Identifying Informative System Metrics: Predicting the predictability of time series using entropy

MSc. Thesis Computer Science

Pradyot Patil

Identifying Informative System Metrics: Predicting the predictability of time series using entropy

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Pradyot Patil
born in Akola, India

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

AI for Fintech Research
ING Bank N.V.
Frankemaheerd 1
Amsterdam, the Netherlands
www.ing.nl

Identifying Informative System Metrics: Predicting the predictability of time series using entropy

Author:	Pradyot Patil
Student id:	5051592
Email:	`p.s.patil@student.tudelft.nl`

Abstract

Building predictive models using cloud metrics for a task like incident prediction in the cloud is becoming ubiquitous in cloud monitoring. For such a forecasting task, if we know beforehand which system metrics are predictable then we can easily build good models. Quantifying the predictability of cloud metrics can help us rank the available system metrics and select a subset of cloud metrics with the lowest complexity. Moreover, storing informative metrics for a longer period can result in better forecasting.

This thesis presents a novel entropy method for quantifying the complexity of time series: Reverse weighted Dispersion Entropy (RWDE). We also present an exploratory study to understand and quantify the complexity of cloud metrics. This exploratory case study has been carried out at ING, a large banking company with in-house cloud architecture. We perform simulation experiments on simulated signals to compare RWDE with other entropy methods. We apply RWDE on cloud metric data from ING to approximate the predictability of these cloud metrics. The experimental results show that RWDE has better performance than other entropy methods and can be used to select informative cloud metrics for a forecasting task.

Further, we establish a relationship between RWDE and model-based predictability of cloud metrics. For each cloud metric, we compare RWDE with predictions from various forecasting models. Our results show that this relationship can be used as a heuristic by practitioners to identify unsuitable forecasting models for certain cloud metrics. We make RWDE and other entropy methods discussed in this study available as an open-source Python package.

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Dr. G. Gousios, Faculty EEMCS, TU Delft
Company supervisor:	Dr. J. Brons, ING
Committee Member:	Dr. J. Rellermeyer, Faculty EEMCS, TU Delft

Preface

I would like to thank my supervisor Georgios Gousios for his invaluable guidance and great discussions during this thesis project. Special thanks to Arie van Deursen for accepting me as a mentee halfway through my thesis. His valuable feedback helped me immensely in completing my thesis. A wonderful experience to work with both of you.

I would like to thank Elvan Kula, Jerry Brons, and Pinar Kahraman for giving me the opportunity to pursue my thesis project at ING. I would like to thank all my amazing colleagues at ING and AI for Fintech Research for all the important feedback they gave me during my research.

Lastly, I would like to thank my family and friends for constantly keeping me motivated and enthusiastic during my studies. I especially want to thank my father for always inspiring and unconditionally supporting me to become a computer scientist.

Pradyot Patil
Delft, the Netherlands
March 8, 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Lately, the software industry has witnessed a shift from stand-alone local software to online services. One such example is the shift from stand-alone Microsoft Office to Microsoft Office 365. We can also witness this trend in the banking and e-commerce industry, Which invest heavily in online banking systems and online services. To that end, Organizations need to set up (or hire) sufficient Information Technology (IT) infrastructure to provide a large-scale service online. This results in a typical IT infrastructure setup for an online service shown in the Figure 1.1 [51].

The important characteristic of online services is that they aim to run continuously with 24x7 availability. However, IT incidents that occur at the site lead to unscheduled system downtime, unplanned costs, and cause a huge impact since the recovery process can require time and resources that may not have been considered. Service outages could lead to judicial consequences and financial loss for an organization. For example, in 2018, the estimated cost of one-hour downtime for amazon.com on a sale day was $72 million [72].

Due to the rising complexity of online services and flaws in software and hardware, cloud systems are prone to failures and decreased reliability. To ensure high availability of online services, DevOps [21] and site reliability engineering practices are deployed. However, manual inspection of complex services and infrastructure can become a tedious job. For tackling this, Artificial Intelligence for IT Operations (AIOps) is an emerging discipline that uses artificial intelligence tools for automating operational tasks such as system monitoring, anomaly detection, root cause analysis, and recovery [11]. Dang et al. [18] defined AIOps as tools that empower software and service engineers to efficiently and effectively build and operate services that are easy to support and maintain by using artificial intelligence and machine learning techniques. If administrators can predict IT incidents using AIOps, they can better allocate their resources and services to mitigate the financial costs. Successfully predicting the future states of IT systems that are complex, stochastic, and po-

