.75$	0.7649	0.744	0.7818	0.0378
Entropy	0.7224	0.7077	0.7347	0.027
Manhattan	0.5836	0.5606	0.6052	0.0446
Euclidean	0.2833	0.2746	0.2945	0.0199
MMD	0.1842	0.1292	0.2069	0.0777

Table 9: JSD results on normal untransformed Host Events data. 8 of 10 metrics reach the 0.5 JSD threshold with Wasserstein reaching a JSD of 1.0 for all 10 runs. Main noticeable difference in normal and uniform results is the JSD for Perplexity dropping from 0.7999 to 0.5338.

Metric	Mean	Min	Max	Range
Wasserstein	1.0	1.0	1.0	0.0
Mahalanobis	0.9799	0.815	1.0	0.185
Cosine	0.9616	0.9449	0.9743	0.0294
l_p: $p = r = 0.5$	0.946	0.9328	0.9587	0.0259
l_p: $p = r = 0.75$	0.8337	0.8161	0.8458	0.0297
Entropy	0.7092	0.6911	0.7374	0.0463
Manhattan	0.688	0.6703	0.7082	0.0379
Perplexity	0.5338	0.5101	0.5474	0.0373
Euclidean	0.4177	0.4015	0.4332	0.0317
MMD	0.1838	0.1336	0.2149	0.0813

drop on the Host Events data like they do on the Network Events data.

The boxplots of the JSD scores in Figure 21 and Figure 22 confirm the results we see in Table 10 and Table 11. For both uniform and normal, Wasserstein has a JSD of 1.0 for all 10 runs. Mahalanobis also contains the single outlier skewing the mean down. The JSD for Perplexity drops again, this time from around 0.6 down to just under 0.5 in the normal.

Examining the differences between the uniform results in Table 10 and normal results in Table 9 we again see that Perplexity drops, this time from 0.6041 to 0.471.

Table 10: JSD results from SQRT transform on Host Events uniform data. 9 of the 10 metrics reach the 0.5 JSD threshold. Overall this transform produces good results on this dataset, in contrast to the results of this transform on the Network Events dataset from Table 4.

Metric	Mean	Min	Max	Range
Wasserstein	1.0	1.0	1.0	0.0
Mahalanobis	0.9827	0.8293	1.0	0.1707
Cosine	0.9596	0.9438	0.9717	0.0279
l_p: $p = r = 0.5$	0.9483	0.9348	0.9587	0.0239
l_p: $p = r = 0.75$	0.8606	0.8468	0.8741	0.0273
Manhattan	0.764	0.7354	0.7836	0.0482
Perplexity	0.6041	0.5853	0.6198	0.0345
Entropy	0.5948	0.5813	0.6156	0.0343
Euclidean	0.5565	0.5451	0.5739	0.0288
MMD	0.1778	0.1336	0.2009	0.0673

Table 11: JSD results from SQRT transform on Host Events normal data. Similar results to Table 10 with 9 of the 10 metrics reaching the 0.5 JSD threshold. The only significant difference from the uniform data is the decrease in Perplexity JSD from 0.6041 to 0.471.

Metric	Mean	Min	Max	Range
Wasserstein	1.0	1.0	1.0	0.0
Mahalanobis	0.9814	0.8221	1.0	0.1779
Cosine	0.9571	0.94	0.97	0.03
l_p: $p = r = 0.5$	0.9568	0.9452	0.9639	0.0187
l_p: $p = r = 0.75$	0.8863	0.8761	0.9082	0.0321
Manhattan	0.809	0.7865	0.8221	0.0356
Euclidean	0.6425	0.6186	0.6558	0.0372
Entropy	0.6144	0.5986	0.6295	0.0309
Perplexity	0.471	0.4443	0.4875	0.0432
MMD	0.1896	0.1535	0.218	0.0645

Log Transform

The results for log transformed Host Events data are shown in Table 12 and Table 13. On the uniform data in Table 12 8 of the 10 metrics reach the 0.5 JSD threshold. On the normal data in Table 13, 8 of 10 metrics reach the 0.5 JSD threshold.

Wasserstein outperforms the other metrics again with all 10 runs having a JSD of 1.0. Table 12 shows that the Wasserstein, l_p: $p = r = 0.5$, and Mahalanobis metrics all once again are in the top three and exhibit similar JSD scores to the untransformed and SQRT transform results.

Figure 23 confirms the results we see in Table 12. There are clear differences in the R-R and R-F distributions for all of the metrics except for Euclidean distance and Maximum Mean Discrepancy (MMD). Examining the differences from the uniform samples in Table 12 and the normal samples in Table 13, we see that again the only significant difference is the drop in Perplexity JSD from 0.7629 in Table 12 to 0.5569 in Table 13.

The boxplots in Figure 24 and Figure 25 show similar results to the untransformed and SQRT transform. Once again in both cases, the Wasserstein JSD is 1.0 for all 10 runs and Mahalanobis contains the single outlier which brings down the mean. Like other transforms, we also see the Perplexity JSD drop. This time it drops from around 0.75 in the uniform data to around 0.55 in the normal data.

Principal Components Analysis

Examining the results of the PCA transform on the Host Events data in Table 14 and Table 15 we see that all 11 of the metrics exceed the 0.5 JSD threshold with 6 of the 11 also getting a 1.0 JSD score for all 10 runs. It is important to note that with the PCA transform, all of the metrics have higher mean scores compared to some of

Table 12: JSD results from log transform on uniform Host Events data. 8 of the 10 metrics reach the 0.5 JSD threshold, indicating once again that this transform works well on this data.

Metric	Mean	Min	Max	Range
Wasserstein	1.0	1.0	1.0	0.0
Mahalanobis	0.9817	0.8239	1.0	0.1761
Cosine	0.9635	0.9478	0.9747	0.0269
l_p: $p = r = 0.5$	0.9385	0.9249	0.9475	0.0226
l_p: $p = r = 0.75$	0.7922	0.7731	0.8063	0.0332
Perplexity	0.7629	0.7498	0.776	0.0262
Entropy	0.6496	0.6316	0.665	0.0334
Manhattan	0.6277	0.6062	0.6523	0.0461
Euclidean	0.3377	0.318	0.3512	0.0332
MMD	0.1783	0.1444	0.201	0.0566

Table 13: JSD results from log transform on normal Host Events data. 8 of the 10 metrics reach the 0.5 JSD threshold. Perplexity JSD drops again from the uniform to normal, this time from 0.7629 to 0.5569.

Metric	Mean	Min	Max	Range
Wasserstein	1.0	1.0	1.0	0.0
Mahalanobis	0.9806	0.8177	1.0	0.1823
Cosine	0.9626	0.9477	0.9768	0.0291
l_p: $p = r = 0.5$	0.9475	0.9317	0.9567	0.025
l_p: $p = r = 0.75$	0.8442	0.8286	0.8587	0.0301
Entropy	0.7179	0.7007	0.7332	0.0325
Manhattan	0.7107	0.6925	0.7355	0.043
Perplexity	0.5569	0.5353	0.5739	0.0386
Euclidean	0.4494	0.4319	0.4631	0.0312
MMD	0.1838	0.1296	0.2007	0.0711

the other transforms which have low scores for Euclidean distance and MMD.

Comparing the differences between the uniform results in Table 14 and the normal results in Table 15 we see that the Perplexity JSD does not experience the large decrease of the other transforms and maintains a JSD score of 1.0 for all 10 runs in both cases. The only difference in the relative rankings is that MMD drops below

Mahalanobis on the normal samples, but it is not a large drop.

Examining the boxplots in Figure 27 and Figure 28 confirms the results from Table 14 and Table 15. This time, in both cases, six of the metrics still have a JSD above 1.0 for all 10 runs. Additionally, for all other metrics we see boxes and whiskers with no outliers in the uniform data, unlike the other transforms. This time, Perplexity JSD does not drop as it stays at 1.0 JSD in both instances. For the first time, the MMD also makes it above the 0.5 threshold in both the uniform and normal instances.

Table 14: JSD results from PCA transform on uniform Host Events data. All 11 metrics exceed the 0.5 JSD threshold with 6 of the 11 getting a 1.0 JSD, indicating complete dissimilarity between the R-R and R-F distributions for these metrics. This indicates that the PCA transform produces good results much as it did with the Network Events data in Table 6.

Metric	Mean	Min	Max	Range
Manhattan	1.0	1.0	1.0	0.0
l_p: $p = r = 0.5$	1.0	1.0	1.0	0.0
l_p: $p = r = 0.75$	1.0	1.0	1.0	0.0
Wasserstein	1.0	1.0	1.0	0.0
Entropy	1.0	1.0	1.0	0.0
Perplexity	1.0	1.0	1.0	0.0
FID	0.8976	0.888	0.9152	0.0272
Cosine	0.7353	0.7214	0.7457	0.0243
MMD	0.6833	0.6653	0.7062	0.0409
Mahalanobis	0.6686	0.6499	0.6984	0.0485
Euclidean	0.6446	0.6274	0.6705	0.0431

Fast Fourier Transform

The FFT results on the Host Events data are much more successful than on the Network Events data. Entropy, Cosine, and Perplexity come out as the top performers and are shown in Table 16. 7 of the 10 metrics meet the 0.5 JSD threshold with 2 of

Table 15: JSD results from PCA transform on normal Host Events data. All 11 metrics exceed the 0.5 JSD threshold with 6 of the 11 getting a 1.0 JSD, indicating complete dissimilarity between the R-R and R-F distributions for these metrics. In contrast to the other transforms, this time the JSD for Perplexity does not drop between the uniform and normal samples.

Metric	Mean	Min	Max	Range
Manhattan	1.0	1.0	1.0	0.0
l_p: $p = r = 0.5$	1.0	1.0	1.0	0.0
l_p: $p = r = 0.75$	1.0	1.0	1.0	0.0
Wasserstein	1.0	1.0	1.0	0.0
Entropy	1.0	1.0	1.0	0.0
Perplexity	1.0	1.0	1.0	0.0
FID	0.8534	0.8369	0.8739	0.037
Cosine	0.7525	0.7396	0.7663	0.0267
Mahalanobis	0.6752	0.6531	0.7004	0.0473
MMD	0.6666	0.648	0.6959	0.0479
Euclidean	0.6549	0.6369	0.6793	0.0424

the 7 getting a 1.0 JSD. Even though it performs well, Mahalanobis distance has a large range of 0.4662, indicating it might not be very stable with the FFT data.

Examining the differences in the uniform data in Table 16 and normal data in Table 17 we see some different things happening with the FFT transform. For the other spaces, there were not many differences in the ordering of the metrics between the uniform and normal data. However, with the FFT transform we see a difference in the ordering of the metrics and a difference in the overall JSD values between the uniform and normal data.

In the uniform data, the top four metrics are Entropy, Perplexity, Cosine, and Mahalanobis. With the normal data, the top four metrics are Mahalanobis , Entropy, Perplexity, and Wasserstein. The overall JSD decreases from the uniform to normal as well, with the Cosine JSD going from 0.9968 to 0.7243. Additionally, Entropy and Perplexity both drop from 1.0 JSD for all 10 runs on uniform data down to around

Table 16: JSD results from FFT transform on uniform Host Events data. 7 of the 10 metrics meet the 0.5 JSD threshold with 2 of the 7 getting a 1.0 JSD, indicating complete dissimilarity between the distributions.

Metric	Mean	Min	Max	Range
Entropy	1.0	1.0	1.0	0.0
Perplexity	1.0	1.0	1.0	0.0
Cosine	0.9968	0.9939	0.9995	0.0056
Mahalanobis	0.9498	0.5316	0.9978	0.4662
Wasserstein	0.895	0.8802	0.9084	0.0282
Euclidean	0.8846	0.8602	0.8985	0.0383
MMD	0.5763	0.5528	0.5983	0.0455
l_p: $p = r = 0.75$	0.2486	0.1085	0.2951	0.1866
Manhattan	0.2435	0.0839	0.2821	0.1982
l_p: $p = r = 0.5$	0.2229	0.0224	0.267	0.2446

Table 17: JSD results from FFT transform on Host Events data. 5 of the 10 metrics meet the 0.5 JSD threshold with. The FFT transform is the only one in which there are significant differences in the order of the metrics and overall JSD between the uniform data (Table 16) and normal data.

Metric	Mean	Min	Max	Range
Mahalanobis	0.9457	0.5359	0.994	0.4581
Entropy	0.8568	0.8436	0.8759	0.0323
Perplexity	0.843	0.8289	0.8597	0.0308
Wasserstein	0.7456	0.7282	0.7612	0.033
Cosine	0.7243	0.7152	0.7471	0.0319
MMD	0.5274	0.4995	0.5579	0.0584
Manhattan	0.4905	0.3958	0.5273	0.1315
l_p: $p = r = 0.75$	0.429	0.3281	0.4696	0.1415
Euclidean	0.4231	0.3815	0.4517	0.0702
l_p: $p = r = 0.5$	0.2872	0.0224	0.3396	0.3172

0.85 on normal data. The largest drop however, is the Euclidean JSD. With the uniform data, the JSD is 0.8846 and with the normal data it drops to 0.4231 which doesn't happen in any of the other transforms.

Examining the boxplots in Figure 30 and Figure 31, we confirm the major differences in the uniform and normal data. In the uniform data, 7 of the 10 metrics are above the 0.5 JSD threshold while in the normal data, only 5 of 10 are above the threshold. In the uniform five of the metrics are around the 0.9 or above range while in the normal data, only one of the metrics is above 0.9.

There are also very low outliers for many of the metrics in both instances, particularly for Mahalanobis at about 0.5 while the other runs are all at 1.0. Manhattan distance experiences a drastic decrease from the uniform to normal, dropping from just under 0.9 to just above 0.4. MMD also performs much better in this transform, making it above 0.5 JSD in the uniform and normal instances.

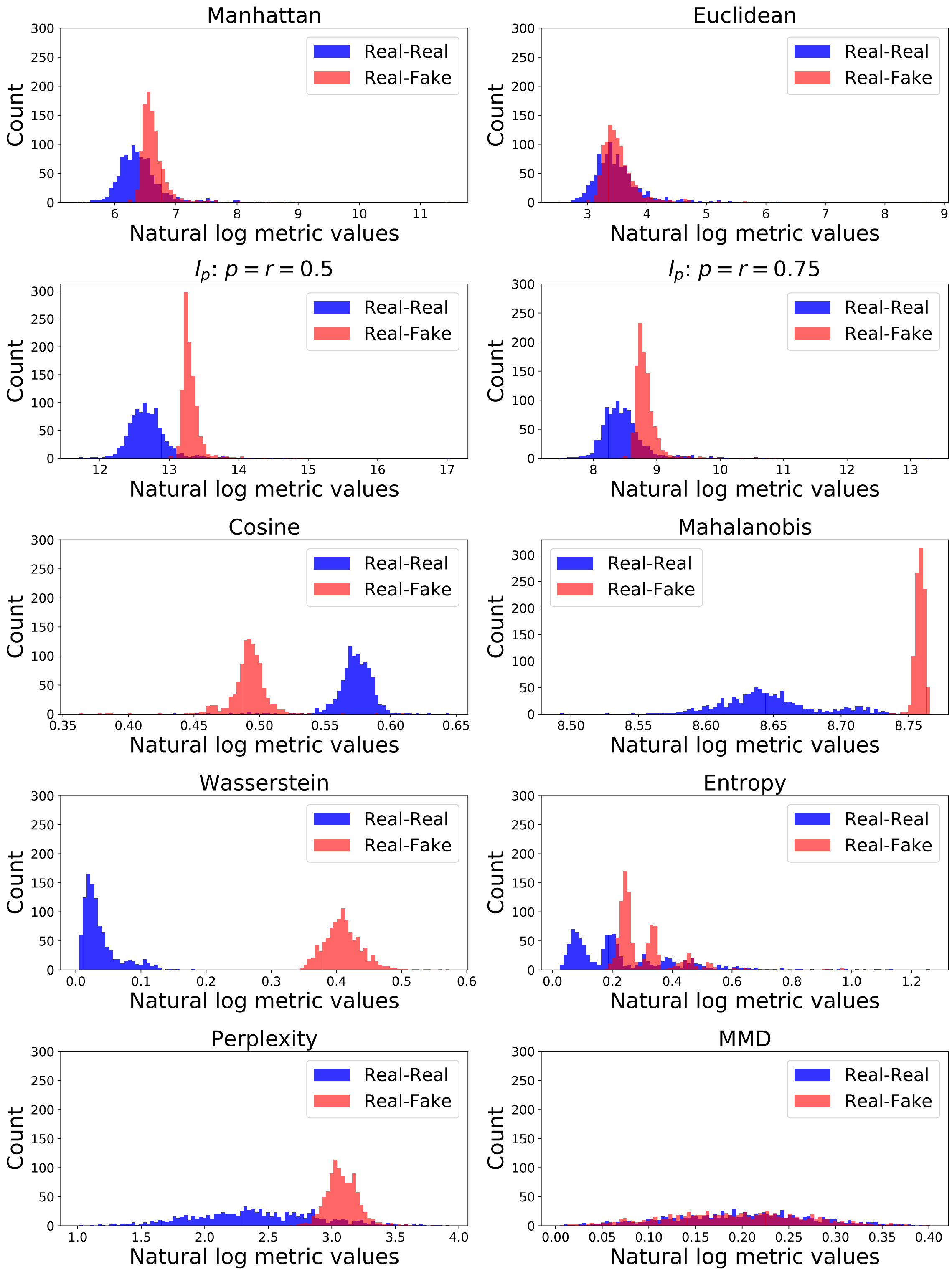

Figure 17: Discriminative results from untransformed Host Events data. Differences in the R-R and R-F distributions are clearly visible for most of the metrics. This corresponds with most of the metrics having higher JSD scores in Table 8.

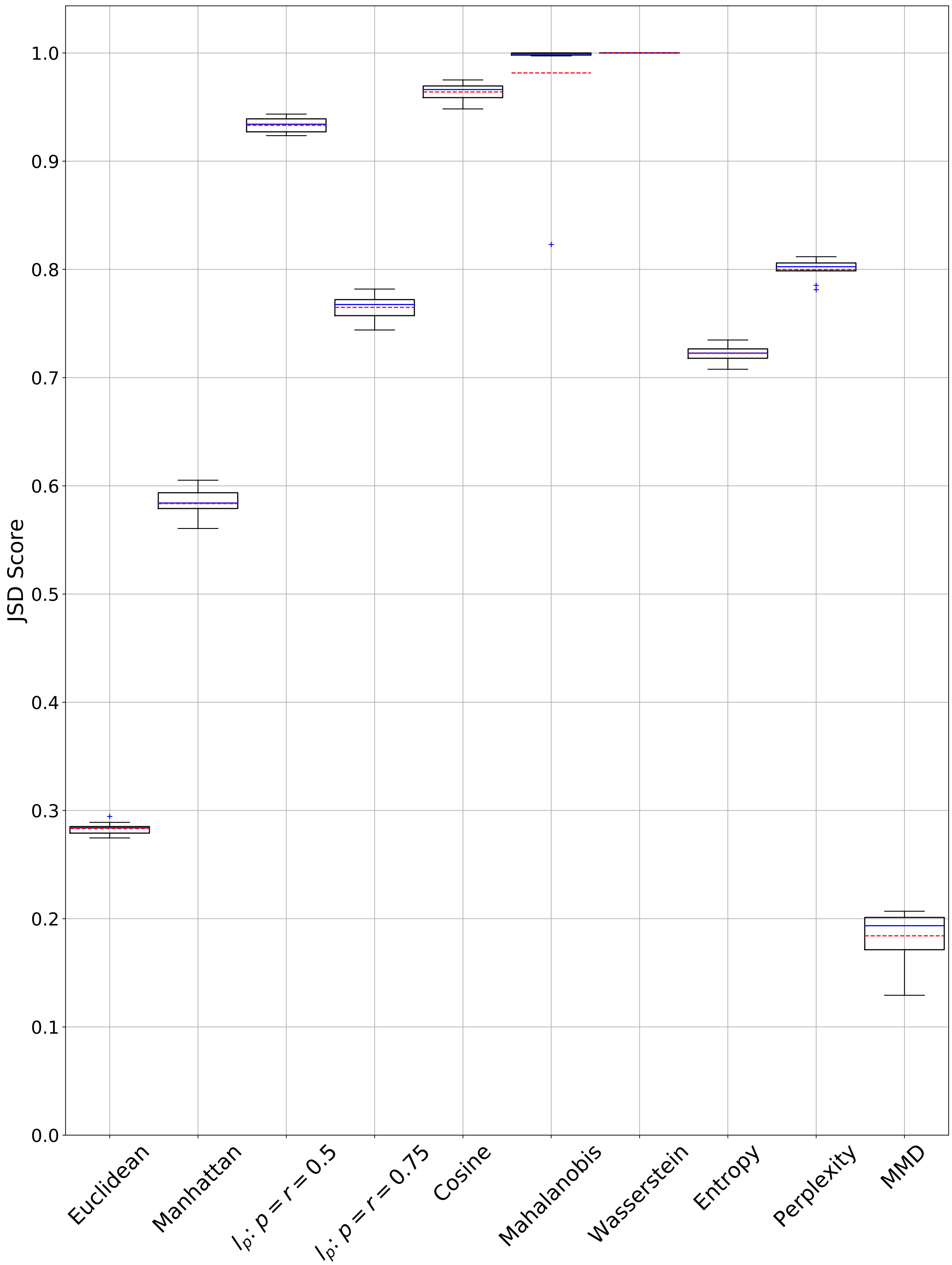

Figure 18: Untransformed uniform Host Events boxplot of JSD scores for 10 runs of the experiment. Wasserstein maintains a JSD of 1.0 for all 10 runs, indicated by the box being just a single line. Mahalanobis also comes close but has an outlier run skewing the mean down.