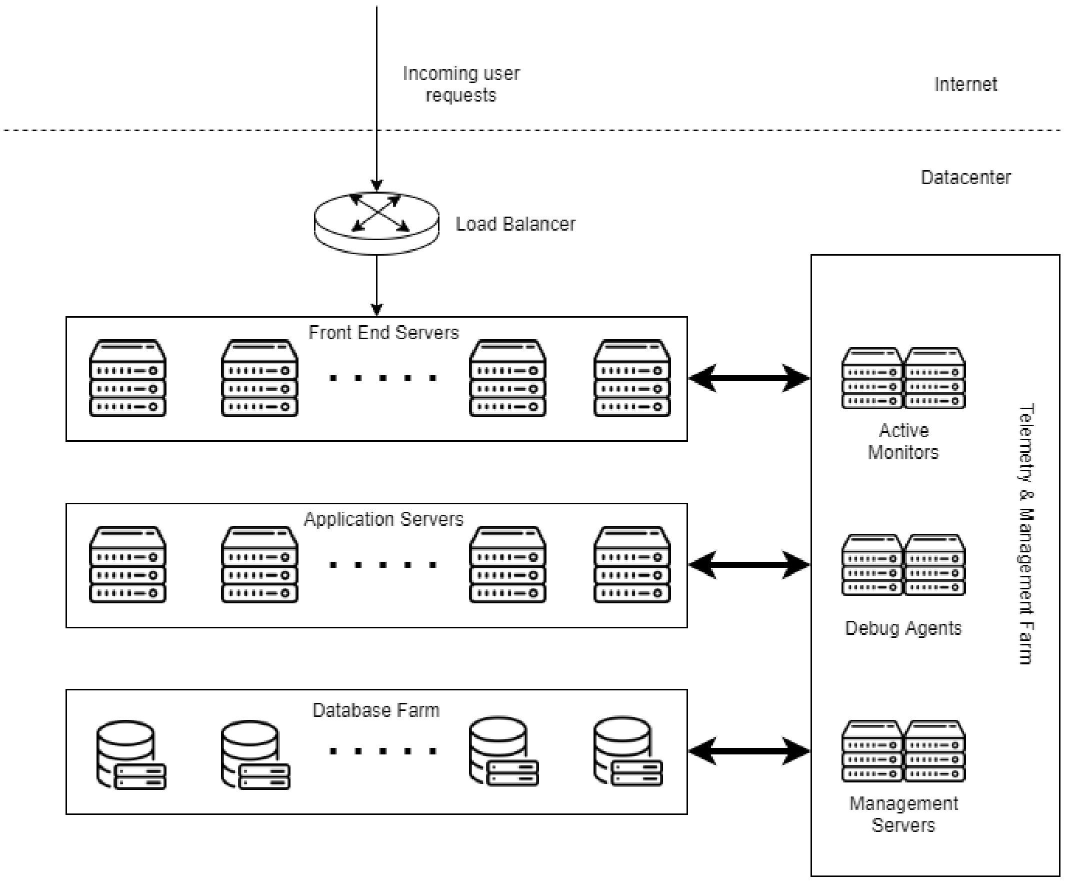


Figure 1.1: Overview of IT infrastructure of a service.[51]

tentially chaotic is a major challenge in the emerging AIOps field.

One of the ways to predict IT incidents is to identify incident alerts from system metrics (CPU usage, disk read/write operations, etc.) [51]. Irregularity in a subset of system metrics values are early symptoms of an incident and can be used to predict an incident. Most of the system metrics collected over time are basically time-series [1] data. The complexity of the time-series dataset spans a wide range. Time series lying on the lower end of the spectrum have a predictive structure and can be easily modeled. Time series lying on the higher end of the spectrum are complex and difficult to model. Moreover, not every time-series dataset is predictable, i.e., some of them could be completely random or white noise. If we know beforehand which system metrics are predictable then we can easily build models for IT incident prediction. Hence, quantifying the complexity of time series can help us rank the available system metrics and select a subset of system metrics with the lowest complexity.

[1]Note that throughout this work we use the convention of hyphenating 'time-series' when used as an adjective, but not when used as a noun ('time series').

Quantifying complexity or ”predicting predictability” is not a new concept and has been central in many disciplines [30]. Understanding complexity of heart rate dynamics in biomedicine [64], understanding dynamics of earthquake tremors in seismology [29] and predicting stock prices in finance [78]. The forecasting performance, in the before mentioned applications, is measured by *the realized predictability* [57], i.e., by using Forecasting Error (FE) or Mean Absolute Error (MAE), which is dependent on the model being used. Ideally, the system's *intrinsic predictability* should be used to judge the proficiency of a model, but often this intrinsic predictability is unknown, or, in certain cases, beyond what we can know. It is vital to understand the relationship between realized and intrinsic predictability to judge the proficiency of a model. Most importantly, intrinsic predictability has the potential to indicate whether the model is limiting the realized predictability of time series. Therefore, if we know the intrinsic predictability of a system and its realized predictability using a specific model, the difference between the two indicates how much predictability can be improved [10].

However, to the best of our knowledge, no research has been done to understand the complexity or dynamics of IT infrastructure. This thesis presents an exploratory study to understand and quantify the complexity of IT infrastructure. In particular, we focus on predicting the predictability of system metrics to identify informative system metrics. Knowing the intrinsic complexity of IT infrastructure components can help administrators to understand the state of infrastructure at a given point in time. Computing the exact intrinsic complexity of a system isn't always possible, but we can approximate the upper bound intrinsic complexity using complexity measures [7].

The research for this thesis is performed in cooperation with ING, a Dutch multinational banking and financial services corporation headquartered in Amsterdam. ING offers over a dozen online banking applications to its 8.5 million customers and has a huge IT infrastructure setup for the same.

1.2 Problem Statement

In IT incident management, one important problem is anomaly detection on system metrics [27]. The normal approach to this is to first recognize the normal behavior, then predicting future behavior, and then assessing whether an incoming point is an anomaly based on the fitted model. Very often, applications stacked on a system yield a lot of metrics, and data points are often recorded in minute granularity. The task of handling this data quickly becomes difficult in terms of storage capacity. Normally, these metrics get aggregated and are stored in a database or cloud, and later on are deleted, which makes it pretty difficult to have good forecasting models on system metrics. Hence, determining which system metrics are valuable to keep and structural enough to be analyzed for anomaly detection can be very helpful.

Quantifying the predictability or complexity of a system metric can help in ranking system metrics based on predictability and then identifying a subset of the most informative system metrics for the IT incident prediction task. Most importantly, a forecasting model might not work on certain system metrics or time series, simply because it cannot capture the underlying linear or non-linear process. If a forecasting model is not a good fit, a practitioner needs to know if it is due to the absence of predictive structure (i.e., complex) in the time series *or* the model used by the practitioner is simply not good enough to exploit the structure of time series.