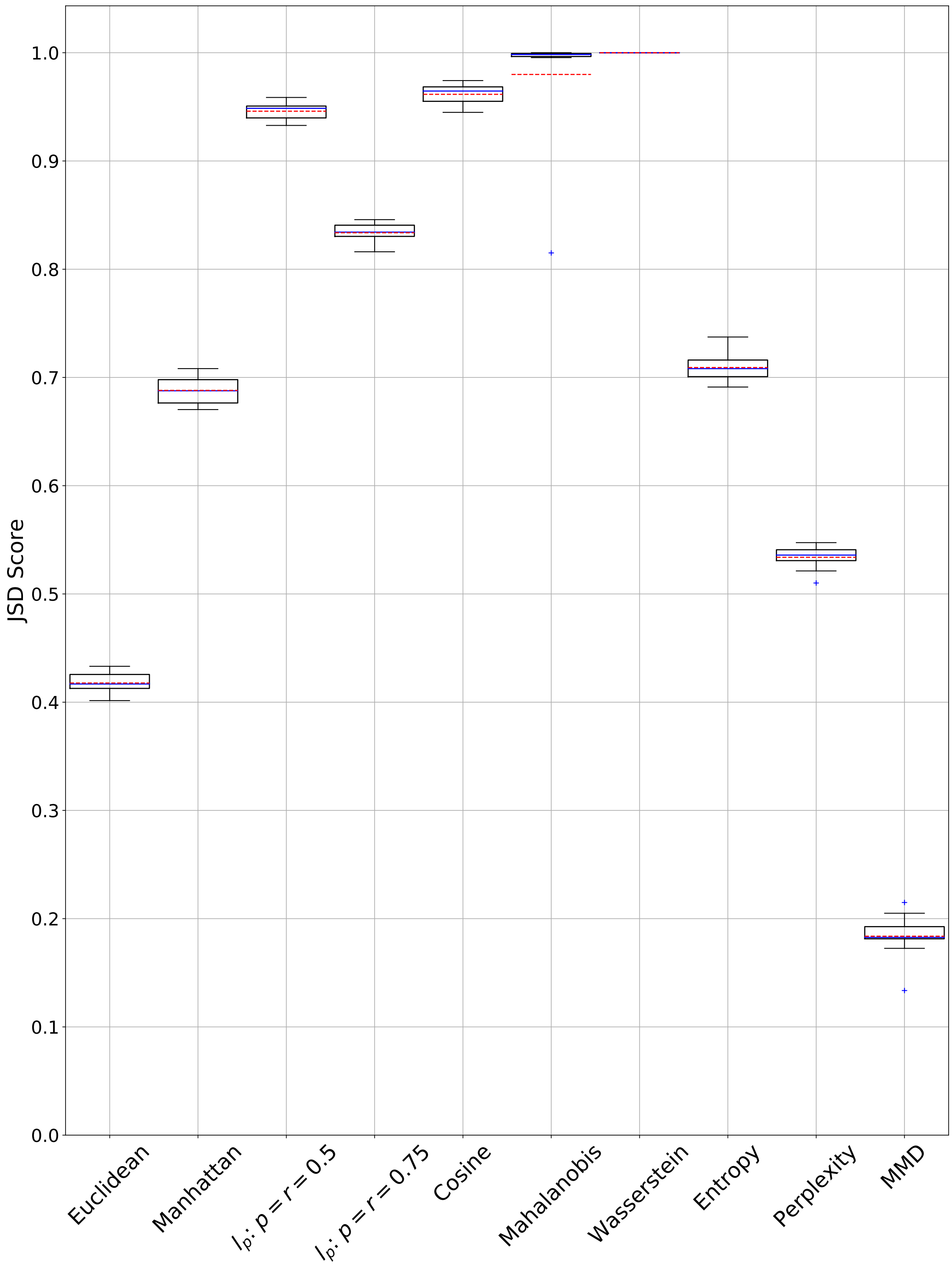

Figure 19: Untransformed normal Host Events boxplot of JSD scores for 10 runs of the experiment. Wasserstein maintains a JSD of 1.0 for all 10 runs, indicated by the box being just a single line. Mahalanobis has an outlier run skewing the mean down. Perplexity also drops from the 0.8 range to the 0.5 range.

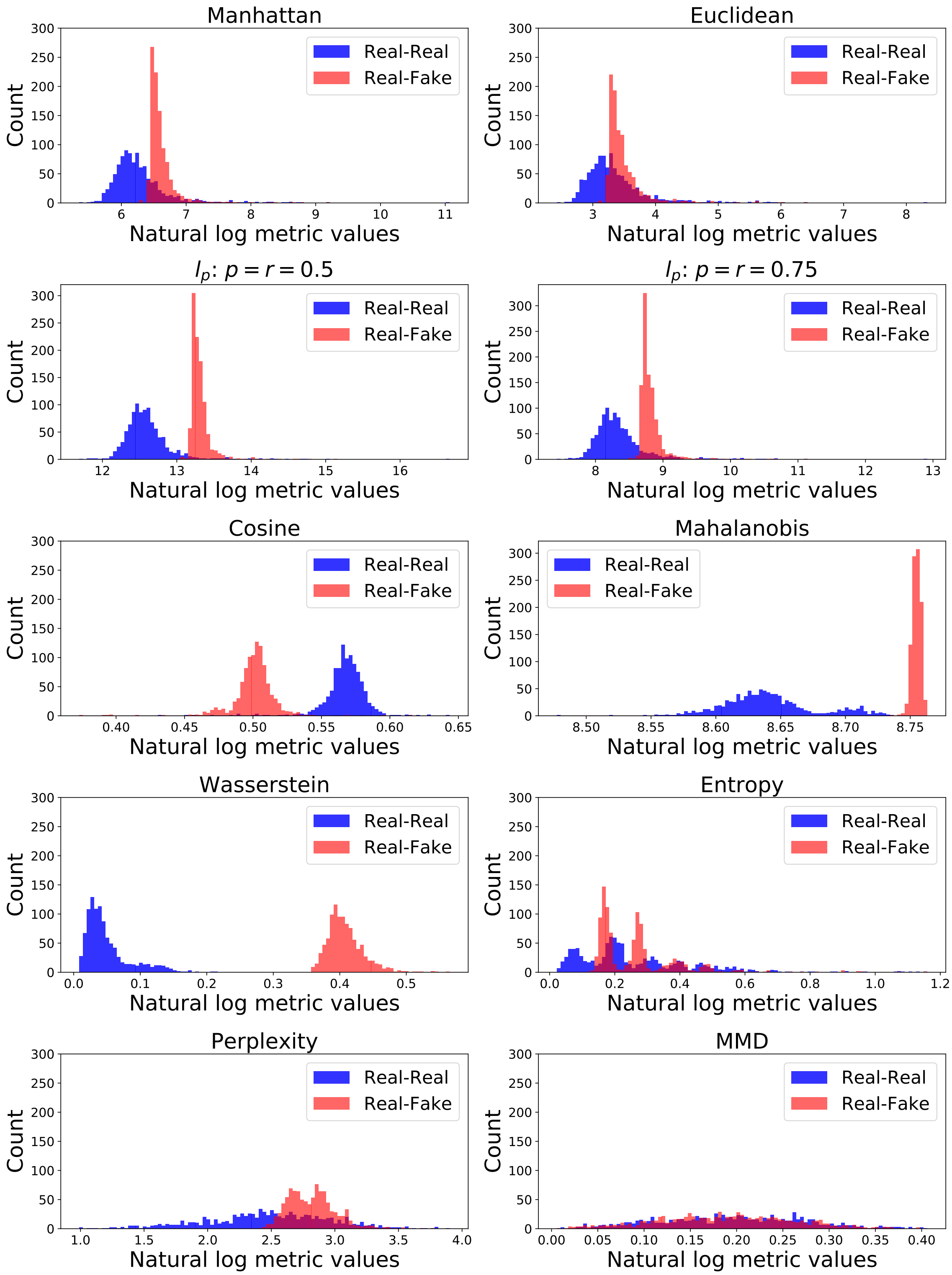

Figure 20: Discriminative results from SQRT transform on Host Events dataset. Differences in the R-R and R-F distributions are clearly visible for most of the metrics which corresponds to the scores in Table 10. This transform performs markedly better on the Host Events data than it did on the Network Events data.

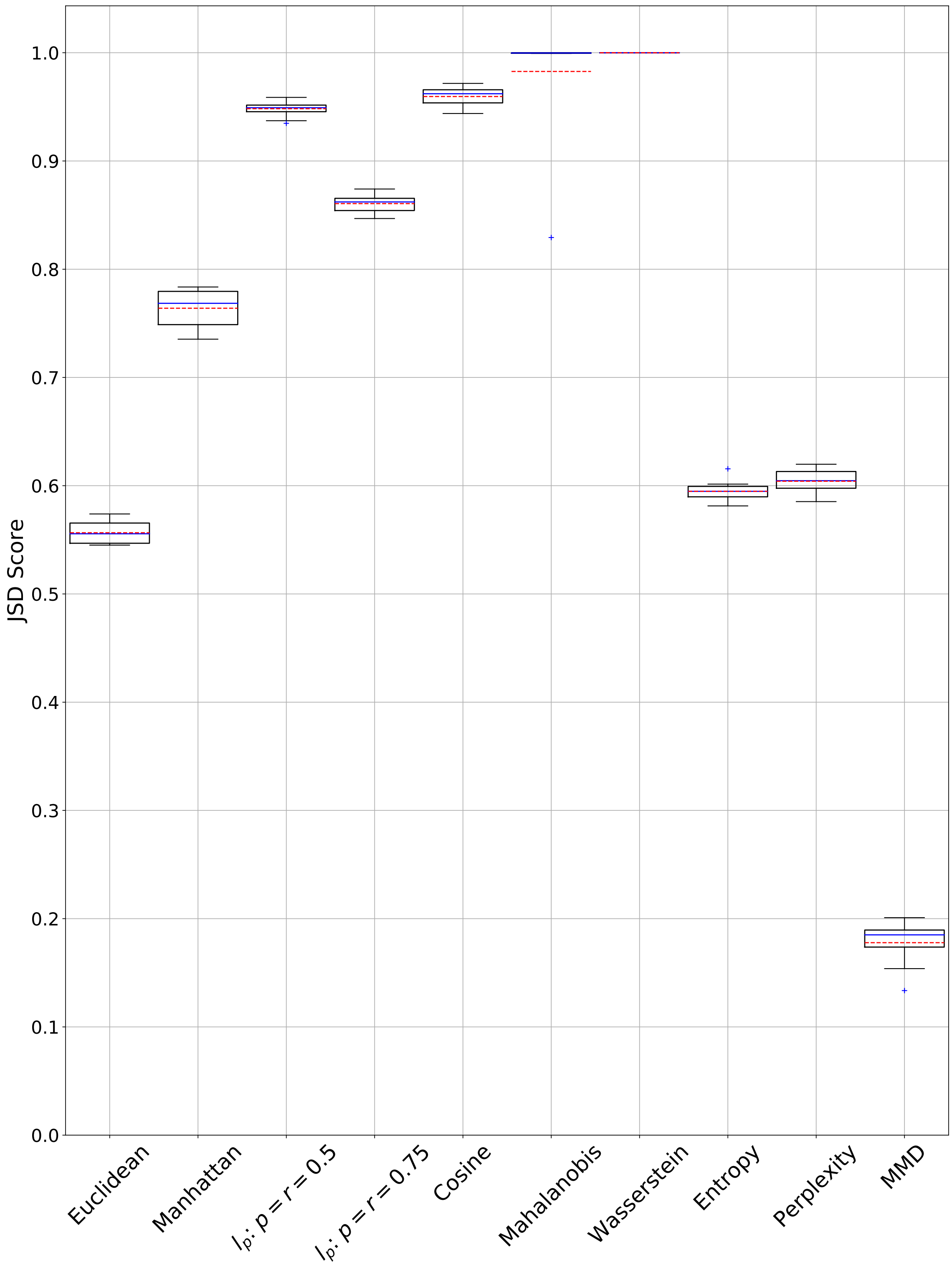

Figure 21: SQRT Transform uniform Host Events boxplot of JSD scores for 10 runs of the experiment. Wasserstein has a JSD of 1.0 for all 10 runs. Mahalanobis also contains the single outlier skewing the mean down. The JSD for Perplexity drops again, this time from around 0.6 down to just under 0.5 in the normal.

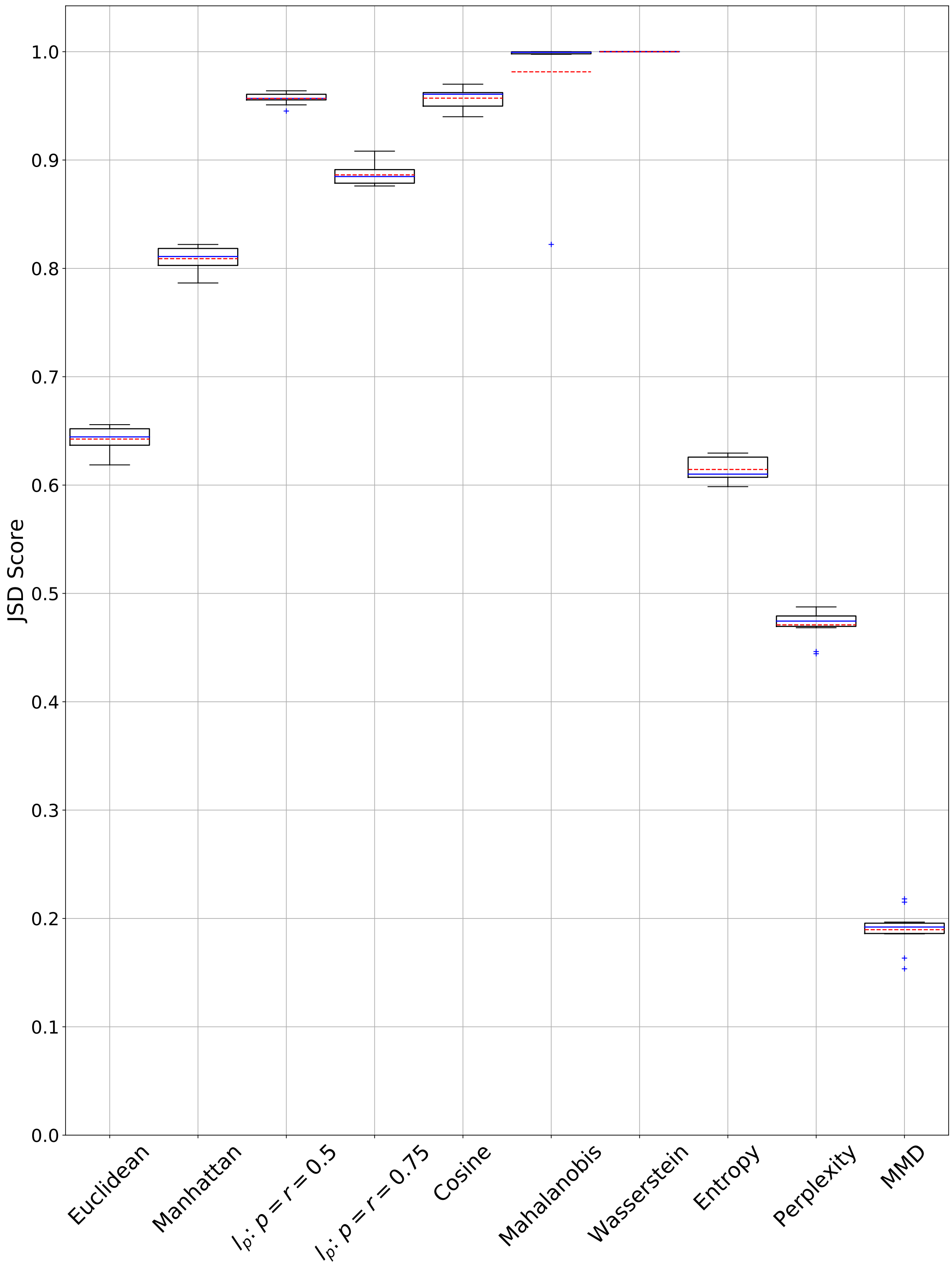

Figure 22: SQRT Transform normal Host Events boxplot of JSD scores for 10 runs of the experiment. Wasserstein has a JSD of 1.0 for all 10 runs. Mahalanobis also contains the single outlier. Perplexity drops from around 0.6 down to just under 0.5. Few significant differences overall from the uniform results.

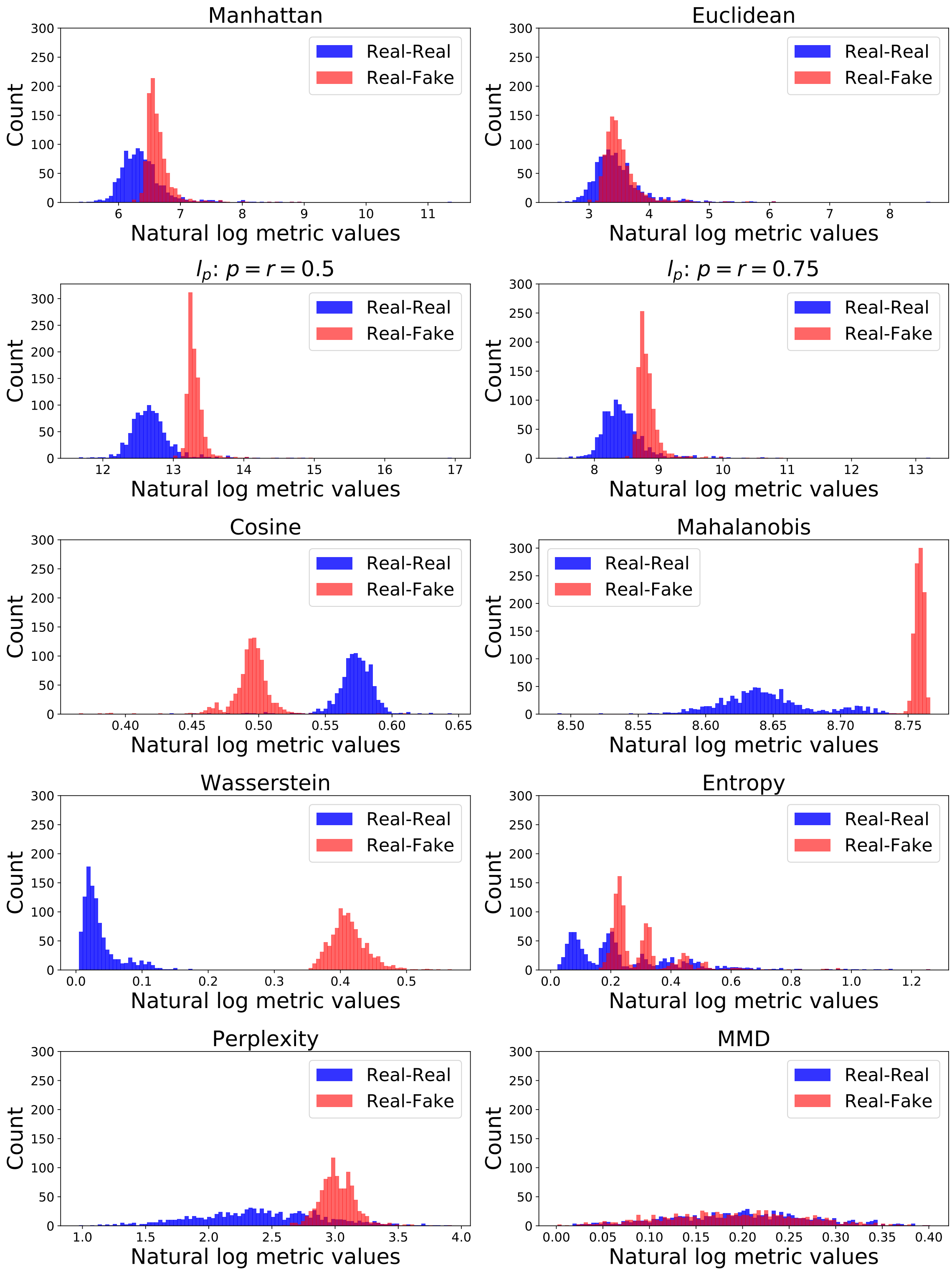

Figure 23: Discriminative results from log transform on Host Events data. Clear differences in the R-R and R-F are visible as indicated by the JSD scores from Table 12. This transform also performs much better on the Host Events data than on the Network events data.

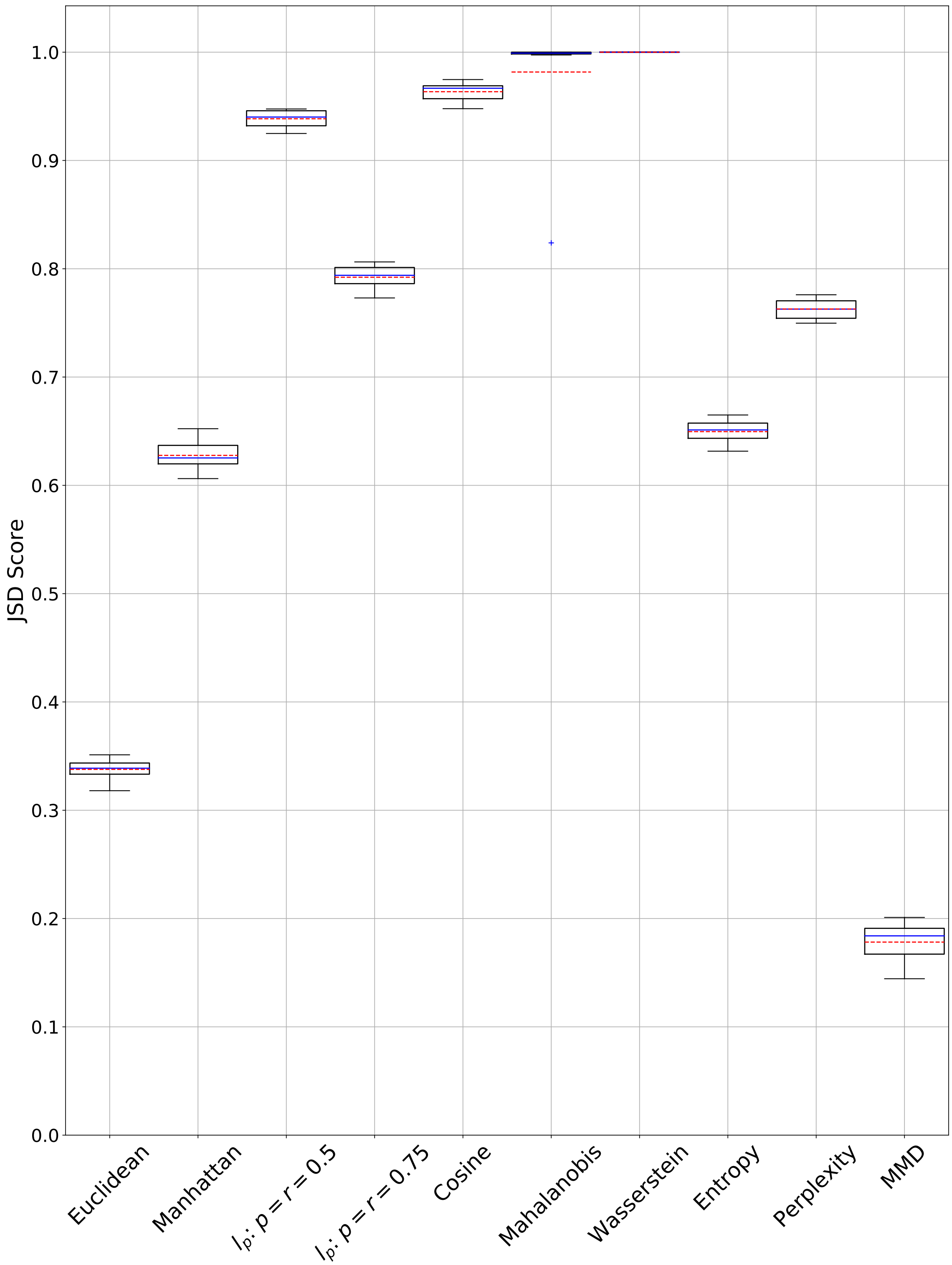

Figure 24: Log Transform uniform Host Events boxplot of JSD scores for 10 runs of the experiment. Wasserstein JSD is 1.0 for all 10 runs and Mahalanobis contains the single outlier which brings down the mean, as it did in the other transforms.

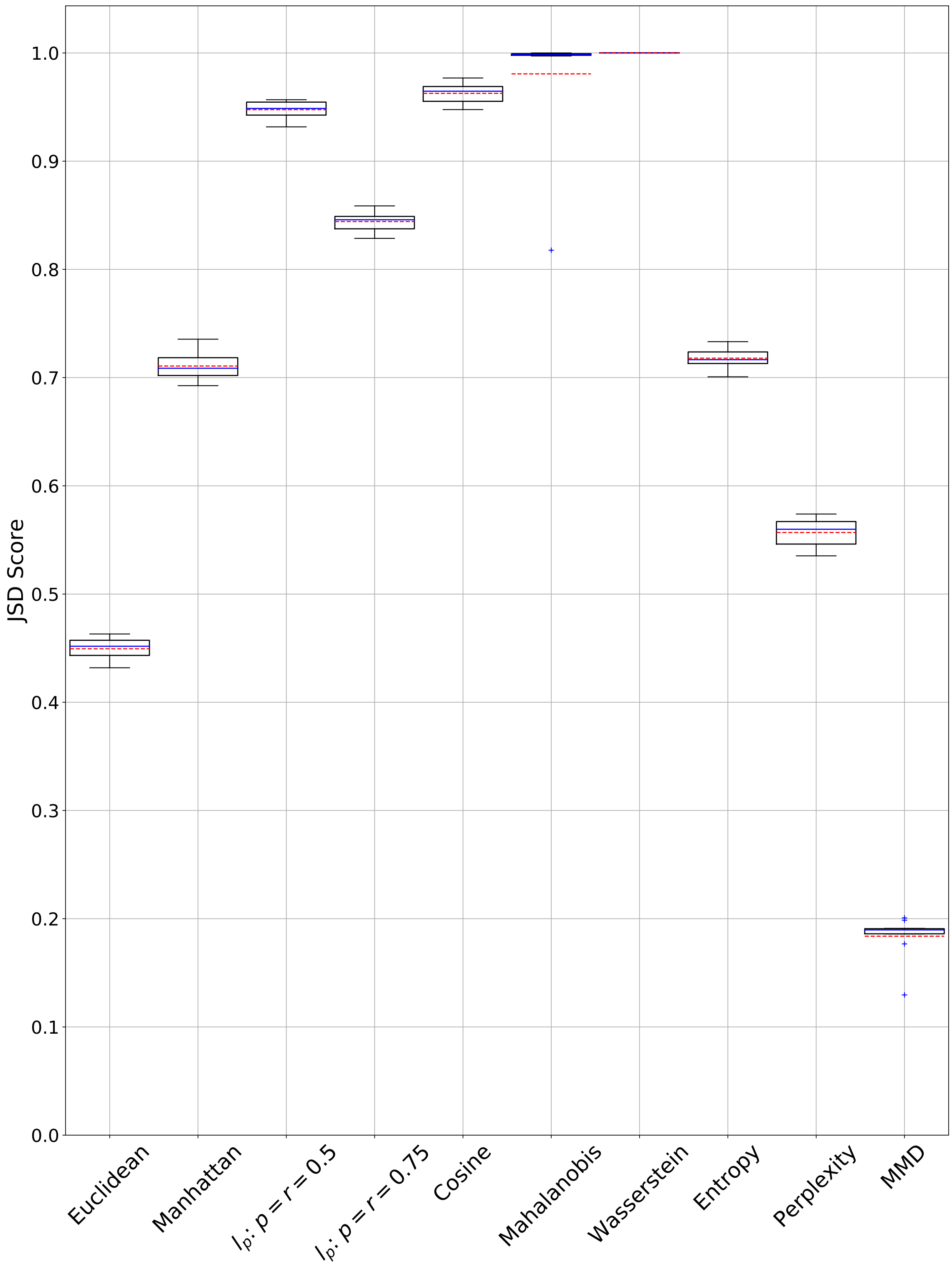

Figure 25: Log Transform normal Host Events boxplot of JSD scores for 10 runs of the experiment. Wasserstein JSD is 1.0 for all 10 runs and Mahalanobis contains the single outlier which brings down the mean. Like the other transforms, Perplexity JSD drops from around 0.75 in the uniform to around 0.55 in the normal.

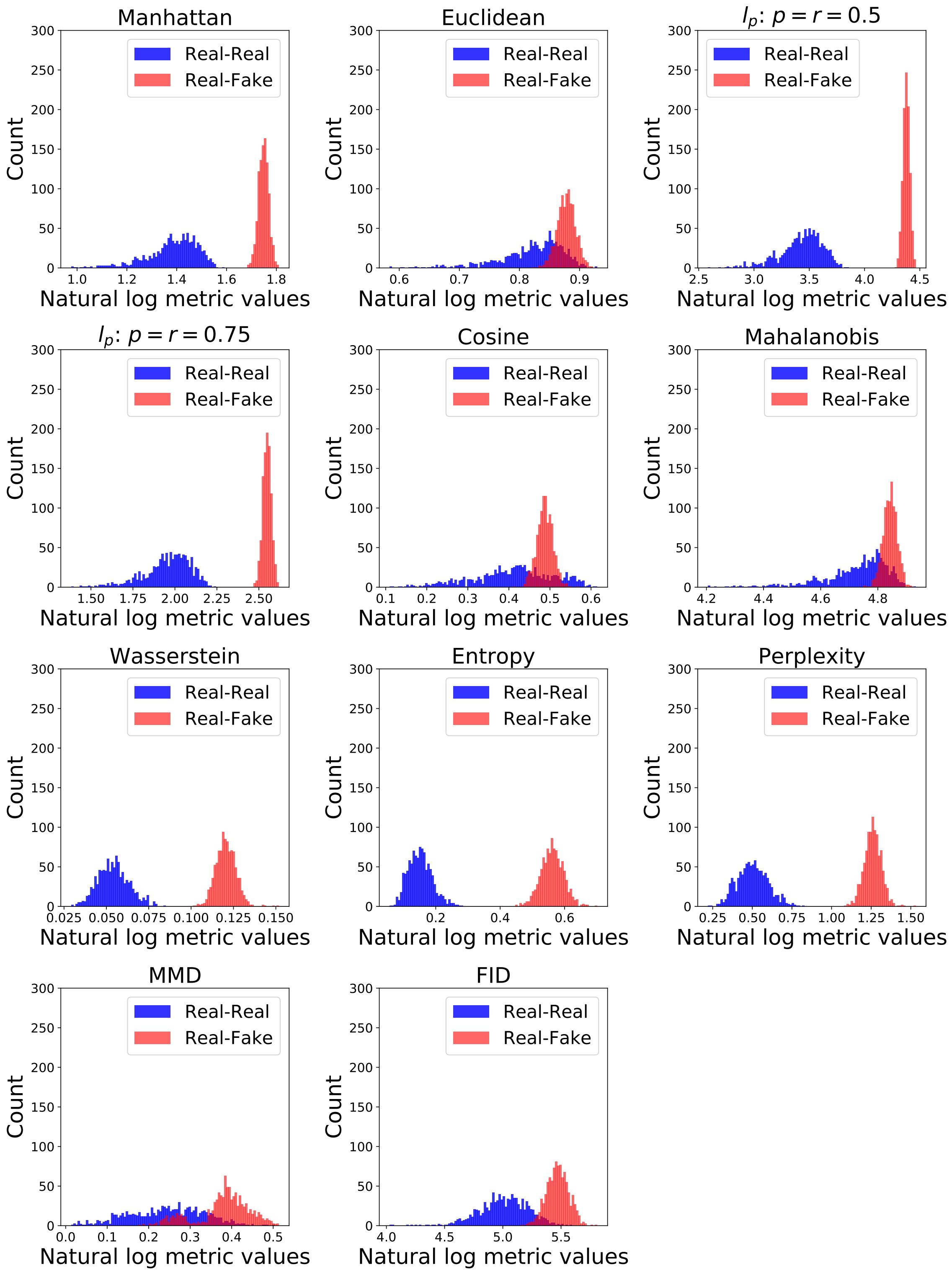

Figure 26: Discriminative results from PCA transform on Host Events data. Visible differences are noticeable between the R-R and R-F distributions for all 11 metrics. This is confirmed by Table 14 with all of the JSD scores above the 0.5 JSD threshold.

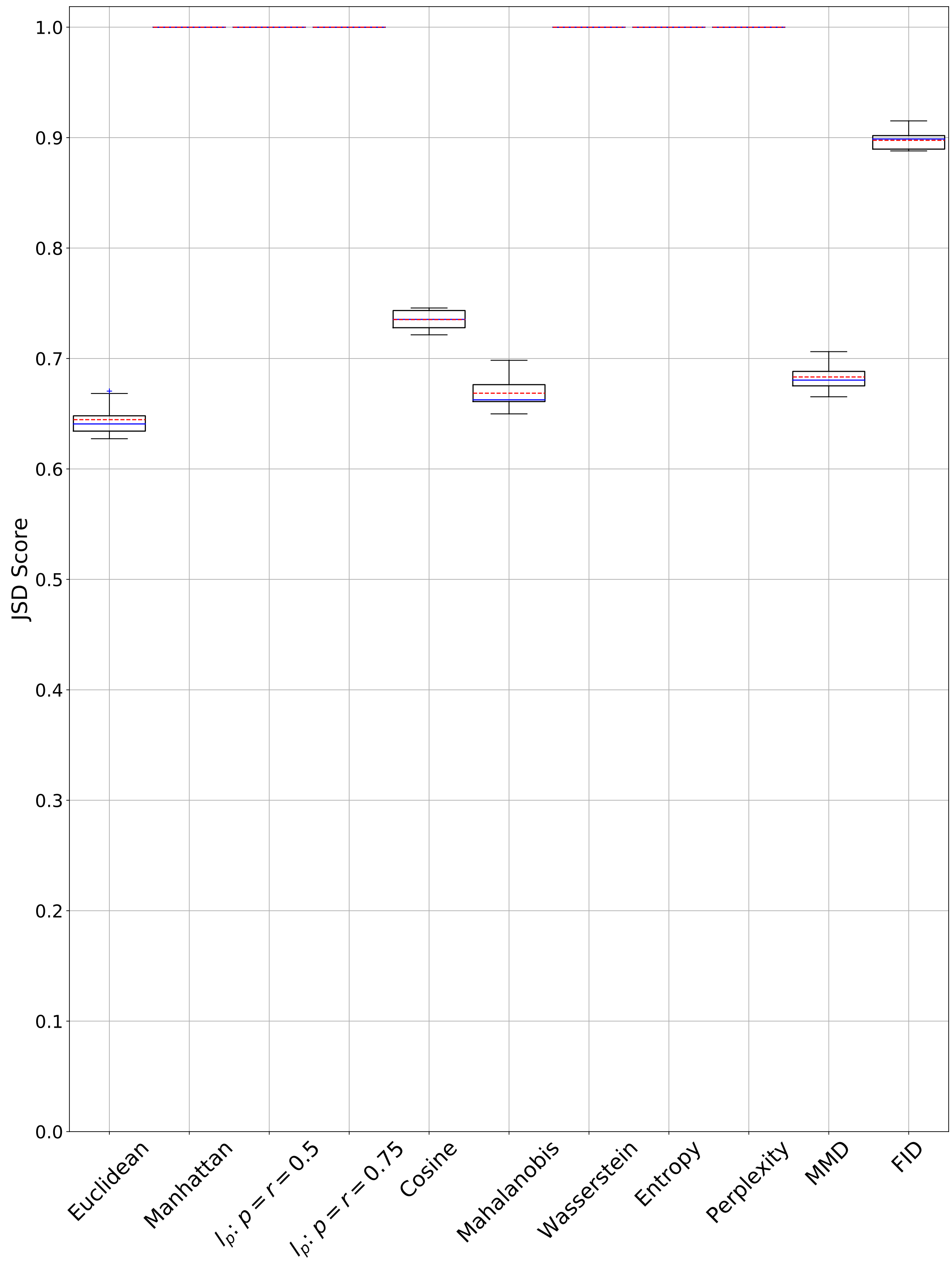

Figure 27: PCA Transform uniform Host Events boxplot of JSD scores for 10 runs of the experiment. This time, 6 of the metrics have a JSD of 1.0 for all 10 runs. Additionally, none of the boxes have any outliers.

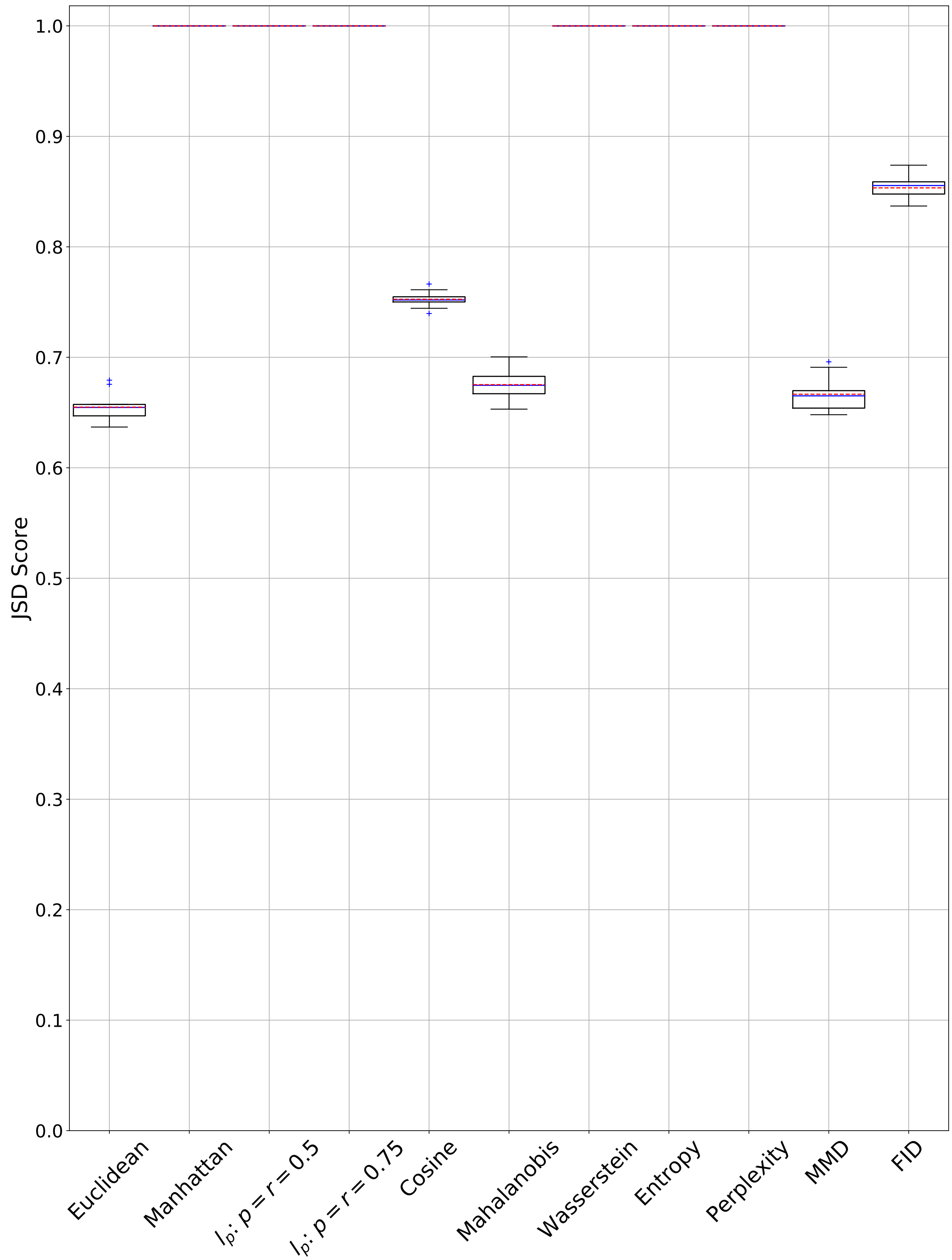

Figure 28: PCA Transform normal Host Events boxplot of JSD scores for 10 runs of the experiment. This time, 6 of the metrics have a JSD of 1.0 for all 10 runs. No drop in Perplexity JSD in this transform as it stays at 1.0 in both the uniform and normal.

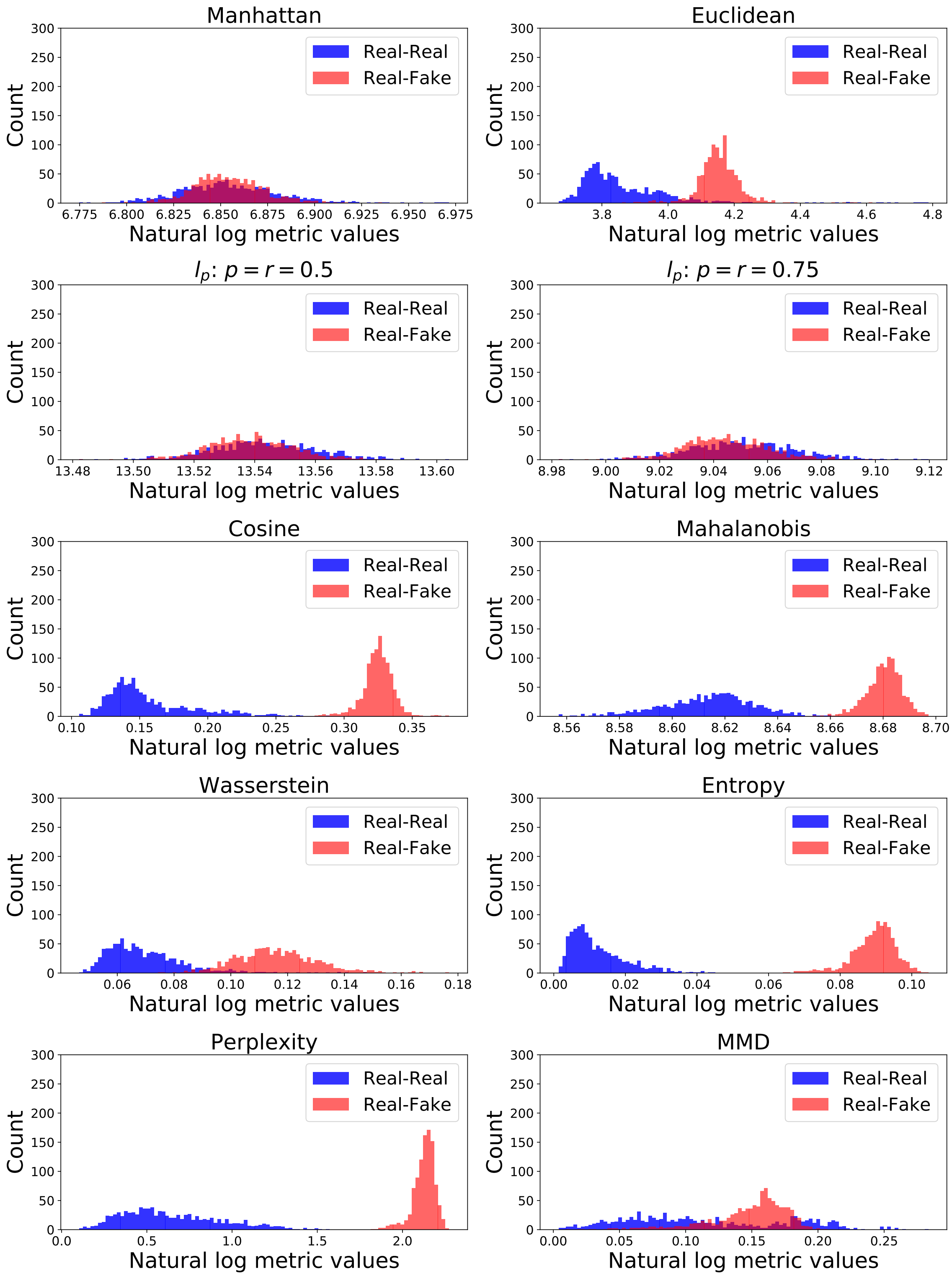

Figure 29: Discriminative results from the FFT transform on Host Events data. Differences in the R-R and R-F distributions are more noticeable than from the FFT transform on the Network Events data. Table 16 confirms this with 7 of the 10 metrics meeting the 0.5 JSD threshold.

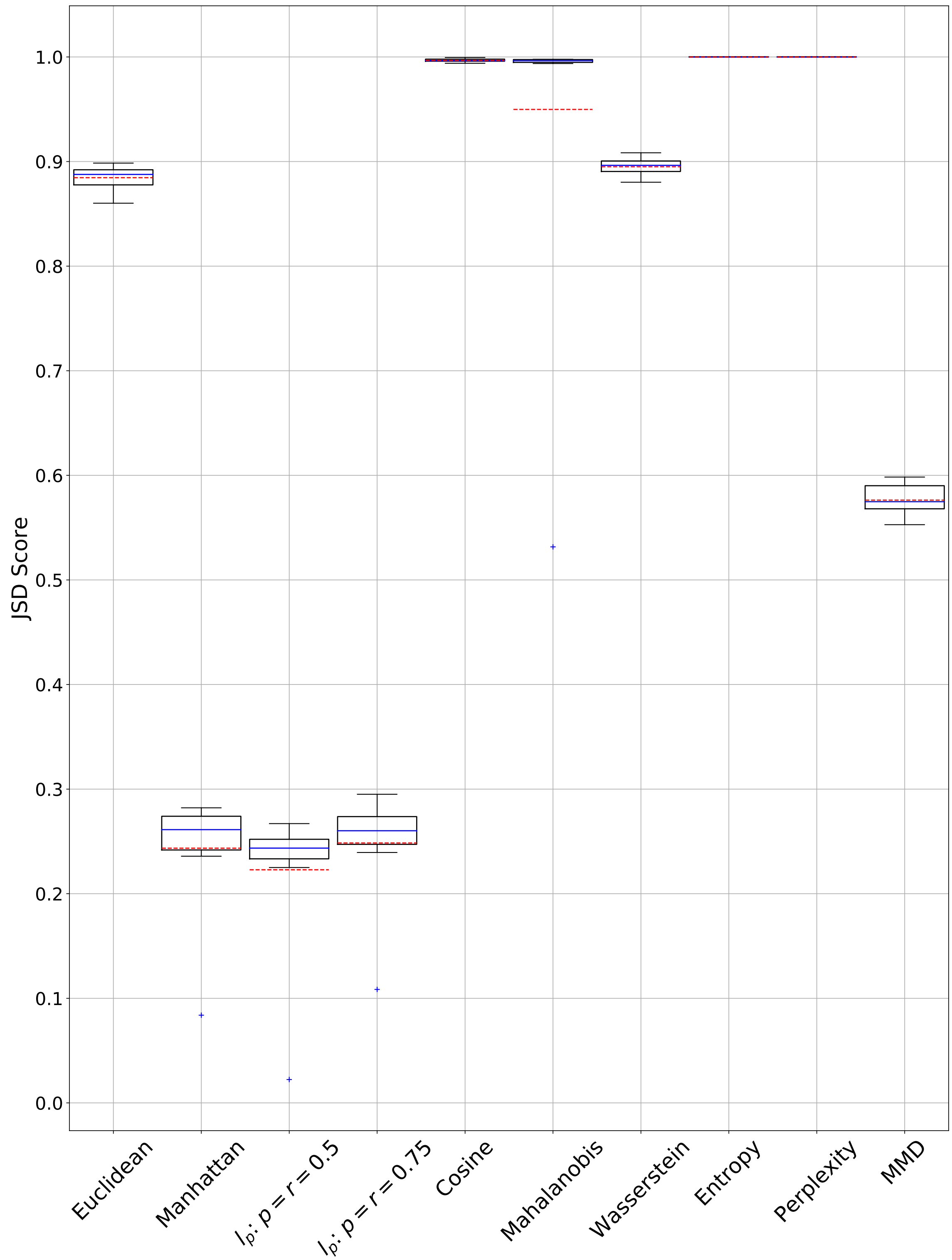

Figure 30: FFT Transform uniform Host Events boxplot of JSD scores for 10 runs of the experiment. Many of the metrics have very low outliers, producing larger differences in the mean and median than we have seen in the other transforms.