1.3 Approach

There has been a lot of research work conducted in the field of measuring the complexity of data [43][50][25], and most of the research corroborates that complexity of data is a major challenge for the prediction task. Most importantly, it is critical for a forecasting model to understand the underlying information generating process of a system. Among different approaches to measuring the complexity of data, entropy-based ones are inspired by nonlinear dynamics. To quantify the complexity of time series and tackle the problem at hand, in this research, we focus on entropy methods. The intrinsic predictability of time series can be approximated with model-free methods, such as entropy. Entropy is defined as the average level of ”information”, ”surprise” or ”complexity”.

The thesis presents a new quantitative metric (entropy method) named Reverse Weighted Dispersion Entropy (RWDE) for quantifying the intrinsic predictability of nonlinear time series. We analyze the entropy method on system metrics collected from deployed systems on ING Private Cloud (IPC) and try to gain valuable insights from the relationship between intrinsic and realized predictability. The major application of the developed entropy method in this thesis will be to address the problem of quantifying the predictability or complexity of system metrics.

1.4 Main Research Questions & Contributions

To characterize the proposed Reverse Weighted Dispersion Entropy (RWDE) method, in this study we address 3 research questions. These research questions focus on the characteristics of RWDE. Over the last two decades, different studies have proposed quite a few entropy methods. It is important to understand how different RWDE is from them and how its performance compares to existing entropy methods. This leads us to our first research question :

RQ1: How does RWDE perform compared to other entropy methods?

RWDE is derived from Permutation Entropy (PE) by adding a couple of features. In order to understand the role of these features in RWDE’s performance, we formulate the following sub-questions:

RQ1.1 what is the effect of the weighted entropy method?

RQ1.2 what is the effect of adding ”distance to noise” in entropy calculation?

RWDE is an entropy method that requires defining a set of parameters to calculate the entropy value. Therefore, it is important to understand the sensitivity of RWDE towards these parameters. So, our second research question is as follows:

RQ2. what is the effect of parameters on RWDE?

Finally, given that RWDE is a method to quantify the intrinsic predictability of time series, it would be interesting to explore the relationship between intrinsic predictability and realized predictability. Insights in such a relationship could help in determining if a model is a mismatch for forecasting a time series. This leads us to our last research question :

RQ3. What is the relationship between intrinsic predictability and realized predictability?

To the best of our knowledge, this is the first exploratory study to understand and quantify the complexity of system metrics. As a step towards better understanding the complexity of system metrics, this thesis makes three contributions :

1. **Entropy Method** : The first contribution is a novel generalizable entropy method; Reverse Weighted Dispersion Entropy (RWDE).

2. **Experiments** : The second contribution is a statistical analysis of RWDE thorough experiments and comparison to other existing entropy methods. Further, we present an empirical analysis of RWDE's performance on real-world large-scale system metric data from ING.

3. **Open Source Contribution.** We make RWDE and other entropy methods discussed in this study available as an open-source Python package.

4. **Critical Analysis** : The last contribution is a critical analysis of RWDE in relation to model-based predictability of time series. We discuss our findings to conclude the usability of RWDE for identifying informative system metrics and determining the suitability of forecasting strategy for these metrics.

Among our findings, we establish that on an average RWDE performs better than existing entropy methods. Our experiments show that RWDE can detect sudden fluctuations, spikes, changes in information density over time, which makes RWDE a suitable feature for applications such as anomaly detection, and clustering. Further, our data analysis on intrinsic predictability calculated by RWDE and realized predictability using forecasting models support that RWDE can approximate the true complexity of a time series. Lastly, based on our data analysis we discuss how RWDE can be used to rank and select system metrics for IT incident prediction tasks or forecasting tasks.

1.5 Thesis Outline

The remainder of this thesis is structured as follows. In Chapter 2, we discuss background information and related work on entropy methods. In Chapter 3, we introduce the novel entropy method: Reverse Weighted Dispersion Entropy (RWDE). In Chapter 4, we describe the deployment context of ING and how the bank monitors IT infrastructure and collects system metrics. Chapter 5 presents simulation experiments to evaluate the performance of RWDE in comparison to other entropy methods. In Chapter 6, we apply RWDE on cloud metric data collected at ING and establish a relationship between entropy and model-based forecasting error. We discuss our main findings on the performance of RWDE and answer the research questions in Chapter 7. Here we also discuss and consider implications, threats to the validity of this study, and present directions for future work. Lastly, in Chapter 8 we conclude the main contributions and findings of this study.

To readers who are interested in learning about the Reverse Weighted Dispersion Entropy (RWDE) only, we recommend reading Chapter 3.

Chapter 2

Background and Related Work

This chapter presents background and related work to this study. A lot of research has been performed in the area of quantifying the complexity of time series using entropy. First, we discuss previous research work similar to this thesis. Later in this chapter, we discuss time series data types, different time-series entropy methods and how they have been applied in different domains.

2.1 Related Work

In the year 1948, Claude Shannon [66] defined entropy as the measure of ”information” inherent in a system’s outcome. Since then quite a few entropy methods have been introduced to measure the information or complexity of time series. In 2002, Bandt and Pompe proposed Permutation Entropy (PE) to quantify the complexity of time series [7]. Permutation Entropy has been used in a wide range of data analysis applications such as finance, manufacturing, and especially in the Bio-medical field [6][3][75]. Due to its advantages, PE has attracted a lot of attention from practitioners and academics. In the last decade, Permutation Entropy has been very popular in research and a lot of studies have proposed variants of PE. Some of the most noticeable variants of PE are Weighted Permutation Entropy (WPE) [22], Reverse Permutation Entropy (RPE) [8], Dispersion Entropy (DE) [62], and Reverse Dispersion Entropy (RDE) [48]. We will understand these entropy methods in detail later in this chapter.