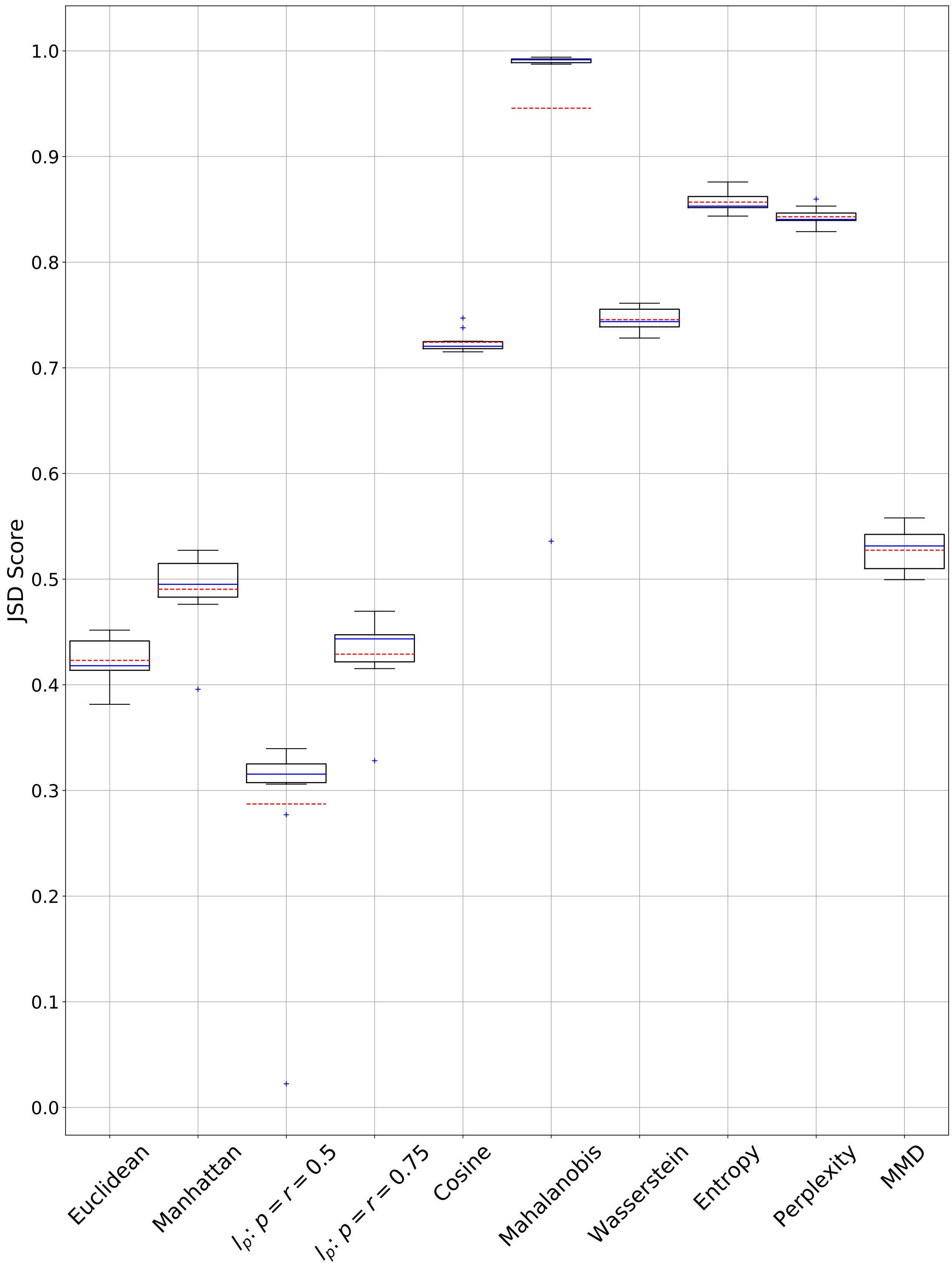

Figure 31: FFT Transform normal Host Events boxplot of JSD scores for 10 runs of the experiment. Manhattan distance experiences a drastic decrease from the uniform to normal, dropping from just under 0.9 to just above 0.4.

Efficiency

In this section, we examine the efficiency of the metrics chose for evaluation. Specifically, we explore the Computational Efficiency and Sample Efficiency.

4.2.1 Time Efficiency

The time efficiency experiment results for the Network Events data are displayed in Figure 32. The results on the Host Events data are displayed in Figure 33. The overall behavior is that the wall-clock time to calculate the metrics increases as the sample length increases which is expected. The wall-clock times displayed in Figure 32 and **??** are separated into two groups. The three metrics with the highest runtimes with both datasets are the FID, MMD, and Mahalanobis distance. These three metrics have $O(n^2)$ runtime complexity based on their implementations. The FID involves calculating a matrix multiply, which is $O(n^2)$. The MMD is also $O(n^2)$ in its complexity. The Mahalanobis distance is not $O(n^2)$, however, we use the Scipy `cdist()` function to calculate the Mahalanobis distance and it calculates pairwise distances between all elements of two collections, thus making it $O(n^2)$.

Examining Figure 32 and **??** further, we notice some non-monotonic behavior for some of the metrics. Generally, it would be expected that the runtimes should be monotonic increasing as the sample length increases. There are two likely reasons for this behavior. First, some of the metrics involve calculating a probability distribution based on input values and this time is included in the calculation. Thus, it is possible that due to background optimizations, a distribution could be generated faster with more samples based on the values of the input. If this occurs, then the sample with more lines could be calculated faster if these background optimizations occur.

Specific to the Host Events runtimes in Figure 33, the Term Frequency - Inverse Document Frequency (TF-IDF) process induces a large amount of zeros into the

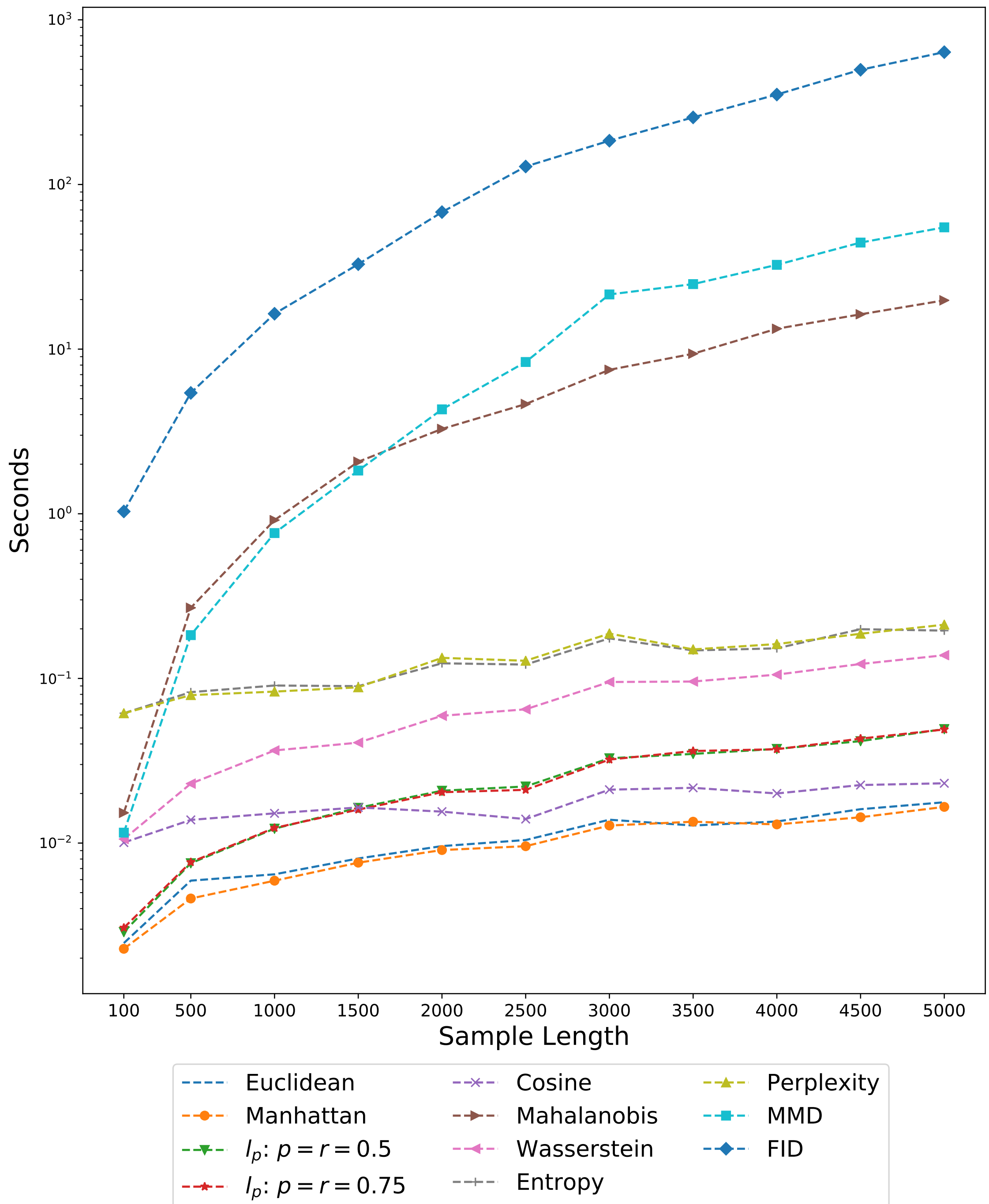

Figure 32: Wall-clock time (seconds) vs. sample length on Network Events. We see increasing times as the sample length increases as expected. The three metrics with the highest runtimes are the $O(n^2)$ complexity metrics while the other metrics are $O(n)$ complexity.

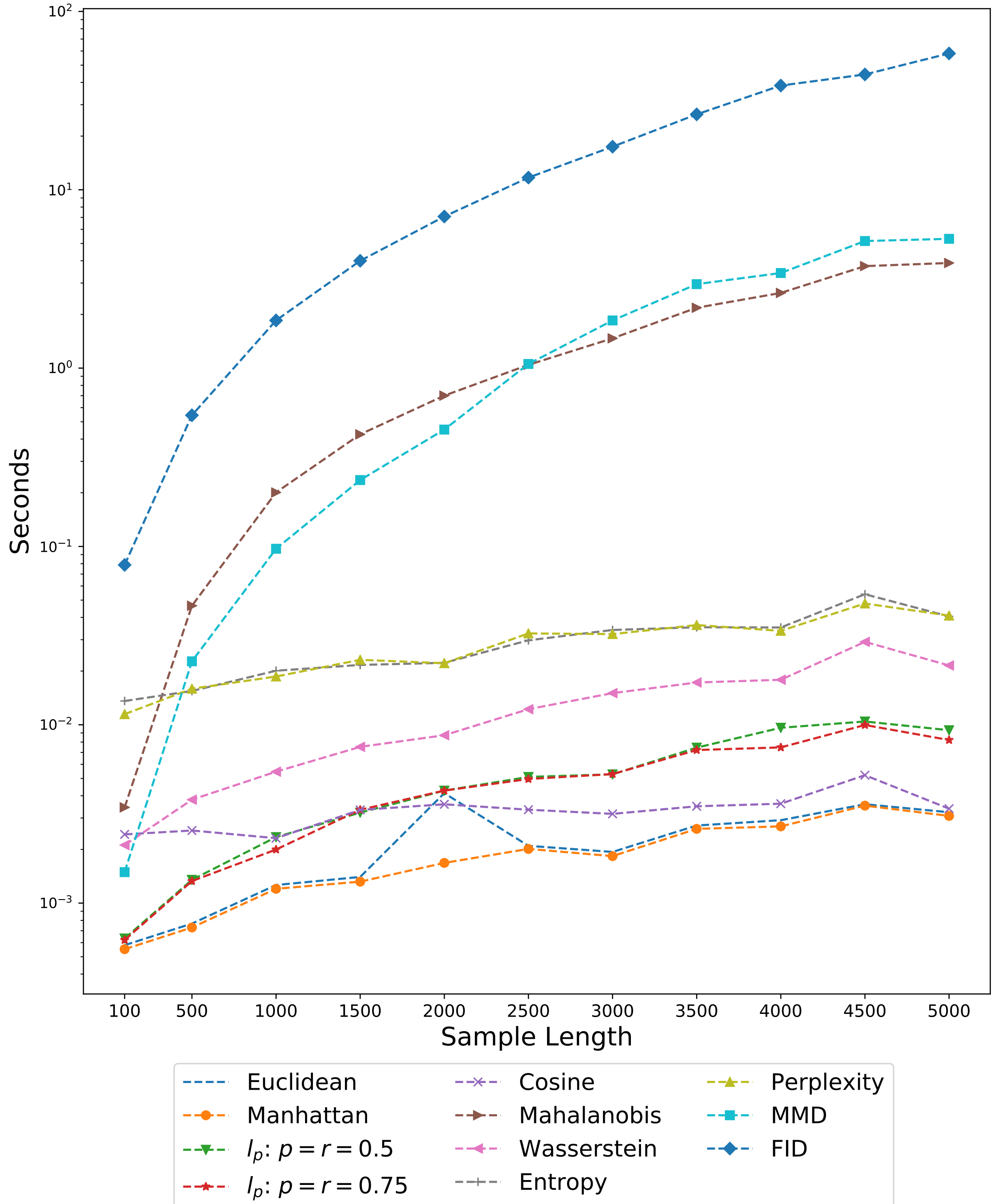

Figure 33: Wall-clock time (seconds) vs. sample length on Host Events data. We see increasing times as the sample length increases as expected. The three metrics with the highest runtimes are the $O(n^2)$ complexity metrics while the other metrics are $O(n)$ complexity.

samples. Based on this, it is possible that a sample with larger length could be more sparse. If this sparsity does occur, it is possible that background optimizations could occur that make the calculation of the metric faster with more lines.

Based on these wall-clock times, we see that the choice of metrics to use depends on application needs and sample lengths. If the application requires larger sample lengths, it may be best to stick with the lower wall-clock time metrics such as the fractional-l_p distances. If efficiency is not as much of a concern, then for smaller sample lengths, the $O(n^2)$ metrics may be suitable for use.

4.2.2 Sample Efficiency - Network Events Data

We received some unexpected results from the sample efficiency experiments. The expected outcome as detailed in [15] was that the JSD scores for the metrics would increase as the sample set size increased. What we observed instead, as shown in Figures 34 to 38, that the JSD score stays relatively constant as the number of samples increases. We had expected to see an increase in the JSD as the number of samples increased. However, depending on the transform we see that most of the JSD scores stay the same or in some cases slightly decrease. Additionally, we see that the transform applied can have an effect on the variance of the JSD score. The SQRT data in Figure 35 shows all of the metrics with relatively low variance while the log data in Figure 36 and PCA data in Figure 37 have much higher variance overall.

What these figures show however, is that the choice of 1,000 samples is an appropriate choice for the number of samples in the discriminative experiments. In each figure the JSD score at 1,000 samples is generally representative of the mean JSD score. As a reminder, 1,000 samples refers to the number of individual samples. We are using 1,000 line samples as well but the same number here is just coincidence.

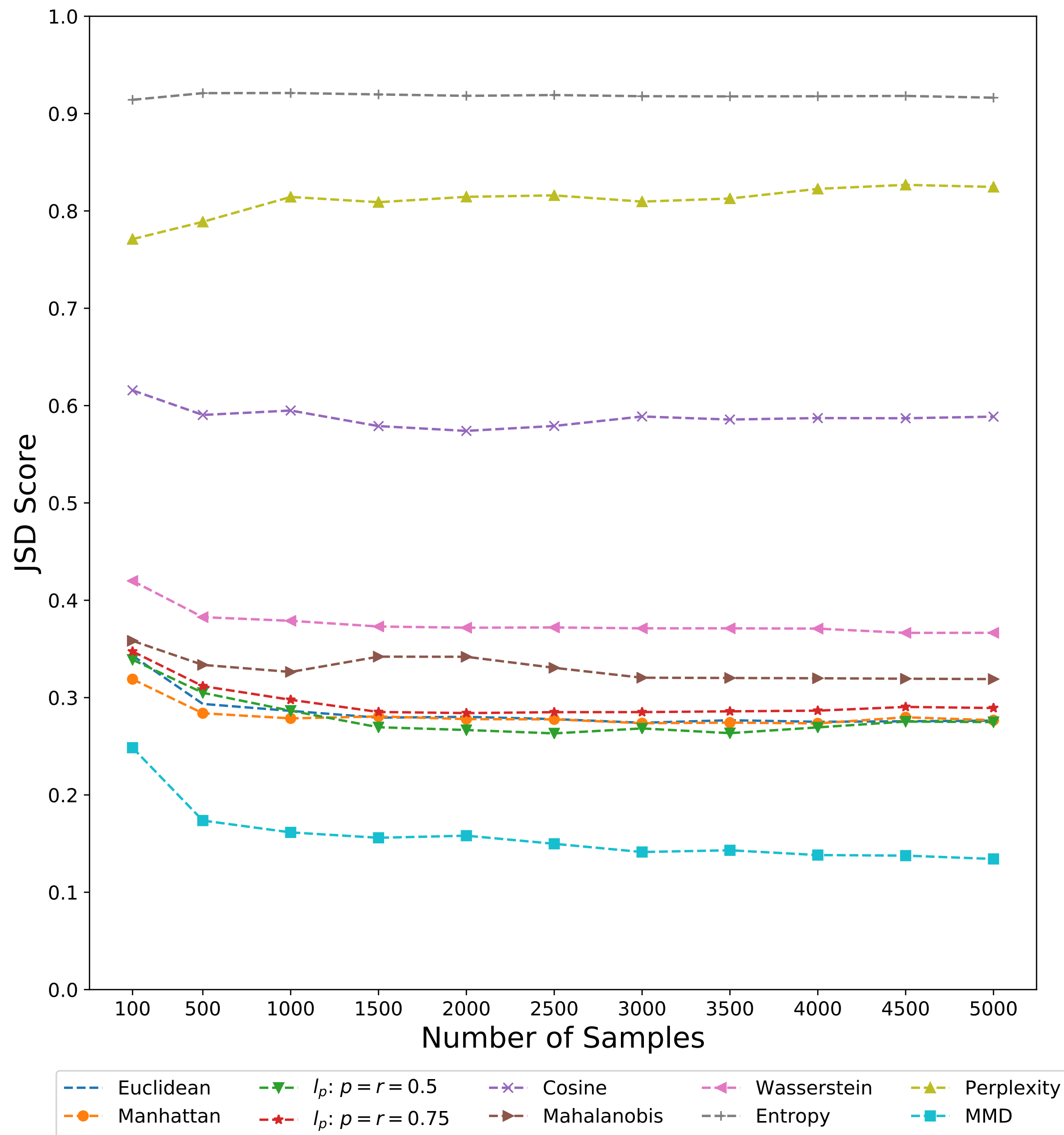

Figure 34: JSD scores vs. number of samples for untransformed Network Events data. The JSD scores stay relatively constant as sample size increases. These scores demonstrate the same ordering as shown in Table 3. However, due to different random seed, the JSD values may differ slightly from Table 3.

4.2.3 Sample Efficiency - Host Events Data

The results of the sample efficiency experiments on the Host Events data differ slightly from the Network Events Data. The results of the sample efficiency experiments are shown in Figures 39 to 43. Here we see mixed results. Some of the metrics

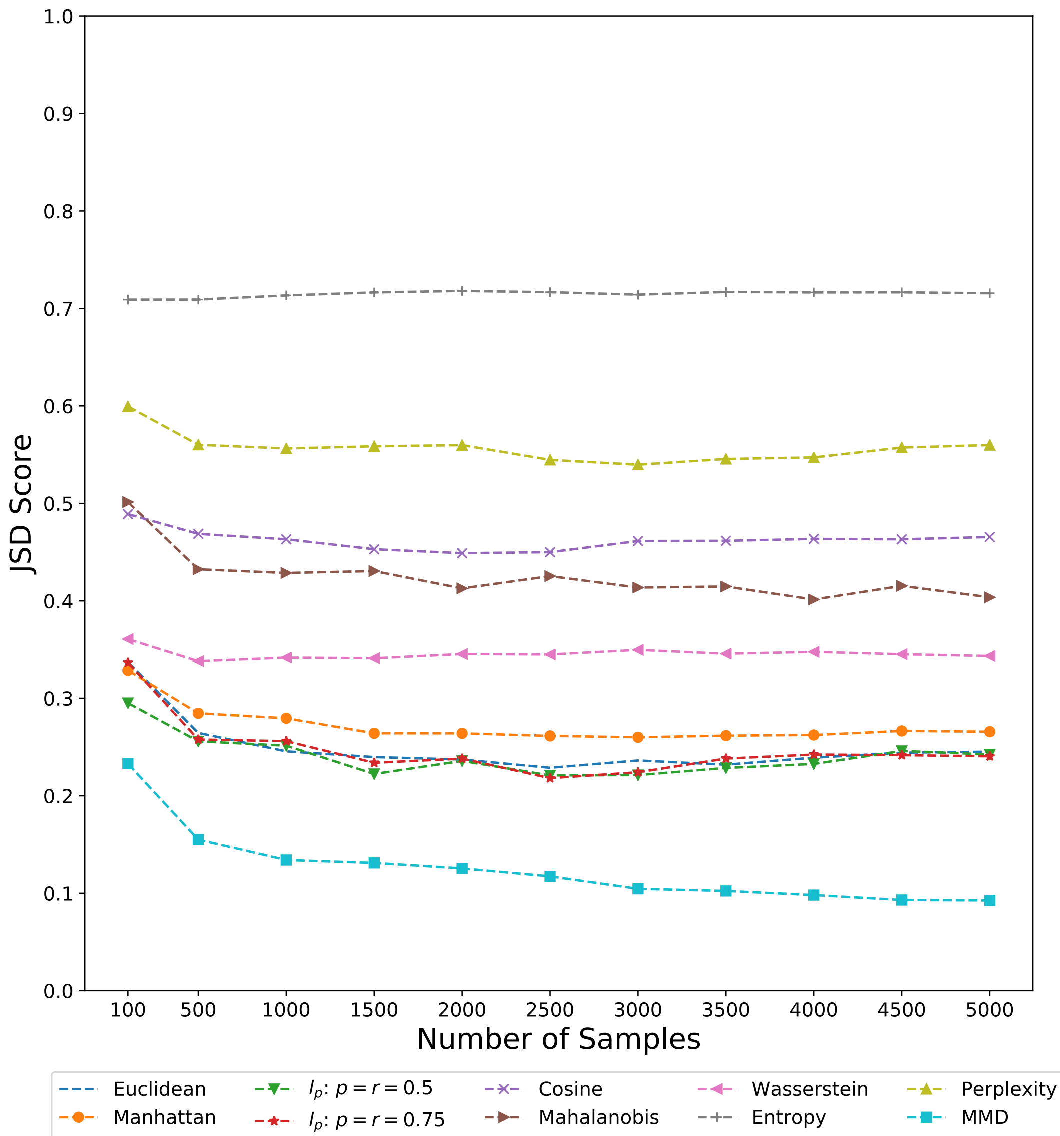

Figure 35: JSD scores vs. number of samples for SQRT transformed Network Events data. JSD score for 1,000 samples is a good representative sample size for this transform. The overall order is also the same as in Table 4.

demonstrate the increasing JSD as we expected, for example the $l_p, p = r = 0.5$ metric in Figure 39. However, the JSD scores for the Mahalanobis distance in particular exhibit a tendency to decrease a large amount as the number of samples increases, a trend that none of the other metrics exhibit. In Figures 42 to 43 we see the large variances of the other transforms lessen and exhibit the more constant behavior

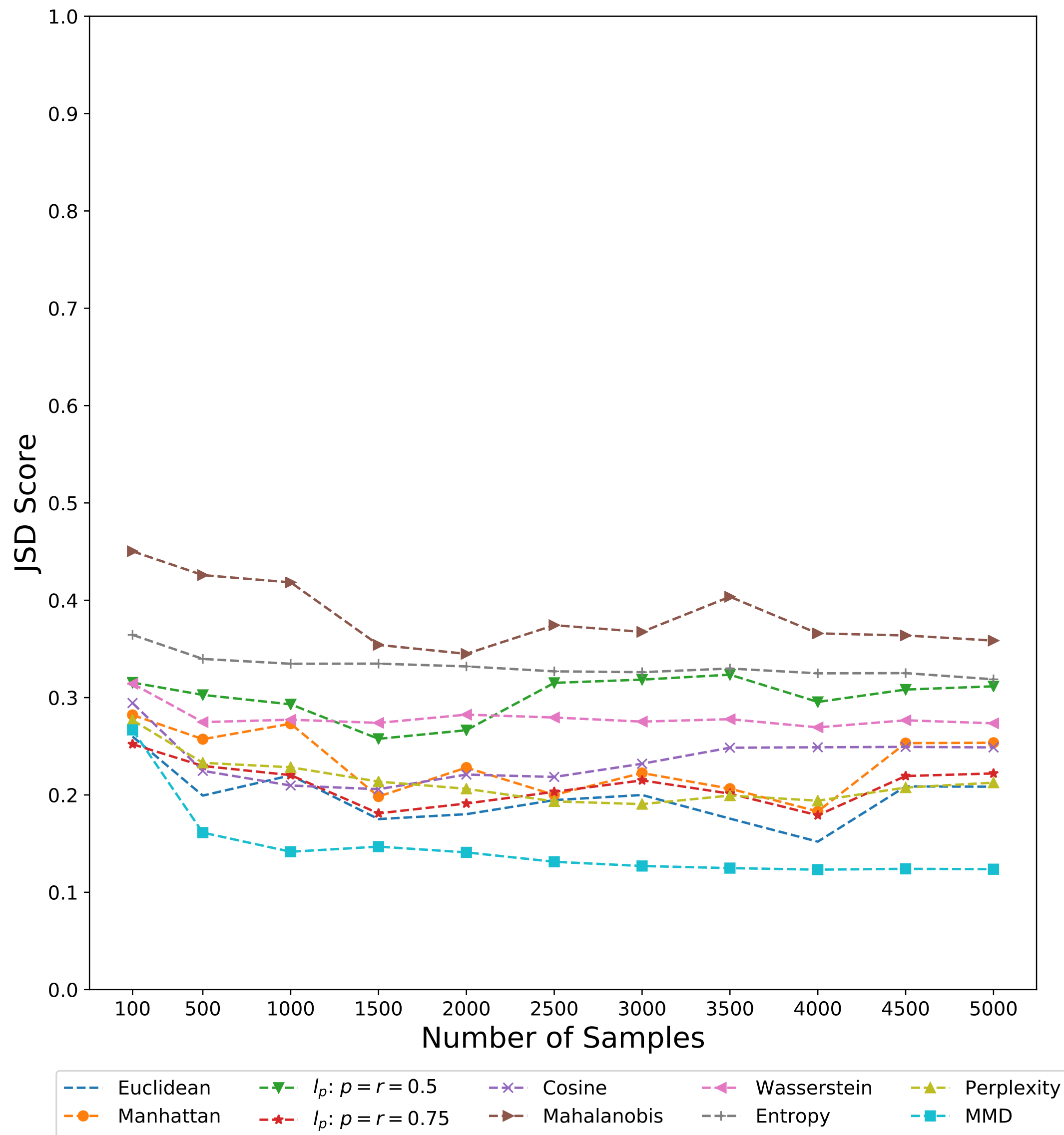

Figure 36: JSD scores vs. number of samples for log transformed Network Events data. The scale of the JSD scores reflects the overall poor results from Table 5.

demonstrated in the Network Events Data.

We also see that these results confirm the choice of 1,000 samples as the representative number of samples on the Host Events data as well. For most of the metrics in each of the figures, the 1,000 sample JSD appears to be a good representation of the JSD score for each metric.

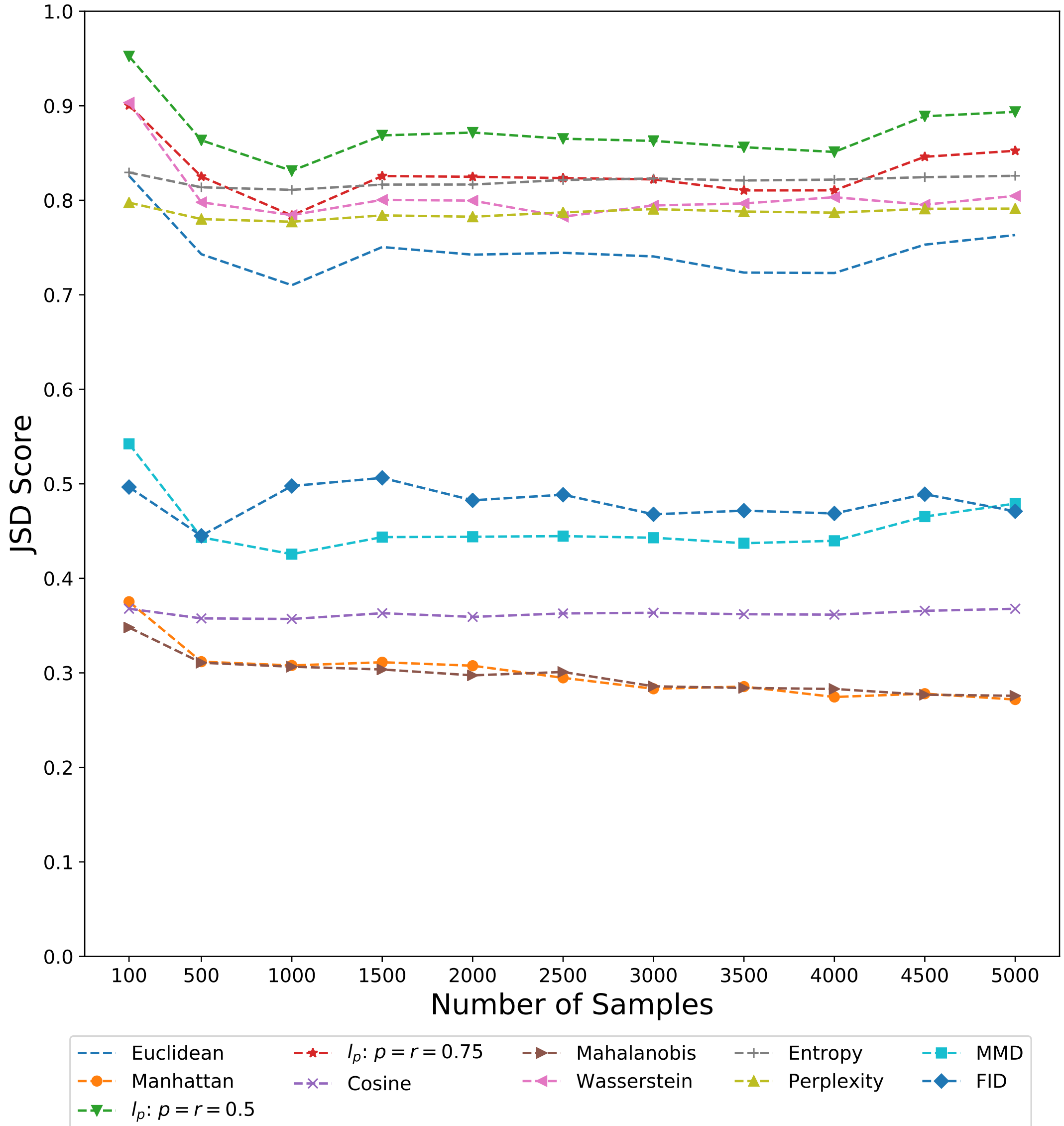

Figure 37: JSD scores vs. number of samples for PCA transformed Network Events data. Similar to the other transforms, the scores stay relatively constant as the number of samples increases.

It should be noted that there is an inflection point at sample size 500 or 1,000 on most of the lines in Figures 34 to 38 and Figures 39 to 43. Part of this inflection and steep increase is to due to a change in the scale, going from 100 to 500 rather than stepping by 500 as it does for the sample sizes 500 and above. There is also sometimes a large jump in JSD between 100 and 500. This is most likely due to the

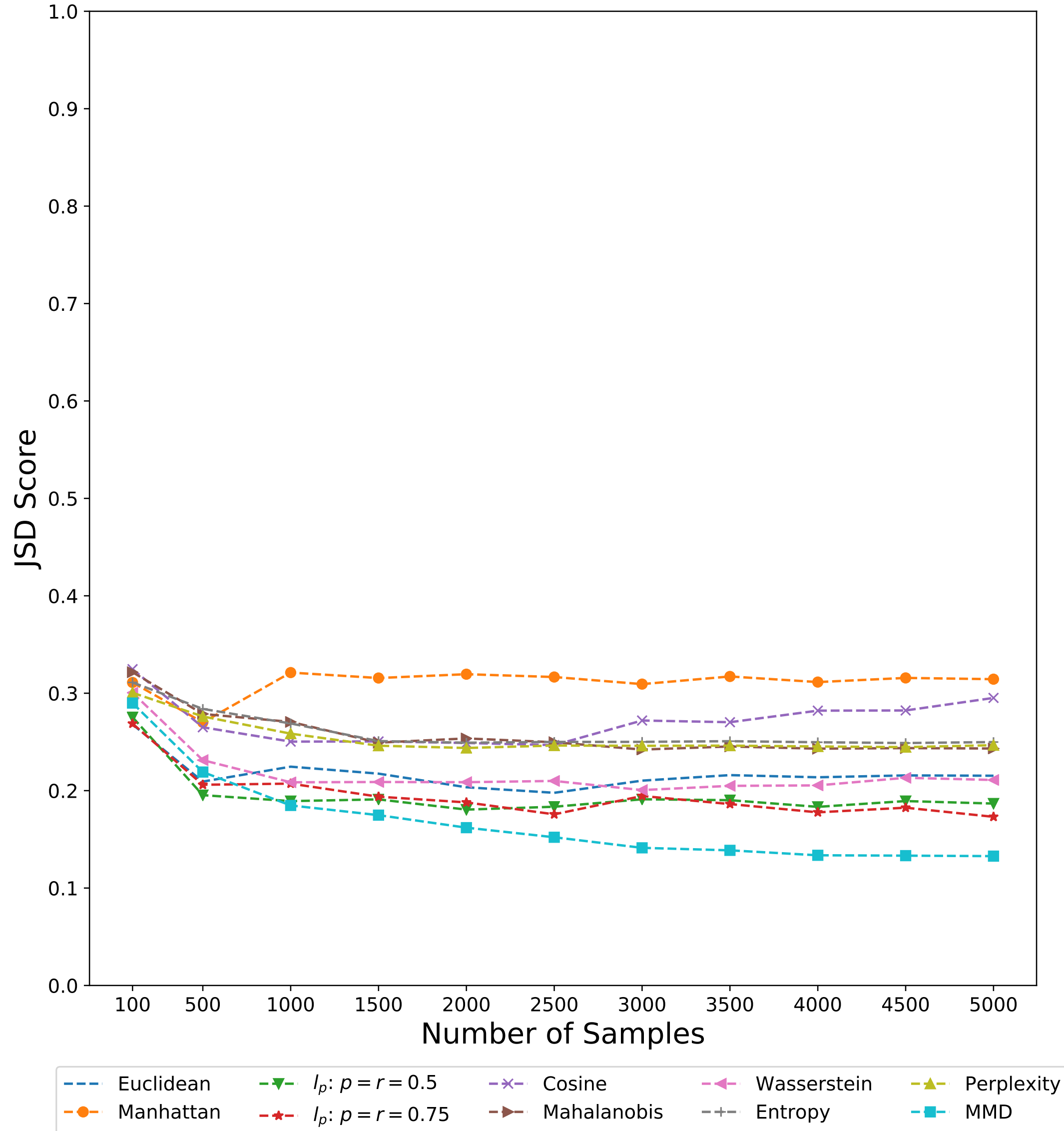

Figure 38: JSD scores vs. number of samples for FFT transformed Network Events data. As with the other transforms on the Network Events data, increasing the number of samples does not greatly affect the JSD values.

larger number of zero probability bins in the sample size 100, leading to the overall JSD value to be influenced by these bins with zero probability. Once sample size is increased to 500 and above we see that there are fewer inflection points in most cases.

Figure 44 and Figure 45 show the JSD scores vs. number of samples for untransformed Network Events and Host Events data respectively. Figure 44 appears

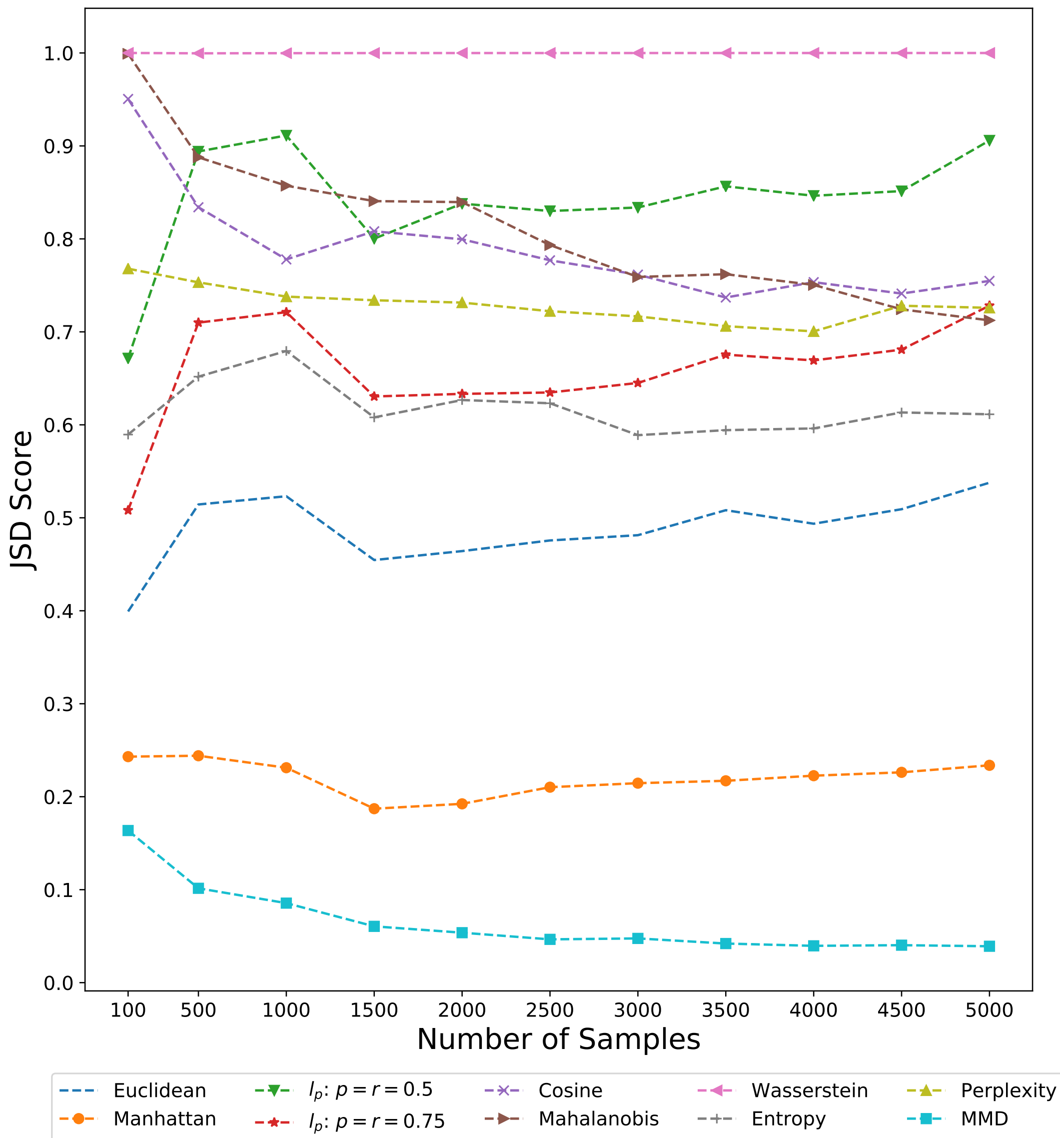

Figure 39: JSD scores vs. number of samples for untransformed Host Events data. The JSD scores for some of the metrics exhibit the same relatively constant behavior. The Mahalanobis distance is less stable as the number of samples increases, as indicated by its large decrease as the number of samples increase. Wasserstein stays at a constant 1.0 for all sample sizes, indicating a complete dissimilarity between the R-R and R-F distributions at all sample sizes

to be the same as Figure 34 with relatively constant behavior. Figure 45 examines the smaller sample sizes between 100 and 1,000 in more detail. Here we see that what appears to be large jumps between 100 and 1,000 samples in Figures 39 to 43

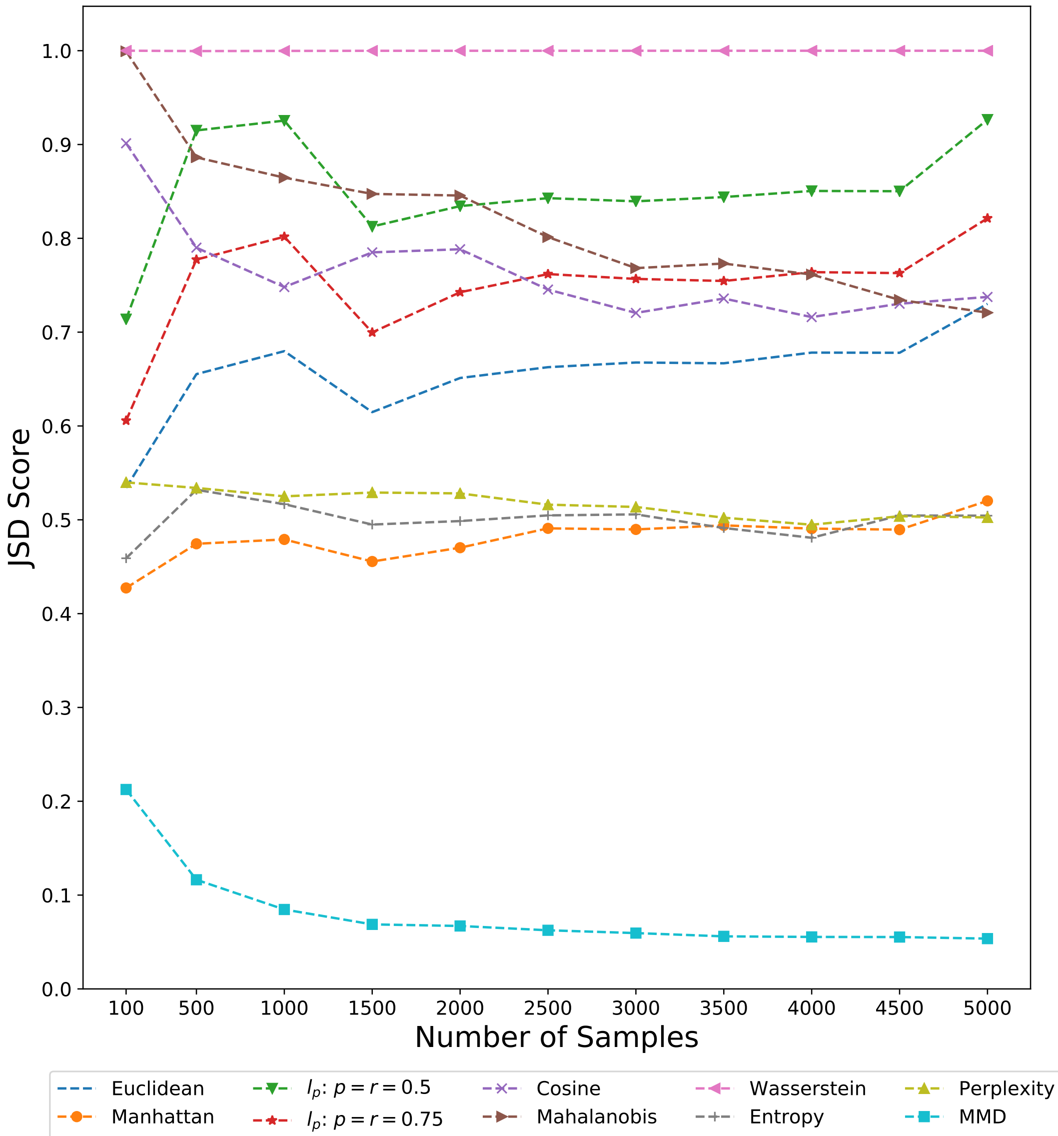

Figure 40: JSD scores vs. number of samples for SQRT transformed Host Events data. Once again, most of the metrics exhibit relatively constant behavior or slight increases in JSD as the number of samples increases. Mahalanobis distance again has a large decrease in JSD. Wasserstein stays at a constant 1.0 for all sample sizes, indicating a complete dissimilarity between the R-R and R-F distributions at all sample sizes

is less drastic when examined in scale. There is an increase in JSD as the number of samples increases which is probably due to the R-R and R-F distributions becoming more unique.