Use of entropy methods in IT system monitoring tools or cloud monitoring tools such as Amazon cloudwatch, Graphite, etc. is common. However, the entropy values are mostly ignored and only taken into account when a threshold is surpassed. There has been only a handful of research on the use of entropy in IT incident detection. Navaz et al. [52] built an anomaly detection system based on Shannon entropy to detect Distributed Denial of Service (DDoS) attacks on cloud systems. Wang et al. [71] in their patent used the entropy method to design a cloud anomaly detection system. However, the system is still a design and has not been deployed in a practical scenario. These studies use very limited cloud metrics for a specific application. The dearth of research in using system metrics for IT incident management can be accredited to the complexity and abundance of system metrics [53][26]. The applications stacked on a system yield a lot of metrics, and data points are often recorded in minute granularity. Many of these cloud metrics have high complexity and most of the forecasting strategies fail to perform well. These factors make it pretty difficult to select the right set of metrics for building a good forecasting model for any application.

Cloud systems yield a large number of system metrics with minute granularity. Mostly, such large-scale metrics are not labelled by experts for applications like anomaly detection. Therefore, many studies have used unsupervised machine learning techniques such as neural networks for anomaly detection [14][59]. Miranskyy et al. [36] used Gated-Recurrent-Unit-based auto-encoder for detecting anomalies from multi-dimensional cloud metric data. Similarly, Gander el al. [24] and Liu et al. [49] proposed frameworks based on unsupervised machine learning for detecting security attacks on a cloud system. However, these studies consider well-known dataset with a limited number of cloud metrics. They do not explain the selection of metrics for anomaly detection applications, rather they consider all of them due to the limited number of metrics.

In this study, we first "predict the predictability" of system metrics using the entropy method to identify informative system metrics. We call the predictability quantified by the entropy method as intrinsic predictability. Secondly, we focus on the relationship between intrinsic predictability and realized predictability. Realized predictability is the predictability quantified by model-based methods (forecasting models). The most closely related work, exploring such a relationship has been done by Garland et al.[26]. They used Weighted Permutation Entropy (WPE) with linear and non-linear forecasting models to establish a relationship between intrinsic predictability and realized predictability. They use experimentally generated computer-performance data for establishing the relationship. Our study differs in mainly two aspects: i) We use a more sophisticated method, Reverse Weighted Dispersion Entropy (RWDE) to quantify intrinsic predictability, ii) Secondly, we focus on real-world cloud metric data from a global, large scale cloud system. Such data cannot be experimentally manufactured. Huang et al. [31] also proposed a very similar study. They used limited time series data with a known predictive structure to establish a relationship. However, in most of the real-world scenarios, data with known predictive structure is hardly available. Most of the system metrics considered in this study exhibit an unknown non-linear structure. In this study, we establish a relationship between intrinsic predictability and realized predictability with RWDE, based on cloud metric data from a large cloud system.

To the best of our knowledge, no research study has been done to understand the complexity of cloud metrics using entropy methods. In this study, we present the first attempt of quantifying the complexity of cloud metrics using a new entropy method : Reverse Weighted Dispersion Entropy (RWDE). We also focus on the relationship between intrinsic predictability determined using RWDE and model-based predictability to determine if a model is not performing as expected according to intrinsic predictability.

2.2 The Time Series Data Type

Observing the dynamics of a real-world process is part of human experience or a result of human curiosity. For example, understanding the complexity of heart rate dynamics in bio-medicine [64], understanding dynamics of earthquake tremors in seismology [37] and predicting stock prices in finance [78]. The dynamics of each system is understood by collecting measurements of a system over a regular time interval. This collection of measurements of a variable or system over time is known as *time series* or time-series data type [23]. Figure 2.1 shows some examples of real-world time-series.

Formally, a time series can be represented as vector of measurements x_i, each collected at time-stamp t_i. Generally, time series measurements are sampled uniformly through time (also known as uniformly-sampled time series) i.e after constant period of time. Uniformly sampled time series basically form an ordered sequential data vector $x = (x_1, x_2, ..., x_N)$ measured at time $t = (0, \Delta t, 2\Delta t, ..., (N-1)\Delta t)$, where Δt represents constant sampling period.

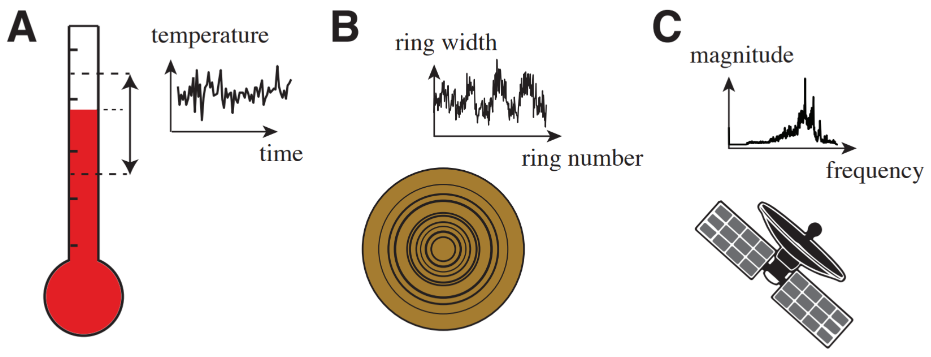


Figure 2.1: Some examples of real-world time-series or sequential data. **A** represents temperature measurements at different time steps. **B** is a sequence of trunk widths from top to bottom of a tree. **C** represents the frequency of signal over time sent by satellite. [23]

The examples shown in Figure 2.1 are time series obtained by measuring a single entity. These types of time series are known as *univariate time-series*. We get a *multivariate time-series* if we measure multiple entities related to a system simultaneously. In this study, we specifically focus on univariate time-series. We are primarily interested in quantifying the complexity or predictability of univariate time-series.