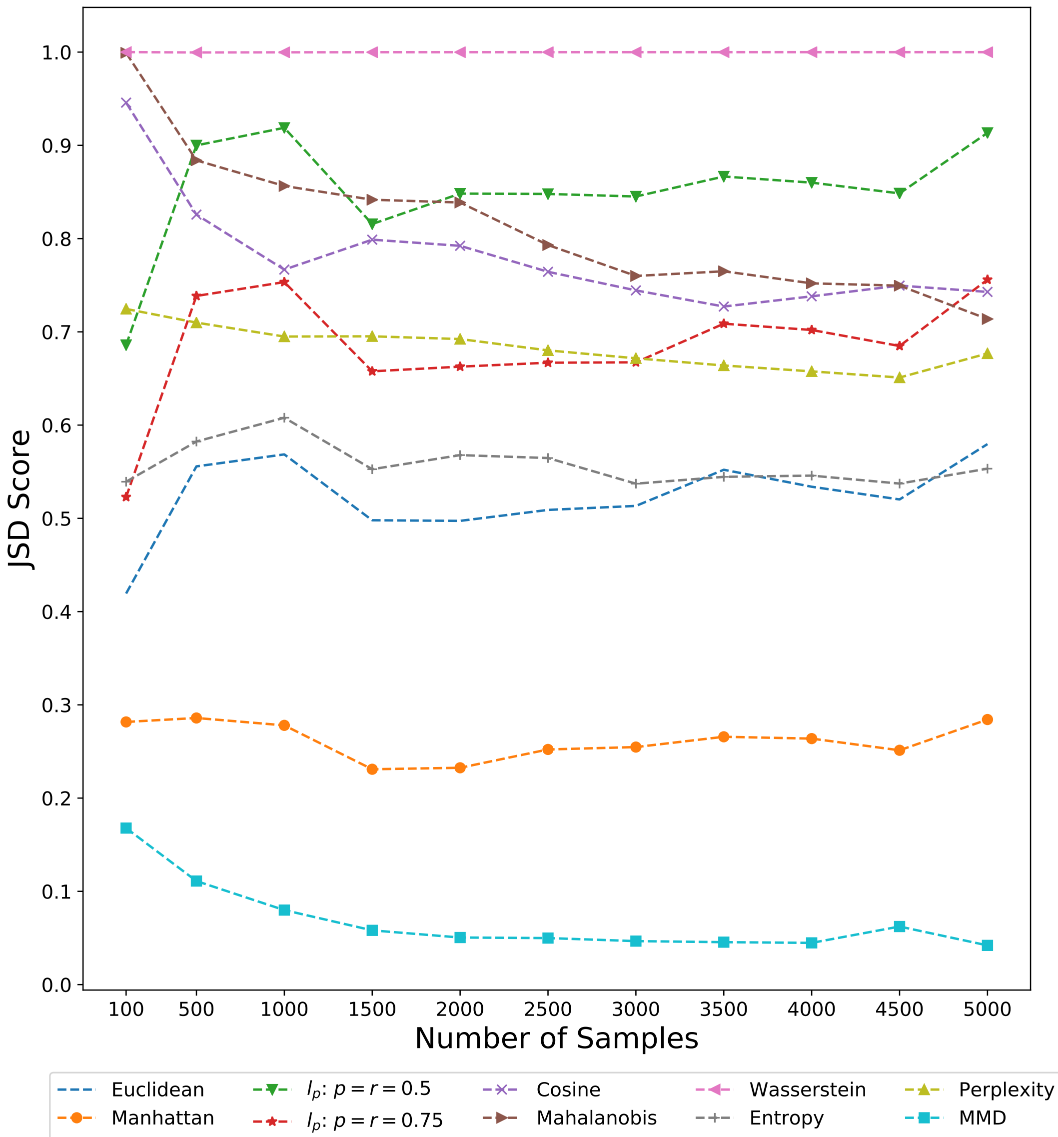

Figure 41: JSD scores vs. number of samples for log transformed Host Events data. Once again, most of the metrics have a relatively constant JSD as the number of samples increases. Again Mahalanobis distance JSD decreases quickly as number of samples increases. Wasserstein stays at a constant 1.0 for all sample sizes, indicating a complete dissimilarity between the R-R and R-F distributions at all sample sizes

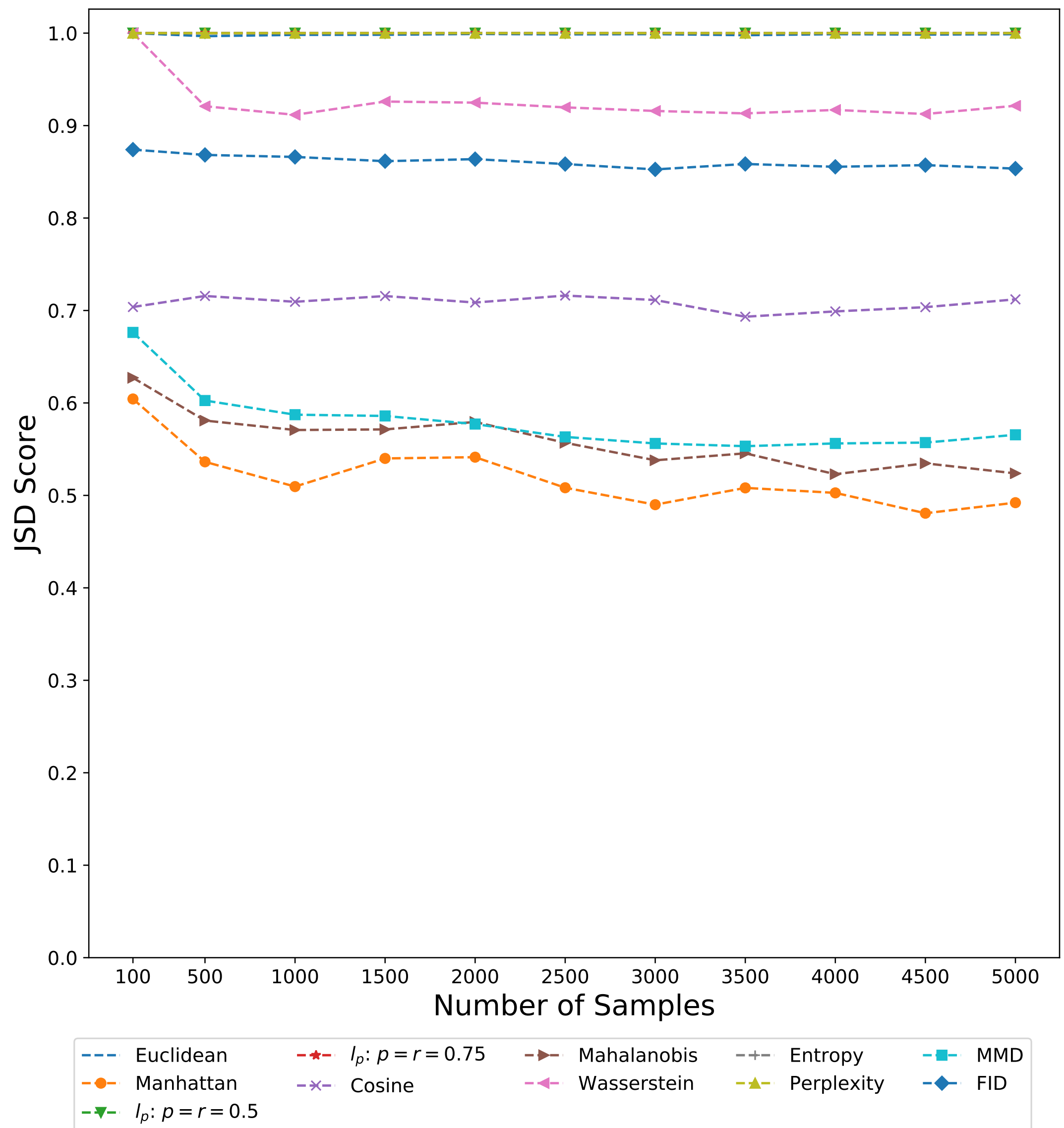

Figure 42: JSD scores vs. number of samples for PCA transformed Host Events data. There are some different behaviors here. The Mahalanobis and Euclidean distances both experience a drop in JSD as number of samples increases. Interestingly, with the Wasserstein distance this time, the Wasserstein distance has a JSD under 1.

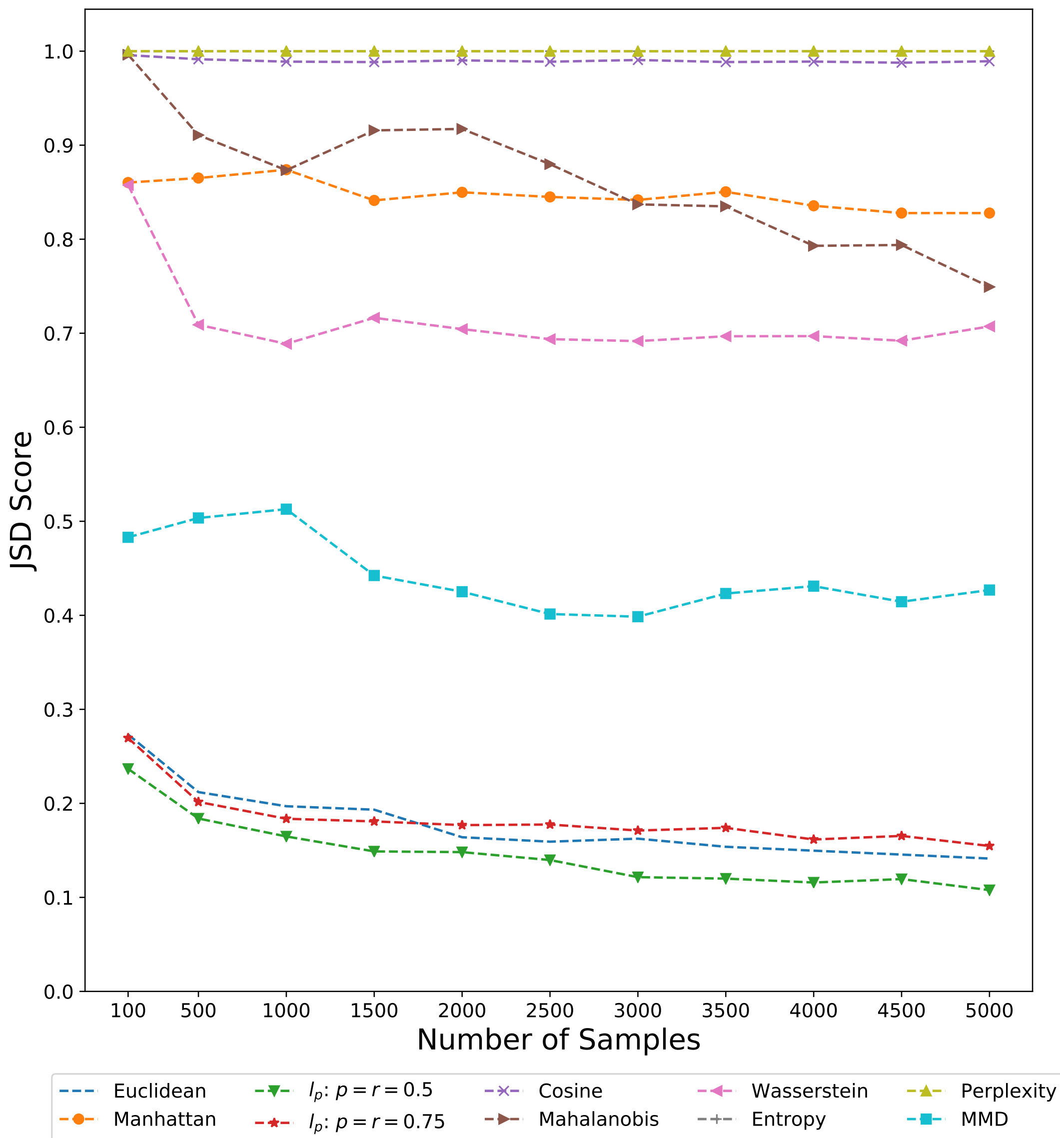

Figure 43: JSD scores vs. number of samples for FFT transformed Host Events data. Wasserstein distance again drops below 1.0 while Entropy and Perplexity have a JSD of 1.0.

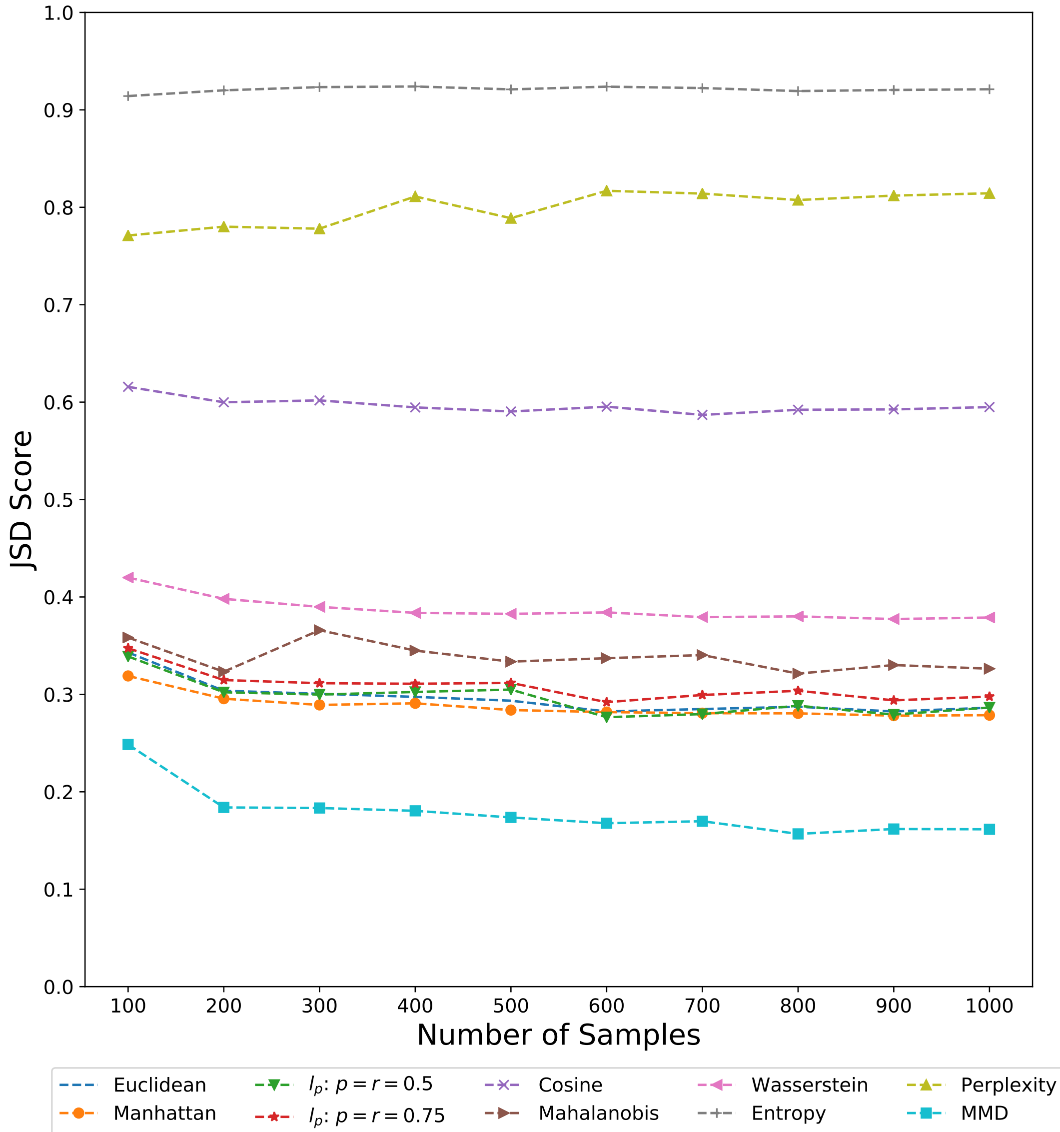

Figure 44: JSD results vs. number of samples for sample sizes between 100 and 1,000 on untransformed Network Events data. When the sample size only changes by 100 each time instead of going from 100 to 500 to 1,000, the JSD value forms a more smooth curve.

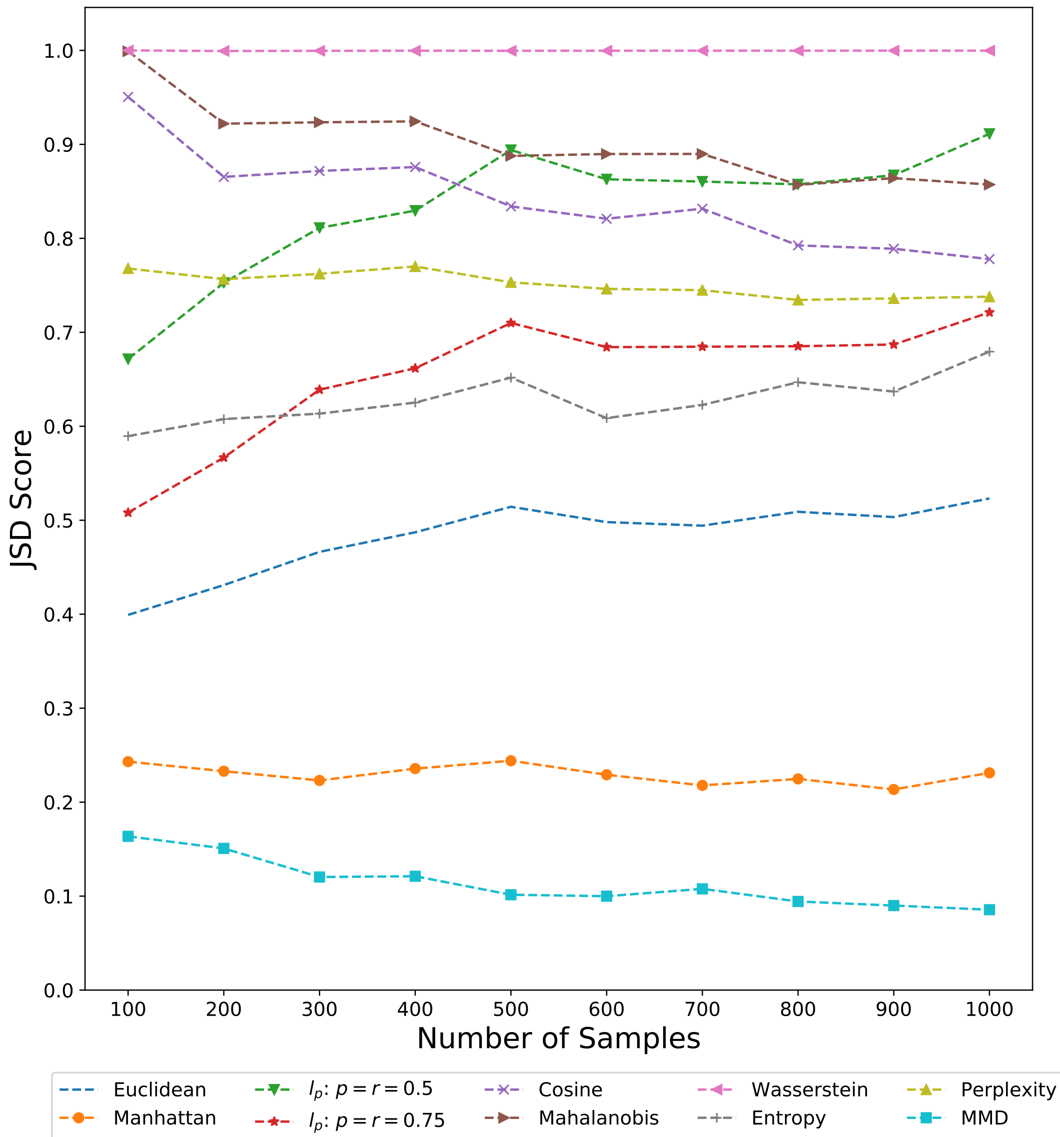

Figure 45: JSD results vs. number of samples for sample sizes between 100 and 1,000 on untransformed Host Events data. When the sample size only changes by 100 each time instead of going from 100 to 500 to 1,000, the JSD value forms a more smooth curve.

V. Conclusions

This chapter presents a summary of the research conducted and presents lessons learned and future work recommendations.

5.1 Research Summary

This research sought to answer three research questions (RQs):

RQ 1 What methods exist for measuring the "closeness" of real semi-structured sequential data to generated semi-structured sequential data?

Answer Based on literature review we found 11 metrics to be evaluated, detailed in Section 2.5

RQ 2 What characteristics should a potential metric possess?

Answer Based on the framework from [15], there are 4 characteristics: Discriminative Ability, Efficiency, Generative Failure Detection, and Overfitting Detection

RQ 3 Given metrics for comparing data and the characteristics we want, what metrics perform best for temporally ordered, semi-structured sequential data?

Answer There is no one-size-fits-all metric. However, Wasserstein distance, the fractional l_p distances, Entropy, and Perplexity provide good results

Section 2.5 presents many of the possible metrics that could be used while Section 3.6 presents the metrics chosen for evaluation. These metrics were chosen based on general use, network data applications, and prior use as Generative Adversarial Network (GAN) evaluation metrics. The metrics chosen are the following:

- Power distance (Equation (4)): Euclidean ($p = r = 2$), Manhattan ($p = r = 1$), fractional l_p distance ($p = r = 0.5$ and $p = r = 0.75$)
- Mahalanobis distance (Equation (5))
- Cosine similarity (Equation (14))
- Wasserstein Distance (Equation (11))
- Maximum Mean Discrepancy (MMD) (Equation (15))
- Fréchet Inception Distance (FID) (Equation (3))
- Entropy (Equation (8))
- Perplexity (Equation (10))

The characteristics that are desirable for a metric to possess are detailed in Section 3.7. The desired characteristics are the following: discriminative ability, efficiency, generative failure detection, and overfitting detection.