2.3 Information Theory and Entropy Basics

The field of information theory concerns itself with the study of quantification, storage, and communication of information. It is at the intersection of various research fields such as probability theory, statistics, computer science, statistical mechanics, information engineering, and electrical engineering [66]. This study is closely related to the field of information theory, as it focuses on quantifying information with respect to a random variable or the distribution associated with it.

One of the most important measures of information in information theory is *entropy*. The entropy of a system or process is defined as the average level of ”information”, ”surprise”, or ”uncertainty” inherent in the system’s possible outcomes. An example is the information or uncertainty present with respect to specifying outcome from a roll of fair dice. The entropy of a system is high if the uncertainty with respect to its outcome is high i.e unpredictable. Similarly, the entropy of a system is low if the uncertainty with respect to its outcome is low, i.e., predictable. The term entropy was first defined by Claude Shannon [66], to measure information content in communication signals.

The foundation of most of the entropy methods is the classical *Shannon entropy*. It is inspired by and named after Boltzmann’s H-theorem [12]. The Shannon entropy (H) of a discrete random variable $X = (x_1, x_2, ..., x_n)$ with probability mass function $P(X)$ is defined as:

$$H(X) = E[I(X)] = E[-log(P(X))]$$

Where E is the Expected value and I represents the information content of X. The above entropy equation can be written explicitly as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log_b P(x_i)$$

Where b is the base of the entropy method and is generally set to 2. Further, Shannon entropy has a maximum value $H(X) = log_b(n)$ when the random variable X has a uniform distribution, i.e., $P(x_i) = 1/n, i = 1, ..., n)$. Similarly, it has minimum value $H(X) = 0$ when X has single-point distribution i.e. $P(x_{i0}) = 1$ for a specific value i_0. Therefore, the bounds of Shannon entropy are $0 \leq H(X) \leq log_b(n)$. Based on the upper bound, the normalized Shannon entropy is defined as follows :

$$H(X) = -\frac{1}{log_b n} \sum_{i=1}^{n} P(x_i) log_b P(x_i)$$

2.4 Time Series Entropy

Entropy as a measure for quantifying disorder or complexity of time series has been heavily researched in the field of time-series analysis. In our study, we assume the behavior of the

IT system is captured by the properties of the time series produced by the system. Therefore, the entropy measure of such a time series is a measure of the system's complexity. Quite a few entropy methods have been proposed in the last decades. However, the most commonly used entropy methods are Sample Entropy (SampEn) and Approximate Entropy (ApEn).

Approximate Entropy [58] was proposed by Pincus in the year 1991 to quantify the unpredictability of time-series data collected from medical systems such as heart rate monitors. Later in the year 2000, Richman and Moorman proposed Sample Entropy [60], which is a modification of Approximate Entropy. ApEn and SampEn are popularly used as statistical techniques to quantify regularity and the unpredictability of fluctuations over time-series data. At present, Sample Entropy is commonly is used method for assessing complexity of psychological signals due to its relatively easier implementation. Although both the algorithms have a complexity of $O(N^2)$, the number of operations required for Sampled Entropy is slightly less than for Approximate Entropy [69].

Disparate to Approximate Entropy and Sample Entropy, in 2002 Bandt and Pompe proposed Permutation Entropy (PE) to quantify the complexity of time series [7]. In this section, we introduce Permutation Entropy and its prominent variants. We do not consider Sample Entropy and Approximate Entropy in this study, because it has been shown in previous studies that PE performs better than SampEn and ApEn [76][44].

2.4.1 Permutation Entropy

The idea behind Permutation Entropy is to capture the distribution of permutation patterns of values present in a time series. In PE method, The patterns formed by assigning rank order to every value are known as *Ordinal Patterns*. Moreover, Ordinal patterns are the defining aspect of all entropy methods derived from PE. Hence in this study, these methods are called ordinal pattern based entropy methods. Bandt and Pompe [7] described Permutation Entropy as a "natural complexity measure for time series". The main properties of PE are:

- PE is a model-free method to quantify the complexity of time series based on entropy and symbolic dynamics.
- PE is robust with respect to non-linear, noisy data, and computationally efficient.

Consider a one-dimensional time series $X(t) = x_t; t = 1, ..., T$. We explain PE similar to the way Reverse Dispersion Entropy (RDE) is explained in [48]. The PE algorithm can be divided into 4 simple steps, as follows:

1. *Partitioning state space*: The first step is to partition the time series $X(t)$ in vectors to form a matrix. The partition is based on two hyper-parameters shown in Table 2.1. Further, we get the matrix by joining all the (L) partition vectors of length m with time delay τ:

Table 2.1: Hyper-parameters for partitioning time series in vectors.

Parameter	Time Delay (τ)	Embedding Dimension (m)
Description	Defines the number of time periods between two partition vectors.	Defines the length of each partition vector.
Valid Range	Any positive integer	Any integer > 1
Recommended Value	1	$3 \leq m \leq 7$

$$\begin{bmatrix} x_1 & x_{1+\tau} & x_{1+2\tau} & \cdots & x_{1+(m-1)\tau} \\ x_2 & x_{2+\tau} & x_{2+2\tau} & \cdots & x_{2+(m-1)\tau} \\ & & \cdots\cdots\cdots\cdots\cdots & & \\ x_i & x_{i+\tau} & x_{i+2\tau} & \cdots & x_{i+(m-1)\tau} \\ & & \cdots\cdots\cdots\cdots\cdots & & \\ x_L & x_{L+\tau} & x_{L+2\tau} & \cdots & x_{L+(m-1)\tau} \end{bmatrix}$$

In this study, we keep $\tau = 1$ as recommended by Bandt and Pompe [7]. This would mean that we don't miss on any patterns in a time series. Therefore, the matrix we get after substituting $\tau = 1$ is :

$$\begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_{1+(m-1)} \\ x_2 & x_3 & x_4 & \cdots & x_{2+(m-1)} \\ & & \cdots\cdots\cdots\cdots\cdots & & \\ x_i & x_{i+1} & x_{i+2} & \cdots & x_{i+(m-1)} \\ & & \cdots\cdots\cdots\cdots\cdots & & \\ x_L & x_{L+1} & x_{L+2} & \cdots & x_{L+(m-1)} \end{bmatrix}$$

2. *Generating ordinal patterns*: In the second step we assign ranks $R = 0, 1, ..., m-1$ to the real-values in each vector or row of the matrix based on value. That is, we replace the element with lowest value in row with 0 and element with highest value with $m-1$. If two elements have same value then the later element in position will get higher rank.Therefore, each row of the matrix becomes a pattern $\pi_i = [j_1, j_2, ..., j_m]$ where $i = 1, 2, ..., L$ and j = 0,1,...,m-1. Each pattern π_i is one of the $m!$ possible permutations.

3. Calculating relative frequencies: In the third step we calculate the relative frequency of each observed pattern π_i. We represent the relative frequency distribution as probability distribution:

$$P(\pi_i) = \frac{number\, of\, occurrences\, of\, pattern\, \pi_i}{total\, number\, of\, patterns\, or\, partition\, vectors\, (L)}$$

4. Computing PE: Finally, we use the probabilities calculated in the previous step and calculate Permutation entropy defined as the Shannon Entropy of permutations:

$$PE(X,m,\tau) = -\sum_{i=1}^{m!} P(\pi_i) log_2 P(\pi_i)$$

Permutation Entropy has maximum value $PE(X,m,\tau) = log_2(m!)$ when the permutations of time series X has a uniform distribution, i.e., $P(\pi_i) = 1/m!, i = 1,...,m!)$. Similarly, it has minimum value $H(X,m,\tau) = 0$ when there is only a single permutation, i.e., $P(\pi_{i_0}) = 1$ for a specific value $i_0 = [1,m!]$. Therefore, the bounds of PE are $0 \leq PE(X,m,\tau) \leq log_2(m!)$. Based on the upper bound, the normalized Permutation Entropy with bounds $0 \leq PE_{NE}(X,m,\tau) \leq 1$ is defined as follows :

$$PE_{NE}(X,m,\tau) = -\frac{1}{log_2 m!} \sum_{i=1}^{m!} P(\pi_i) log_2 P(\pi_i)$$

Example: Consider a time series of seven values

$$x = \{4,7,9,10,6,11,3\}$$

We set the embedding dimension or order $m = 3$. Then, after performing step 1, the partition state space we obtain is:

$$\begin{bmatrix} 4 & 7 & 9 & 10 & 6 \\ 7 & 9 & 10 & 6 & 11 \\ 9 & 10 & 6 & 11 & 3 \end{bmatrix}$$

From the partition state, we obtain the ordinal patterns :

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 0 & 2 \\ 2 & 2 & 0 & 2 & 0 \end{bmatrix}$$

In the third step, we calculate the relative frequencies of ordinal patterns. From above matrix, the patterns have following frequencies : (0,1,2)=2/5, (1,2,0)=2/5 and (1,0,2)=1/5. Finally, we calculate the Shannon Entropy of these frequencies :

$$PE(x) = -2 \cdot \frac{2}{5} log_2 \frac{2}{5} - \frac{1}{5} log_2 \frac{1}{5} \approx 1.522$$

2.4.2 Weighted Permutation Entropy

Weighted Permutation Entropy (WPE) was introduced in the year 2013 by Fadlallah et al. [22] to incorporate amplitude changes of time series in Permutation Entropy. The idea behind WPE is to make PE robust towards observational noise which is larger locally but smaller than the overall trend of the time series. For example, consider a signal switching between two points with very small additive noise. The PE of such a signal would be close to 1, i.e., a complex signal, due to noise around fixed points. However, we know that the signal has pretty much a deterministic structure. In this case, if we weigh the permutations with respect to amplitude changes in the time series, we get a better approximation of the predictive structure in time series.

All the steps for computing WPE are the same as for PE apart from step 3, i.e., calculating relative frequencies of permutations. For WPE, we calculate the weight of each permutation $x_i^m, i = 1,2,...L$ as follows :

$$\omega(x_i^m) = \frac{1}{m}\sum_{j=i}^{i+m}(x_j - \bar{x}_i^m)^2$$

where x_i^m is a sequence of values $x_i,...,x_{i+m}$ or the i^{th} embedding vector of length m and $\bar{x}_i^m$ is the arithmetic mean of the embedding vector. Using the calculated weights, we calculate the weighted probability of each pattern as follows:

$$P_w(\pi) = \frac{\sum_{i\leq N-m}\omega(x_i^m)\cdot\delta(x_i^m,\pi)}{\sum_{i\leq N-m}\omega(x_i^m)}$$

where N is the length of the time series and $\delta(x,y) = 1$ if $x = y, 0$ otherwise. Using the above equation we calculate the weighted probability of each possible permutation $\pi_i, i = 1,2,...,m!$. The weighted probability of each permutation gives more weight to permutations that contain large amplitude changes compared to permutations with small amplitude changes. Lastly, we compute the WPE similar to PE in step 4 :

$$WPE(X,m,\tau) = -\sum_{i=1}^{m!}P_w(\pi_i)log_2P_w(\pi_i)$$

Further, we divide WPE with $log_2(m!)$ to get the normalized WPE ($0 \leq WPE_{NE}(X,m,\tau) \leq 1$) :

$$WPE_{NE}(X,m,\tau) = -\frac{1}{log_2m!}\sum_{i=1}^{m!}P_w(\pi_i)log_2P_w(\pi_i)$$