The third research question experimentally seeks to combine the first two research questions by evaluating each of the metrics chosen on each of the desired characteristics. This research experiments with discriminative ability and efficiency. Experiment methodology for generative failure detection and overfitting detection are detailed here and implementation of these experiments is left as future work.

5.1.1 Discriminative Ability

The results of the discriminative ability experiment on both datasets are quite informative. There are three main outcomes of this experiment:

- The type of data being evaluated impacts the ability to distinguish between real and generated data as demonstrated by the difference in results between Network Events and Host Events data

- The choice of feature space is vitally important. The overall ability to discriminate between real and generated semi-structured sequential data hinges on the feature space that the distances are calculated in

- Given a suitable feature space, the choice of metric significantly affects the sample distance distributions

On both datasets, the choice of feature space is crucial to the overall discriminative performance. For Jensen-Shannon Distance (JSD) the threshold at which the score is "discriminative" is based on visual correspondence with the distributions being visually different with little overlap. For this work, a JSD score of 0.5 seems to correspond with a visibly noticeable difference in distributions.

On the Network Events dataset, we see a drastic increase in overall JSD score for all metrics from the untransformed (or any other) space to the Principal Component Analysis (PCA) space with 3 or less of 11 metrics with a JSD of 0.5 or greater to 8 of 11 metrics with a JSD of 0.5 or greater. Conversely, we see a drastic decrease in performance from all spaces on the Network Events data to the Fast Fourier Transform (FFT) feature space with all metrics having less than a 0.4 JSD score.

We also see that the choice of metric can affect the sample distance distributions. For example, Examining the MMD in most of the feature spaces on the Host Events data, MMD shows little to no difference in the sample distance distributions (JSD$<$ 0.3). However, in the PCA and FFT space, the JSD score improves drastically (0.5-0.6 range) showing a visible difference in the sample distributions.

As these two outcomes suggest, there is no one-size-fits-all metric that emerges from this research. However, examining the results we can see that there are a couple of metrics that perform well in most suitable feature spaces. Wasserstein distance performs well in the majority of the feature spaces. Entropy, Perplexity, and the fractional l_p-distances also perform well in many of the feature spaces. This suggests that these metrics may generalize well to other feature spaces for future work.

5.1.2 Efficiency

For the efficiency characteristic, time efficiency and sample efficiency are explored.

The results of time efficiency are straightforward. The metrics with higher run-times at higher sample lengths correspond to metrics that have higher algorithmic complexity in either calculation or implementation.

The sample efficiency results did not come out as expected following the methodology from [15]. Instead of seeing an increase in JSD score as was expected, the JSD tended to stay relatively constant in most cases, with some positive and negative changes in others. However, no metric's JSD went from a low level to a high level or vice versa. One interesting outcome of the sample efficiency experiment is that it confirmed the experimental use of 1,000 samples as a good representation of JSD scores at larger numbers of samples. A number of samples equal to 1,000 samples produced the best combination of mean JSD score and metric calculation time.

5.2 Future Work

In this section, we detail future work that can be done to improve on this research moving forward. The list below summarizes items that should be explored in future research.

- Develop GAN and apply evaluation framework in model feature space
- Parallelize metric computation code and utilize Graphics Processing Unit (GPU)
- Evaluate framework on other datasets such as CICIDS 2017 [2] or Snort Logs [61]
- Evaluate metrics on generative failure detection and overfitting detection

Although generative failure detection and overfitting detection were left as future work, we lay out an experiment methodology here. The ideas for these experiments are based on the experiments from [15].

5.2.1 Generative Failure Detection

Generative failure in the form of mode collapsing and mode dropping is a common problem for generative models and especially GANs. A complete explanation of these failures is provided in Section 2.2.1. Our approaches for testing for these generative failures are the same as the ones described in [15].

To detect mode collapsing, we can sample two disjoint sets of real samples S_r and S'_r. We can then find a certain number of clusters in one of the sets and then progressively replace each cluster with its center and measure $d(S_r, S'_r)$ as we replace more and more clusters. Ideally, as the number of clusters replaced increases, the scores will increase.

To detect mode dropping, we take S_r as before and construct S'_r by randomly removing clusters. Samples that are removed are then replaced with samples randomly selected from the remaining clusters. As with mode collapsing, ideally $d(S_r, S'_r)$ should increase as more and more clusters are dropped.

5.2.2 Overfitting Detection

Overfitting is a possibility when utilizing a finite training set. Assuming that the generator is trained on a training set of real data, we can use the validation set approach typical of other Machine Learning (ML) applications to test for overfitting. To simulate the overfitting process we use an approach that is the same as the approach in [15]. This process is the following: We construct a set of samples, S'_r that is a mix of samples from the training set S_r^{tr} and a second validation set S_r^{val} with the overlap fraction between S_r^{tr} and S_r^{val} as a parameter. We then increase the fraction of the set that is made up of training set examples and track the value of $d(S_r', S_r^{val})$. If we expect overfitting, we can assume that the maximum score would be achieved when the overlap fraction of S_r' is 0. Thus we can then normalize the score of each metric by this value to reflect the increase. Ideally the metric scores should increase as the second set increasingly overlaps with the training set.

5.3 Contributions

There are three main contributions from this work. First, this work provides the first known framework for evaluating metrics for semi-structured sequential synthetic data generation based on a framework for evaluating metrics for image generation. Second, this work provides a "black box" evaluation framework which is generator agnostic, meaning that it has broad applicability. Third, this research provides the first known evaluation of metrics for semi-structured sequential data generation.

5.4 Summary

There is still much work to be done in the area of GAN metric evaluation. Hopefully future work will continue to improve the ability of GANs to generate synthetic semi-structured sequential data.

Appendix A. User Guide

This section outlines the steps necessary to reproduce the experiments described in Chapter III and Chapter IV.

1.1 System Configuration

- Computer with Python installed, preferably through Anaconda
- Jupyter Notebooks or Jupyter Lab (comes with Anaconda)
- Ensure `NumPy`, `SciPy`, `Pandas`, and `Matplotlib` are installed to your Python or Anaconda environment

1.2 Dataset Preparation

- Download the dataset from `https://csr.lanl.gov/data/2017.html`, or obtain from the repository
- Network Events files have the format: `netflow_day-XX.bz2` where XX is the 2 digit day. This research uses the `netflow_day-02.bz2` file
- Host Events files have the format `wls_day-XX.bz2` where XX is the 2 digit day. This research uses the `wls_day-02.bz2` file
- Extract the zip file desired. The Network Events file will have the format `netflow_day-XX` with no extension. The Host Events file will have the format `wls_day-XX.json` format.

1.3 Data Generation

1.3.1 Real Samples

Follow these steps to create real samples for the Host Events and Network Events dataset

- Using the extracted file, `create_real_samples.py` contains the necessary code for generating the real data samples. This file contains several variables that will need to be changed based on actual file locations. Set the variable `original_host_file` to the actual name of the Host Events file that you want to use. Set the `original_netflow_file` to desired Network Events file.

- For Network Events data, run the function `create_real_samples()` with specified arguments for how many samples to create and how many lines the samples should contain. The created samples will be CSV files that have the format `netflow_day-XX_sample_i.txt`

- For Host Events data, run the function `create_real_host_samples()` with specified arguments for how many samples to create and how many lines the samples should contain. The created samples will have the format `wls_day-XX.json_sample_i.txt`

- Host Events real samples require an additional processing step using Term Frequency - Inverse Document Frequency (TF-IDF). To perform this, use `create_real_tfidf_samples.py`. Set the appropriate `*_dir` variables to reflect data file locations.

- To create the processed TF-IDF samples, run the function `create_host_samples()` with the `num_samples` argument set to whatever number of samples is desired. 1,000 line CSV files will be output with the format `tf_idf_sample_i.txt`

1.3.2 Fake Samples

To create fake samples, use the code within `random_generator_sample.py`. Real samples for the desired dataset (Host or Network Events) must have been created to generate fake samples

- To create real Network Events samples, run the function `real_data_global_max()`. This will iterate through the entire repository of real samples and track the global minimum and maximum for each feature. This function will output a CSV file called `real_data_maxes.csv`

- To create real Host Events samples, run the function `host_data_global_values()`. This will iterate through the entire repository of real Host samples and track the global minimum, maximum, mean, and standard deviation for each feature. This function will output a CSV file called `real_host_data_maxes.csv`

- To generate fake Network Events samples, ensure that `real_data_maxes.csv` has been created. Copy the values from `real_data_maxes.csv` into the respective `REAL_MAXES` or `REAL_MINS` variable. Run the function `generate_random_samples()` with the desired number of samples and sample length as arguments to generate the desired fake samples.

- To generate fake Host Events samples, ensure that `real_host_data_maxes.csv` has been created. Within `generate_random_host_samples()` ensure that the `infile` variable points to the location of `real_host_data_maxes.csv`. Run `generate_random_host_samples()` with the desired number of samples and sample length as arguments to create fake Host samples. 2 different datasets will be generated by this function, located in folders with the distribution name. The first dataset will be samples generated from a Uniform random distribution based on the global minimum and maximum for each feature. The second dataset will be samples generated from a Normal random distribution based on the global mean and standard deviation for each feature.

1.4 Discriminative Ability Experiment - Network Events Dataset

With the real and fake Network Events sample repositories generated, we can run the discriminative ability experiments. For our work, we created 10,000 samples in each repository, however, this number can change. 10,000 samples is a good number since the experiment pulls 1,000 samples 10 times, thus ensuring that if necessary, 10 disjoint sets of samples can be pulled.

- To run the experiments, use `disc_experiments.py`. Modify any of the global variables to fit the desired values. Default values are the values which this research used.

- Running the code in `disc_experiments.py` will generate 5 different folders (one for each transform) labelled `untrans, sqrt, log, pca, fft`.Each folder contains 2 files: `real_data_exp.csv` and `fake_data_exp.csv`

- With the above folders and files generated, run the code in `thesis_project_-disc_v2.ipynb` to conduct the analysis and generate plots. Make sure to change any directory references to fit your directory structure.

- Running all cells of the Jupyter Notebook (`.ipynb` file) will generate results files and figures.

  - Histogram Figures will be located in `figures/` generated in `.pdf` and `.png` and have the format `hist_mat_vert_1000_1000` and will be located in a directory with the appropriate transform (one of the 5 listed above).

  – Results will be in 2 different files in the `results/` directory. For each transform, there will be a `JSD_results_TRANSFORM NAME HERE.csv` and `JSD_results_TRANSFORM NAME HERE_stats.csv` file. The results file contains the raw JSD scores for each of the 10 runs. The stats file contains the mean, min, max, and range of JSD scores for each metric ordered by decreasing mean JSD.

  – Box-and-Whisker plots of each of the metrics JSD scores in each transform will also be generated and have the format `box_whisker_network_-TRANSFORM NAME HERE.pdf` and `box_whisker_network_TRANSFORM NAME HERE_zoomed.pdf`

1.5 Discriminative Ability Experiment - Host Events Dataset

With the real and fake Host Events sample repositories generated, we can run the discriminative ability experiments on the Host Events dataset. For our work, we created 10,000 samples in each repository for each distribution, however, this number can change. 10,000 samples is a good number since the experiment pulls 1,000 samples 10 times, thus ensuring that if necessary, 10 disjoint sets of samples can be pulled.

- To run the experiments, use `disc_experiments_host.py`. Modify any of the global variables to fit the desired values. Default values are the values which this research used.

- Running the code in `disc_experiments_host.py` will generate 5 different folders (one for each transform) labelled `untrans, sqrt, log, pca, fft`.Each folder contains 4 files: `real_data_exp_host_uniform.csv, real_data_exp_-`

host_normal.csv, fake_data_exp_host_uniform.csv and fake_data_exp_host_-normal.csv

- With the above folders and files generated, run the code in thesis_project_-disc_v2_host.ipynb to conduct the analysis and generate plots. Make sure to change any directory references to fit your directory structure.

- Running all cells of the Jupyter Notebook (.ipynb file) will generate results files and figures

- The default is for all of the results to use the uniform distribution, so all files will have the uniform extension in the name. To get normal results, change the input file to have the normal extension instead of uniform and change all figures and results from uniform to normal. The following instructions use uniform as the distribution name

  – Histogram Figures will be located in figures/ generated in .pdf and .png and have the format hist_mat_vert_1000_1000 and will be located in a directory with the appropriate transform (one of the 5 listed above) and the sub-directory of the distribution (uniform or normal).

  – Results will be in 2 different files in the results/ directory. For each transform, there will be a JSD_results_host_uniform_TRANSFORM NAME HERE.csv and JSD_results_host_uniform_TRANSFORM NAME HERE_stats.csv file. The results file contains the raw JSD scores for each of the 10 runs. The stats file contains the mean, min, max, and range of JSD scores for each metric ordered by decreasing mean JSD.

  – Box-and-Whisker plots of each of the metrics JSD scores in each transform will also be generated and have the format box_whisker_host_uniform_-

TRANSFORM NAME HERE.pdf and box_whisker_host_uniform_TRANSFORM NAME HERE_zoomed.pdf

1.6 Efficiency Experiment - Host Events Dataset

With the real and fake Network Events samples generated, the following steps detail how to re-create the Efficiency experiments.

- Run the code in efficiency_experiments.py to generate the results for the efficiency experiment. Results will be located in 5 folders, one for each transform. Within each folder, the files will have the format real_data_exp_eff_{SAMPLE SIZE}.csv and fake_data_exp_eff_{SAMPLE SIZE}.csv. SAMPLE SIZE will be 100, 500, 1,000, 1,500,...,5,000. NOTE: This will take a long time (several days)

- Run the cells in thesis_project_efficiency.ipynb to generate the figures for the efficiency experiments. Ensure that the directory with the results file is pointed to within the code

- The time efficiency experiment will output a figure in the figures/ directory with the name time_efficiency.eps and will output the results in a CSV file names time_efficiency_results.csv

- The rest of the efficiency results will be figures named sample_efficiency_-TRANSFORM_jsd.pdf

1.7 Efficiency Experiment - Host Events Dataset

With the real and fake Host Events samples generated, the following steps detail how to re-create the Efficiency experiments. These instructions (and this research) use the uniform samples, but normal results can be created/used by replacing "uniform" with "normal" in all file names or commands.

- Run the code in `efficiency_experiments_host.py` to generate the results for the efficiency experiment. Results will be located in 5 folders, one for each transform. Within each folder, the files will have the format `real_data_exp_eff_host_uniform_{SAMPLE SIZE}.csv` and `fake_data_exp_eff_host_uniform_{SAMPLE SIZE}.csv`. SAMPLE SIZE will be 100, 500, 1,000, 1,500,...,5,000. NOTE: This will take a long time (several days)

- Run the cells in `thesis_project_efficiency_host.ipynb` to generate the figures for the efficiency experiments. Ensure that the directory with the results file is pointed to within the code

- The time efficiency experiment will output a figure in the `figures/` directory with the name `time_efficiency_host.eps` and will output the results in a CSV file names `time_efficiency_host_results.csv`

- The rest of the efficiency results will be figures named `sample_efficiency_host_TRANSFORM_jsd.pdf`

Appendix B. Metric Calculation Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 12 13:03:47 2019

@author: mnewlin
"""

import numpy as np
import pandas as pd
import scipy
from scipy import stats
from scipy.linalg import sqrtm

from scipy.spatial.distance import pdist
from scipy.spatial.distance import cosine
from scipy.spatial.distance import cdist
#from scipy.spatial.distance import jensonshannon as js

from scipy.stats import wasserstein_distance as wasserstein
from scipy.special import rel_entr

from scipy.stats import norm, entropy
from scipy.stats.mstats import gmean
import time


"""
    Generates probabilities for matrices X and Y, assuming given
   distribution
```

```
    distribution defaults to normal (may add other distributions
   later)

"""
def generate_probs(X,Y, dist='norm'):
    X = np.nan_to_num(np.array(X))
    Y = np.nan_to_num(np.array(Y))
    num_rows = X.shape[0]
    num_cols = X.shape[1]
    norm_x = np.zeros((num_rows, num_cols))
    norm_y = np.zeros((num_rows, num_cols))
    if dist == 'norm':
        for j in range(num_cols):
            xj = X[:,j]

            prob_xj = norm.pdf(xj, loc=xj.mean(), scale=xj.var())
            norm_x[:,j] = prob_xj

            yj = Y[:,j]

            prob_yj = norm.pdf(yj, loc=yj.mean(), scale=yj.var())
            norm_y[:,j] = prob_yj
    norm_x = np.nan_to_num(norm_x)
    norm_y = np.nan_to_num(norm_y)
    return norm_x, norm_y

"""
    Calculates the Power distance between two matrices X and Y
    Defaults to Euclidean Distance unless parameters p and r are
   provided
"""
def l_p_distance(X,Y,p=2,r=2):
```

```
    X = np.array(X)
    Y = np.array(Y)
    if (X.shape != Y.shape):
        print("Usage: Matrices must be the same shape.")
        return -1
    num_cols = X.shape[1]
    distances = np.zeros((num_cols,1))
    for i in range(num_cols):
        x = X[:,i]
        y = Y[:,i]
        diff = np.abs(x-y)
        distances[i] = np.power(np.sum(np.power(diff,p)),(1/r))

    return np.mean(np.nan_to_num(distances))

"""
    Calculates the cosine similarity between two matrices X and Y
    0 --> X and Y are the same
    1 --> X and Y are orthogonal
"""
def cosine_similarity(X,Y):
    X = np.array(X)
    Y = np.array(Y)
    num_cols = X.shape[1]
    cos_sims = np.array([])
    for i in range(num_cols):
        cos_sim = cosine(X[:,i], Y[:,i])
        cos_sims = np.append(cos_sims, cos_sim)
        cos_sims = np.nan_to_num(cos_sims)
        cos_sims = np.where(cos_sims > 1, 1, cos_sims)
    return np.mean(cos_sims)

```

```
"""
    Calculates the Mahalanobis distance between 2 matrices X and Y
"""
def mahalanobis_distance(X,Y):
    X = np.array(X)
    Y = np.array(Y)
    stack = np.vstack([X, Y])
    VI = np.linalg.pinv(np.cov(stack, rowvar=False))
    d_mat = cdist(X,Y, metric='mahalanobis', VI=VI)
    return np.trace(d_mat)

def alt_mahalanobis(X,Y):
    X = np.array(X)
    Y = np.array(Y)
    prob_x, prob_y = generate_probs(X,Y)
    # Generate Positive Definite Matrix
    #XXT = np.matmul(X.T,X)
    #YYT = np.matmul(Y.T,Y)
    #stack = np.vstack([XXT, YYT])
    #VI = np.linalg.pinv(np.cov(stack, rowvar=False))
    #mahalanobis = cdist(XXT, YYT, 'mahalanobis', VI=VI)
    stack = np.vstack([prob_x, prob_y])
    VI = np.linalg.pinv(np.cov(stack, rowvar=False))
    mahalanobis = cdist(prob_x, prob_y, 'mahalanobis', VI=VI)
    return np.mean(np.nan_to_num(mahalanobis))

"""
    Calculates the chi squared distance between 2 matrices X and Y
    This function relies on the generate_probs function to generate
    probabilities for the values of the matrices X and Y in order to
     calculate
    the chi-squared distance.
```

```
"""
def chi_squared_dist(X,Y):
    X = np.array(X)
    Y = np.array(Y)
    num_cols = X.shape[1]
    prob_x, prob_y = generate_probs(X,Y)
    chi_squares = np.array([])
    for j in range(num_cols):
        prob_yj = prob_y[:,j]
        epsilon = 1e-6
        prob_yj = np.where(prob_yj == 0, epsilon, prob_yj)
        chi_squares = np.append(chi_squares, np.sum(np.divide(np.square(prob_x[:,j]-prob_y[:,j]), prob_yj)))
    return np.mean(np.nan_to_num(chi_squares))
"""
    Calculate the Wasserstein Distance between Matrices X and Y
"""

def KL(P,Q, eps=1e-5):
    """ Epsilon is used here to avoid conditional code for
    checking that neither P nor Q is equal to 0. """
    epsilon = eps

    # You may want to instead make copies to avoid changing the np arrays.
    P_prime = np.where(P==0, P+epsilon, P)
    Q_prime = np.where(Q==0, Q+epsilon, Q)


    divergence = np.sum(np.multiply(P_prime,np.log(P_prime/Q_prime)))
    return divergence
```

```

"""
    Function for scipy jenson shannon divergence
    https://scipy.github.io/devdocs/generated/scipy.spatial.distance.jensenshannon.html
    As of writing this file, this is still in a dev version of scipy so
    this function was copied out of scipy source github at
    https://github.com/scipy/scipy/blob/089e3b2/scipy/spatial/distance.py#L1235-L1292

    Original code has base=None but I use base=2 so that JSD bounded between 0 and 1
"""
def jensenshannon(p, q, base=2):
    """
    Compute the Jensen-Shannon distance (metric) between
    two 1-D probability arrays. This is the square root
    of the Jensen-Shannon divergence.
    The Jensen-Shannon distance between two probability
    vectors `p` and `q` is defined as,
    .. math::
       \\sqrt{\\frac{D(p \\parallel m) + D(q \\parallel m)}{2}}
    where :math:`m` is the pointwise mean of :math:`p` and :math:`q`
    and :math:`D` is the Kullback-Leibler divergence.
    This routine will normalize `p` and `q` if they don't sum to 1.0.
    Parameters
    ----------
    p : (N,) array_like
        left probability vector
    q : (N,) array_like
```

```
            right probability vector
    base : double, optional
            the base of the logarithm used to compute the output
            if not given, then the routine uses the default base of
            scipy.stats.entropy.
    Returns
    -------
    js : double
            The Jensen-Shannon distance between `p` and `q`
    .. versionadded:: 1.2.0
    Examples
    --------
    >>> from scipy.spatial import distance
    >>> distance.jensenshannon([1.0, 0.0, 0.0], [0.0, 1.0, 0.0],
   2.0)
    1.0
    >>> distance.jensenshannon([1.0, 0.0], [0.5, 0.5])
    0.46450140402245893
    >>> distance.jensenshannon([1.0, 0.0, 0.0], [1.0, 0.0, 0.0])
    0.0
    """
    p = np.asarray(p)
    q = np.asarray(q)
    p = p / np.sum(p, axis=0)
    q = q / np.sum(q, axis=0)
    m = (p + q) / 2.0
    left = rel_entr(p, m)
    right = rel_entr(q, m)
    js = np.sum(left, axis=0) + np.sum(right, axis=0)
    if base is not None:
        js /= np.log(base)
    return np.sqrt(js / 2.0)
```