Example: We again consider the same time series of seven values:

$$x = \{4,7,9,10,6,11,3\}$$

The first two steps of WPE are similar to WPE. In the third step, we calculate the weight each permutation. For example, the weight for first two appearance of (0,1,2) are :

$$\omega(0,1,2)_1 = \frac{1}{3}[(4-6.67)^2 + (7-6.67)^2 + (9-6.67)^2] \approx 4.22$$

$$\omega(0,1,2)_2 = \frac{1}{3}[(7-8.67)^2 + (9-8.67)^2 + (10-8.67)^2] \approx 1.55$$

Similarly, we calculate weights for other ordinal patterns. We get the sum of weights for all patterns ≈ 24.22. Then, we compute the relative weight of each unique pattern. For example for pattern (0,1,2) :

$$P_w(0,1,2) = \frac{4.22+1.55}{24.22} \approx 0.238$$

Finally, we compute the Shannon entropy of relative weights to get $WPE(x) \approx 1.413$.

2.4.3 Reverse Permutation Entropy

Reverse Permutation Entropy (RPE) was proposed by Bandt in the year 2017. Bandt used RPE to identify different sleep stages from human brain signals. RPE quantifies the complexity of time series by calculating the distance of a given time series from white noise. The terminology "Reverse" comes from the fact that the convention for interpreting the entropy value of RPE is opposite with respect to PE. A high RPE value (i.e., close to 1) indicates a large distance from white noise, i.e., predictable time series, whereas, a low RPE value (i.e. close to 0) indicates an insignificant distance from white noise, i.e., complex time series. Li et al. [47] applied RPE on ship-radiated underwater signals to extract features from signals and concluded RPE was more stable than PE.

All the steps for computing RPE are the same as for PE apart from the last step. Unlike PE, Reverse Permutation Entropy uses the Euclidean distance to calculate entropy rather than Shannon Entropy. Generally, a white noise signal is expected to have a uniform distribution of patterns. For each pattern given in a time series, RPE uses the Euclidean distance to calculate the distance of the respective pattern distribution from a uniform distribution of patterns. RPE is defined as :

$$RPE(X,m,\tau) = \sum_{i=1}^{m!}(P(\pi_i) - \frac{1}{m!})^2 = \sum_{i=1}^{m!} P(\pi_i)^2 - \frac{1}{m!}$$

Reverse Permutation Entropy has a minimum value $RPE(X,m,\tau) = 0$ when the permutations of time series X has a uniform distribution, i.e., $P(\pi_i) = 1/m!, i = 1,...,m!)$. Similarly, it has maximum value $RPE(X,m,\tau) = 1 - \frac{1}{m!}$ when there is only a single permutation pattern, i.e., $P(\pi_{i_0}) = 1$ for a specific value $i_0 = [1,m!]$. Therefore, the bounds of RPE are $0 \leq RPE(X,m,\tau) \leq 1 - \frac{1}{m!}$. Based on the upper bound, the normalized Reverse Permutation Entropy with bounds $0 \leq RPE_{NE}(X,m,\tau) \leq 1$ is defined as follows :

$$RPE_{NE}(X,m,\tau) = \frac{\sum_{i=1}^{m!} P(\pi_i)^2 - \frac{1}{m!}}{1 - \frac{1}{m!}}$$

Example: We again consider the same time series:

$$x = \{4,7,9,10,6,11,3\}$$

For RPE, only the last step differs from PE. In the last step, we calculate the distance of each pattern distribution from a uniform distribution of patterns as follows:

$$RPE(x) = [2 \cdot (\frac{2}{5})^2 + (\frac{1}{5})^2] - \frac{1}{3!} \approx 0.193$$

2.4.4 Dispersion Entropy

Dispersion Entropy (DE) was introduced in the year 2016 by Azami and Rostaghi [62] to quantify the complexity of time series and was first applied to the bearing fault database [77]. The bearing fault database consist of vibration signals of the ball bearings used in machinery. The authors used DE to distinguish normal bearing vibrations from the faulty ones. Unlike PE, Dispersion Entropy does not use ordinal patterns for computing entropy. Rather DE uses the Normal Cumulative Distribution Function (NCDF) [20] to generate permutations. Dispersion Entropy is as computationally efficient as PE. However, it performs better than PE in terms of detecting amplitude changes and distinguishing different databases. In a different study [63], Rostaghi compared the performance of Dispersion Entropy, Permutation Entropy, and Approximate Entropy on time series data from real-time systems to conclude that DE is more suitable for real-time systems.

Similar to PE, the Dispersion Entropy algorithm can be computed in four steps :

1. *Mapping time series to c classes*: Consider a time series $X = \{$x(i), i=1,2,...,N$\}$ with N values. We map X to $Y = \{$y(i), i=1,2,...,N$\}$ using NCDF, where $y(i)$ ranges from 0 to 1. Then, We map Y to $Z = \{$z(i), i=1,2,...,N$\}$ using a linear mapping function $round(c \cdot y(i) + 0.5)$. Where c is the number of classes and $z(i)$ is a positive integer in the range $[1, c]$.