```


def wasserstein_dist(X,Y):
    X = np.array(X)
    Y = np.array(Y)
    #x_prob, y_prob = generate_probs(X,Y)
    num_cols = X.shape[1]
    wass_dists = np.array([])
    for x in range(num_cols):
        u = X[:, x]
        v = Y[:, x]
        d = wasserstein(u,v)
        wass_dists = np.append(wass_dists, d)
    return gmean(np.where(wass_dists==0,1,wass_dists))

"""
    Calculates the Difference in standardized entropy between two
    matrices X and Y
"""
def calc_entropy(X,Y, sample_length, standardized=True):
    sample_size = 1
    if standardized:
        sample_size = np.log(sample_length)
    X = np.array(X)
    Y = np.array(Y)
    norm_x,norm_y = generate_probs(X,Y)
    num_cols = X.shape[1]
    ents = np.array([])
    for j in range(num_cols):
        ent_x = np.nan_to_num(entropy(norm_x[:,j])/sample_size)
        ent_y = np.nan_to_num(entropy(norm_y[:,j])/sample_size)
        diff = np.abs(ent_x - ent_y)
```

```
        ents = np.append(ents, diff)
    return np.mean(np.nan_to_num(ents))

"""
    Calculates the Difference in perplexity between two matrices X
   and Y
"""
def calc_perplexity(X,Y, sample_length, standardized=True):
    sample_size = 1
    if standardized:
        sample_size = np.log(sample_length)
    X = np.array(X)
    Y = np.array(Y)
    norm_x,norm_y = generate_probs(X,Y)
    num_cols = X.shape[1]
    perps = np.array([])
    for j in range(num_cols):
        ent_x = np.nan_to_num(entropy(norm_x[:,j])/sample_size)
        ent_y = np.nan_to_num(entropy(norm_y[:,j])/sample_size)
        perp_x = np.power(2,ent_x)
        perp_y = np.power(2,ent_y)
        diff = np.abs(perp_x - perp_y)
        perps = np.append(perps, diff)
    return np.mean(np.nan_to_num(perps))

"""
    Calculates the Frechet Inception Distance between matrices X and
    Y
    Implementation details taken from
    https://machinelearningmastery.com/how-to-implement-the-frechet-
   inception-distance-fid-from-scratch/
"""
```

```python
def fid(X,Y):
    X = np.array(X)
    Y = np.array(Y)
    prob_x, prob_y = generate_probs(X,Y)
    mu_x = np.mean(prob_x, axis=0)
    mu_y = np.mean(prob_y, axis=0)

    Cx = np.cov(prob_x,rowvar=False)
    Cy = np.cov(prob_y, rowvar=False)
    ssdiff = np.sum(np.square(mu_x-mu_y))
    covmean = scipy.linalg.sqrtm(Cx.dot(Cy))
    score = ssdiff + np.trace(Cx + Cy - 2.0*covmean)
    return np.abs(score)
"""
    Calculates (X-Y)^2 for matrices X and Y
    Returns distance matrix M
"""
def distance(X,Y, sqrt=False):
    X = np.array(X)
    Y = np.array(Y)
    X2 = np.matmul(X,X.T)
    Y2 = np.matmul(Y,Y.T)
    XY = np.matmul(X,Y.T)
    M = X2+Y2-2*XY
    if sqrt:
        M = np.sqrt(np.abs(M))
    return M

"""
    Calculated the Maximum Mean Discrepancy between
    real and fake distributions using the Gaussian Kernel (RBF)
"""
```

```
def mmd(Mxx,Mxy, Myy, sigma):
    mu = np.mean(Mxx)
    Mxx = np.nan_to_num(np.exp(np.divide(-Mxx,mu*2*sigma*sigma)))
    Mxy = np.nan_to_num(np.exp(np.divide(-Mxy,mu*2*sigma*sigma)))
    Myy = np.nan_to_num(np.exp(np.divide(-Myy,mu*2*sigma*sigma)))
    a = Mxx.mean() + Myy.mean() - 2*Mxy.mean()
    mmd = np.sqrt(np.abs(a))
    return mmd

"""
    Calculates the Bhattacharyya distance between X and Y
"""
def bhattacharyya(X, Y):
    X = np.array(X)
    Y = np.array(Y)
    num_cols = X.shape[1]
    prob_x, prob_y = generate_probs(X,Y)
    dist = np.array([])
    for j in range(num_cols):
        x = prob_x[:,j]
        y = prob_y[:,j]
        bc = np.sum(np.sqrt(np.multiply(x,y)))
        bd = -np.log(bc)
        dist = np.append(dist, bd)
    return np.mean(np.nan_to_num(dist))



"""
    Score two sets of samples based on a given metric
"""
def score_set(S1, S2, sample_length, num_samples, metric='lp', p=2,
   r=2, standardized=True, G1=None, G2=None):
```

```
    dist_matrix = np.array([])
    if metric == 'lp':
        for x in range(num_samples):
            d = l_p_distance(S1[x], S2[x], p=p, r=r)
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'cosine':
        for x in range(num_samples):
            d = cosine_similarity(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'mahalanobis':
        for x in range(num_samples):
            d = mahalanobis_distance(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'chi_squared':
        for x in range(num_samples):
            d = chi_squared_dist(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'wasserstein':
        for x in range(num_samples):
            d = wasserstein_dist(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'fid':
        for x in range(num_samples):
            d = fid(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'entropy':
        for x in range(num_samples):
            d = calc_entropy(S1[x], S2[x], sample_length=
   sample_length, standardized=standardized)
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'perplexity':
        for x in range(num_samples):
```

```
            d = calc_perplexity(S1[x], S2[x], sample_length=sample_length, standardized=standardized)
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'bd':
        for x in range(num_samples):
            d = bhattacharyya(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
    elif metric == 'mmd':
        for x in range(num_samples):
            Mxx = distance(S1[x],S2[x], sqrt=True)
            Myy = distance(G1[x],G2[x], sqrt=True)
            Mxy = distance(S1[x], G1[x], sqrt=True)
            d = mmd(Mxx, Mxy, Myy, sigma=1)
            dist_matrix = np.append(dist_matrix, d)
    return np.mean(dist_matrix), np.std(dist_matrix), dist_matrix

"""
    Score two sets of samples based on a given metric
"""
def time_score_set(S1, S2, sample_length, num_samples, metric='lp', p=2, r=2, standardized=True, G1=None, G2=None):
    dist_matrix = np.array([])
    t_start = -1.0
    t_end = -1.0
    if metric == 'lp':
        t_start = time.time()
        for x in range(num_samples):
            d = l_p_distance(S1[x], S2[x], p=p, r=r)
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'cosine':
        t_start = time.time()
```

```
        for x in range(num_samples):
            d = cosine_similarity(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'mahalanobis':
        t_start = time.time()
        for x in range(num_samples):
            d = mahalanobis_distance(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'chi_squared':
        t_start = time.time()
        for x in range(num_samples):
            d = chi_squared_dist(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'wasserstein':
        t_start = time.time()
        for x in range(num_samples):
            d = wasserstein_dist(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'fid':
        t_start = time.time()
        for x in range(num_samples):
            d = fid(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'entropy':
        t_start = time.time()
        for x in range(num_samples):
            d = calc_entropy(S1[x], S2[x], sample_length=
```

```
sample_length, standardized=standardized)
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'perplexity':
        t_start = time.time()
        for x in range(num_samples):
            d = calc_perplexity(S1[x], S2[x], sample_length=
sample_length, standardized=standardized)
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'bd':
        t_start = time.time()
        for x in range(num_samples):
            d = bhattacharyya(S1[x], S2[x])
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    elif metric == 'mmd':
        t_start = time.time()
        for x in range(num_samples):
            Mxx = distance(S1[x],S2[x], sqrt=True)
            Myy = distance(G1[x],G2[x], sqrt=True)
            Mxy = distance(S1[x], G1[x], sqrt=True)
            d = mmd(Mxx, Mxy, Myy, sigma=1)
            dist_matrix = np.append(dist_matrix, d)
        t_end = time.time()
    t_diff = t_end - t_start
    return dist_matrix, t_diff
```

Bibliography

1. S. Abt and H. Baier, "A Plea for Utilising Synthetic Data when Performing Machine Learning Based Cyber-Security Experiments," in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop - AISec '14*. New York, New York, USA: ACM Press, 2014, pp. 37–45. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2666652.2666663

2. I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018, pp. 108–116. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html

3. M. Małowidzki, P. Berezi, and M. Mazur, "Network Intrusion Detection : Half a Kingdom for a Good Dataset," in *Proceedings of NATO STO SAS-139 Workshop*, 2015. [Online]. Available: https://www.wil.waw.pl/art_prac/2015/Network_Intrusion_Detection.pdf

4. G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, "UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs," *Computers & Security*, vol. 73, pp. 411–424, 2017. [Online]. Available: https://doi.org/10.1016/j.cose.2017.11.004

5. B. Ricks, P. Tague, and B. Thuraisingham, "Large-scale realistic network data generation on a budget," in *Proceedings - 2018 IEEE 19th International Conference on Information Reuse and Integration for Data Science, IRI 2018*, 2018, pp. 23–30. [Online]. Available: http://mews.sv.cmu.edu/research/emews/

6. M. Newlin, M. Reith, and M. Deyoung, “Synthetic Data Generation with Machine Learning for Network Intrusion Detection Systems,” in *European Conference on Information Warfare and Security, ECCWS*, vol. 2019-July, 2019, pp. 785–789.

7. D. Garcia Torres, “Generation of Synthetic Data with Generative Adversarial Networks,” Ph.D. dissertation, KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science (EECS), 2018. [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1331279&dswid=-2834

8. F. Maymí, S. Lathrop, F. Maymí, and S. Lathrop, “AI in Cyberspace: Beyond the Hype,” *Cyber Defense Review*, vol. Volume 3, pp. 71–81, 2018. [Online]. Available: https://cyberdefensereview.army.mil/CDR-Content/Articles/Article-View/Article/1716483/ai-in-cyberspace-beyond-the-hype/

9. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, New Jersey: Prentice Hall, 2010.

10. G. James, D. Witten, T. Hastie, and R. Tibshirani, “Statistical Learning,” in *An Introduction to Statistical Learning.* New York: Springer Science+Business Media, 2013, ch. 2, pp. 15–58.

11. ——, “Classification,” in *An Introduction to Statistical Learning.* New York: Springer Science+Business Media, 2013, ch. 4, pp. 127–173.

12. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Nets,” *ArXiv*, 2014. [Online]. Available: http://www.github.com/goodfeli/adversarial

13. "Generative Adversarial Network Architecture." [Online]. Available: https://www.researchgate.net/figure/Generative-Adversarial-Network-Architecture_fig1_321865166

14. S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang, "Generalization and equilibrium in generative adversarial nets (GANs)," in *34th International Conference on Machine Learning, ICML 2017*, vol. 1, 2017, pp. 322–349.

15. Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, and K. Weinberger, "An empirical study on evaluation metrics of generative adversarial networks," *ArXiv*, 6 2018. [Online]. Available: http://arxiv.org/abs/1806.07755

16. M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv*, 1 2017. [Online]. Available: https://arxiv.org/pdf/1701.07875.pdf

17. I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," *ArXiv*, 2017. [Online]. Available: http://arxiv.org/abs/1704.00028

18. L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient," in *Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017, pp. 2852–2858. [Online]. Available: http://arxiv.org/abs/1609.05473

19. J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob. Berkeley AI Research Labratory, UC Berkeley, 2017, pp. 2242–2251. [Online]. Available: https://arxiv.org/pdf/1703.10593.pdf

20. X. Liu, X. Kong, L. Liu, and K. Chiang, "TreeGAN: Syntax-Aware Sequence Generation with Generative Adversarial Networks," in *Proceedings - IEEE International Conference on Data Mining, ICDM*, vol. 2018-Novem, 2018, pp. 1140–1145. [Online]. Available: https://arxiv.org/pdf/1808.07582.pdf

21. C. Wang, C. Xu, X. Yao, and D. Tao, "Evolutionary Generative Adversarial Networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 6, pp. 921–934, 2019.

22. D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

23. P. Bachman and D. Precup, "Data Generation as Sequential Decision Making," in *Advances in Neural Information Processing Systems*, vol. 2015-Janua, 2015, pp. 3249–3257. [Online]. Available: https://arxiv.org/pdf/1506.03504.pdf

24. S. Lu, Y. Zhu, W. Zhang, J. Wang, and Y. Yu, "Neural Text Generation: Past, Present and Beyond," 2018. [Online]. Available: http://arxiv.org/abs/1803.07133

25. V. Hajdik, J. Buys, M. W. Goodman, and E. M. Bender, "Neural Text Generation from Rich Semantic Representations," 2019, pp. 2259–2266. [Online]. Available: http://svn.delph-in.net/erg/tags/

26. S. Tulyakov, M. Y. Liu, X. Yang, and J. Kautz, "MoCoGAN: Decomposing Motion and Content for Video Generation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 12 2018, pp. 1526–1535.

27. C. Yin, Y. Zhu, S. Liu, J. Fei, and H. Zhang, "An Enhancing Framework for Botnet Detection Using Generative Adversarial Networks," in *2018 International*

Conference on Artificial Intelligence and Big Data, ICAIBD 2018, 2018, pp. 228–234.

28. M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using Generative Adversarial Networks," *Computers and Security*, vol. 82, pp. 156–172, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818308393?via%3Dihub

29. L. Theis, A. Van Den Oord, and M. Bethge, "A Note on the Evaluation of Generative Models," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

30. T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," *arXiv*, 6 2016. [Online]. Available: http://arxiv.org/abs/1606.03498

31. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, 2016, pp. 2818–2826.

32. J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE conference on computer vision and pattern recognition*, 2010, pp. 248–255. [Online]. Available: https://www.researchgate.net/publication/221361415

33. M. Lucic, K. Kurach, Marcin Michalski, B. O. Bousquet, and S. Gelly, "Are GANs Created Equal? A Large-Scale Study," *ArXiV*, 2018. [Online]. Available: https://arxiv.org/pdf/1711.10337.pdf

34. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, “GANs trained by a two time-scale update rule converge to a local Nash equilibrium,” in *Advances in Neural Information Processing Systems*, vol. 2017-Decem, Long Beach, CA, 2017, pp. 6627–6638.

35. M. Deza and E. Deza, *Encyclopedia of distances.* Springer Berlin Heidelberg, 2009. [Online]. Available: http://link.springer.com/content/pdf/10.1007/978-3-642-00234-2_1.pdf

36. P. C. Mahalanobis, “On the generilised distance in statistics,” *Proceedings of the National Institute of Sciences of India*, vol. 2, no. 1, pp. 49–55, 1936.

37. D. J. Weller-Fahy, B. J. Borghetti, and A. A. Sodemann, “A Survey of Distance and Similarity Measures Used Within Network Intrusion Anomaly Detection,” *IEEE Communications Surveys and Tutorials*, vol. 17, no. 1, pp. 70–91, 2015.

38. K. Wang and S. J. Stolfo, “Anomalous Payload-Based Network Intrusion Detection,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3224, pp. 203–222, 2004.

39. C. E. Shannon, “A Mathematical Theory of Communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

40. Y. Wang, Z. Zhang, L. Guo, and S. Li, “Using entropy to classify traffic more deeply,” in *Proceedings - 6th IEEE International Conference on Networking, Architecture, and Storage, NAS 2011*, 2011, pp. 45–52.

41. P. Kawthekar, R. Rewari, and S. Bhooshan, “Evaluating Generative Models for Text Generation,” *arXiv*, pp. 1–8, 2017. [Online]. Available: https://web.stanford.edu/class/cs224n/reports/2737434.pdf

42. L. N. Vaserstein, “Markov Processes over Denumerable Products of Spaces, Describing Large Systems of Automata,” *Problemy Peredachi Informatsii*, vol. 5, no. 3, pp. 64–72, 1969. [Online]. Available: https://arxiv.org/pdf/1908.09899.pdf

43. S. Kullback and R. Leibler, “On Information and Sufficiency,” *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. [Online]. Available: https://projecteuclid.org/euclid.aoms/1177729694

44. J. Lin, “Divergence Measures Based on the Shannon Entropy,” *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 145–151, 1991.

45. D. M. Endres and J. E. Schindelin, “A new metric for probability distributions,” pp. 1858–1860, 2003.

46. A. Singhal, “Modern Information Retrieval: A Brief Overview,” *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 24, no. 4, pp. 35–43, 2001. [Online]. Available: http://trec.nist.gov

47. A. Gretton, K. M. Borgwardt, M. J. Rasch, A. Smola, B. Schölkopf, and A. Smola GRETTON, “A Kernel Two-Sample Test,” *Journal of Machine Learning Research*, vol. 13, no. Mar, pp. 723–773, 2012. [Online]. Available: http://jmlr.csail.mit.edu/papers/v13/gretton12a.html

48. M. Wurzenberger, F. Skopik, G. Settanni, and W. Scherrer, “Complex log file synthesis for rapid sandbox-benchmarking of security- and computer network analysis tools,” *Information Systems*, vol. 60, no. C, pp. 13–33, 8 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S030643791530212X

49. V. Kulkarni and B. Garbinato, “Generating synthetic mobility traffic using RNNs,” in *Proceedings of the 1st Workshop on Artificial Intelligence and Deep Learning for Geographic Knowledge Discovery - GeoAI ’17*. New

York, New York, USA: ACM Press, 2017, pp. 1–4. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3149808.3149809

50. M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

51. S. Semeniuta, A. Severyn, and S. Gelly, "On Accurate Evaluation of GANs for Language Generation," 2018. [Online]. Available: http://arxiv.org/abs/1806.04936

52. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, 2002, pp. 311–318. [Online]. Available: https://www.aclweb.org/anthology/P02-1040

53. R. Wirth, "CRISP-DM : Towards a Standard Process Model for Data Mining," in *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, 2000, pp. 29–39.

54. "CRISP-DM Process Diagram." [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/b/b9/CRISP-DM_Process_Diagram.png

55. M. J. M. Turcotte, A. D. Kent, and C. Hash, "Unified Host and Network Data Set," in *Data Science for Cyber-Security*. World Scientific, 11 2018, ch. 1, pp. 1–22. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/9781786345646_001

56. P. Buneman, "Semistructured data," in *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems - PODS*, 1997, pp. 117–121. [Online]. Available: http://www.cis.upenn.edu/~db.

57. Scikit-Learn, “Category Encoders,” 2016. [Online]. Available: http://contrib.scikit-learn.org/categorical-encoding/index.html

58. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/stable/index.html

59. F. Pedregosa, “Scikit-Learn RobustScaler,” 2016. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

60. A. Rajaraman and J. D. Ullman, “Data Mining,” in *Mining of Massive Datasets*. Cambridge University Press, 2005, ch. 1, pp. 1–19. [Online]. Available: https://www.cambridge.org/core/product/identifier/CBO9781139058452A007/type/book_part

61. M. Roesch, “Snort-Lightweight Intrusion Detection for Networks,” in *Proceedings of LISA ’99*, 1999, pp. 229–238. [Online]. Available: https://static.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf

Acronyms

AI Artificial Intelligence. 5

ANN Artificial Neural Network. 5

BLEU Bilingual Language Evaluation Understudy. 11, 20

CFG Context-Free Grammar. 9

CNN Convolutional Neural Network. 5

CRISP-DM Cross-Industry Standard Process for Data Mining. ix, 23

DL Deep Learning. 1, 2, 5, 6, 32

FFT Fast Fourier Transform. ix, x, xi, xii, 32, 42, 43, 60, 61, 63, 82, 85, 97

FID Fréchet Inception Distance. 13, 14, 20, 22, 31, 32, 42, 79, 96

GPU Graphics Processing Unit. 98

IDF Inverse Document Frequency. 29, 30

IDS Intrusion Detection System. 1, 2, 18

JSD Jensen-Shannon Distance. ix, x, xi, xii, 17, 33, 34, 36, 38, 39, 40, 41, 42, 43, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 82, 85, 82, 85, 82, 85, 87, 85, 87, 85, 89, 97, 98, 105, 106

JSON JavaScript Object Notation. ix, 25, 27

KLD Kullback-Leibler Divergence. 17, 34

MC Monte Carlo. 8

ML Machine Learning. iv, 1, 2, 5, 28, 99

MLE Maximum Likelihood Estimation. 11

MMD Maximum Mean Discrepancy. 18, 20, 22, 31, 32, 58, 59, 60, 63, 79, 96, 97

PCA Principal Component Analysis. ix, x, xi, xii, 32, 41, 42, 43, 58, 60, 63, 82, 85, 97

PMF Probability Mass Function. 33

R-F Real-Fake. 33, 36, 38, 39, 40, 42, 43, 54, 58, 60, 63, 85, 87, 85

R-R Real-Real. 33, 36, 38, 39, 40, 42, 43, 54, 58, 60, 63, 85, 87, 85

RNN Recurrent Neural Network. 5, 11, 19, 20

SeqGAN Sequence GAN. ix, 8, 9

SQRT Square Root. ix, x, xi, xii, 32, 39, 40, 43, 55, 57, 58, 63, 82, 85

TCP Transmission Control Protocol. 11, 12, 25

TF Term Frequency. 29, 30

TF-IDF Term Frequency - Inverse Document Frequency. 29, 30, 79, 102

UDP User Datagram Protocol. 12, 25

UHNDS Unified Host and Network Data Set. ix, xii, 25, 28

VAE Variational Auto-Encoder. 10

WGAN Wasserstein GAN. 7, 8, 16, 19, 134

WGAN-GP Wasserstein GAN (WGAN) - Gradient Penalty. 8, 12, 16

XML Extensible Markup Language. 25

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2020	Master's Thesis	Oct 2018 — Mar 2020

4. TITLE AND SUBTITLE

Quantitative Analysis of Evaluation Criteria for Generative Models

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Marvin W. Newlin, 2d Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-20-M-048

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

Intentionally Left Blank

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The goal of this research is to provide a framework that can be used to inform and improve the process of generating synthetic semi-structured sequential data. A series of experiments evaluating a chosen set of metrics on discriminative ability and efficiency is performed. This research shows that the choice of feature space in which distances are calculated in is critical. The ability to discriminate between real and generated data hinges on the space that the distances are calculated in. Additionally, the choice of metric significantly affects the sample distance distributions in a suitable feature space. There are three main contributions from this work. First, this work provides the first known framework for evaluating metrics for semi-structured sequential synthetic data generation. Second, this work provides a "black box" evaluation framework which is generator agnostic. Third, this research provides the first known evaluation of metrics for semi-structured sequential data.

15. SUBJECT TERMS

Generative Models, Evaluation Metrics, Machine Learning, Generative Adversarial Networks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lt Col Mark E. DeYoung, AFIT/ENG
U	U	U	UU	149	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636; mark.deyoung@afit.edu