2. *Phase space reconstruction*: We construct Z into L embedding vectors with the same time delay τ and embedding dimension m, respectively. Here we consider that time delay is unity i.e. $\tau = 1$. The matrix consisting of all embedding vectors can be represented as follows:

$$\begin{bmatrix} z_1 & z_2 & z_3 & \cdots & z_{1+(m-1)} \\ z_2 & z_3 & z_4 & \cdots & z_{2+(m-1)} \\ & & \cdots\cdots\cdots\cdots & & \\ z_i & z_{i+1} & z_{i+2} & \cdots & z_{i+(m-1)} \\ & & \cdots\cdots\cdots\cdots & & \\ z_L & z_{L+1} & z_{L+2} & \cdots & z_{L+(m-1)} \end{bmatrix}$$

   Where the number of embedding vectors L is equal to $N - (m-1)$. Each embedding vector is of length m and has values in the range $[1, c]$. Therefore, the total number of possible permutation patterns is c^m.

3. *Calculate the probability of each Dispersion pattern* : The probability of i_{th} possible pattern is expressed as:

$$P(\pi_i) = \frac{number\,of\,occurrences\,of\,pattern\,\pi_i}{L} \; where(1 \leq i \leq c^m)$$

4. *Calculating Dispersion Entropy*: Finally, the probabilities calculated in the third step are used to calculate Dispersion Entropy defined as the Shannon Entropy of Dispersion patterns:

$$DE(X,m,c,\tau) = -\sum_{i=1}^{c^m} P(\pi_i) log_2 P(\pi_i)$$

Further, the normalized Dispersion Entropy DE_{NE} is calculated by dividing DE by $log_2(c^m)$:

$$DE_{NE}(X,m,c,\tau) = -\frac{1}{log_2(c^m)} \sum_{i=1}^{c^m} P(\pi_i) log_2 P(\pi_i)$$

Example: We consider the following time series for Dispersion Entropy:

$$x = \{9, 8, 1, 12, 5, 3, 1.5, 8.01, 2.99, 4, 1, 10\}$$

We set order $m = 2$ and classes $c = 3$. First, we map the time series to c classes using the NCDF function. We get the following series after passing x to the NCDF function:

$$x = \{0.82, 0.75, 0.21, 0.94, 0.52, 0.05, 0.241, 0.75, 0.35, 0.43, 0.11, 0.87\}$$

Then, similar to PE we generate the partition space in step 2 and then compute the probability or frequency of each dispersion pattern. We get the following pattern distribution:

$$P(\pi_{11}) = \frac{1}{11}, P(\pi_{12}) = \frac{0}{11}, P(\pi_{13}) = \frac{3}{11},$$
$$P(\pi_{21}) = \frac{2}{11}, P(\pi_{22}) = \frac{1}{11}, P(\pi_{23}) = \frac{0}{11},$$
$$P(\pi_{31}) = \frac{1}{11}, P(\pi_{32}) = \frac{2}{11}, P(\pi_{33}) = \frac{1}{11}$$

Finally, $DE(x = -\sum_{i=1}^{c^m} P(\pi_i) log_2 P(\pi_i) \approx 2.66$ is computed. Remember, we use log_2 in this example for calculation.

2.4.5 Reverse Dispersion Entropy

In 2019, Li et al. [48] proposed Reverse Dispersion Entropy (RDE) which combined all the characteristics of PE, DE, and RDE, i.e., ordinal patterns, amplitude variance, and distance information. Similar to RPE, RDE uses Euclidean distance to measure the distance of the dispersion pattern distribution from a uniform distribution.

All the steps for computing RDE are the same as DE apart from the last step. RDE is computed as follows:

$$RDE(X,m,c,\tau) = \sum_{i=1}^{c^m}(P(\pi_i) - \frac{1}{c^m})^2 = \sum_{i=1}^{c^m} P(\pi_i)^2 - \frac{1}{c^m}$$

Reverse Dispersion Entropy has a minimum value $RDE(X,m,c,\tau) = 0$ when the dispersion patterns of time series X have a uniform distribution i.e. $P(\pi_i) = 1/c^m, i = 1,...,c^m)$. Similarly, it has maximum value $RDE(X,m,c,\tau) = 1 - \frac{1}{c^m}$ when there is only a single Dispersion pattern, i.e., $P(\pi_{i_0}) = 1$ for a specific value $i_0 = [1,c^m]$. Therefore, the bounds of RDE are $0 \leq RDE(X,m,c,\tau) \leq 1 - \frac{1}{c^m}$. Based on the upper bound, the normalized Reverse Dispersion Entropy with bounds $0 \leq RDE_{NE}(X,m,c,\tau) \leq 1$ is defined as follows :

$$RDE_{NE}(X,m,c,\tau) = \frac{\sum_{i=1}^{c^m} P(\pi_i)^2 - \frac{1}{c^m}}{1 - \frac{1}{c^m}}$$

Example: For RDE, we consider the same time series as for Dispersion Entropy:

$$x = \{9,8,1,12,5,3,1.5,8.01,2.99,4,1,10\}$$

We keep all the parameters the same as the example for DE. All the steps for RDE are the same as DE apart from last step. In last step, we compute the distance of each dispersion pattern distribution from a uniform distribution $RDE(X,m,c,\tau) = \sum_{i=1}^{c^m} P(\pi_i)^2 - \frac{1}{c^m} \approx 0.0624$

2.5 The Kolgomorov-Sinai Entropy

In dynamical systems, it is difficult to explain or quantify the random behavior exhibited by the system. In the year 1958, Kolmogorov introduced Kolmogorov Complexity [4]; an ideal entropy for quantifying chaos in dynamical systems [9]. Kolmogorov Complexity was later reformulated by Sinai and was later known as Kolmogorov-Sinai Entropy (KSE) [70]. At present, KSE is known as a central measure for quantifying the complexity of a dynamical system [28]. Russian mathematician Yakov Pesin showed that a dynamical will display chaotic behavior if the KSE is greater than zero. However, determination or approximation of KS Entropy is a hard problem and is not feasible. All the entropy methods discussed in this study, attempt to approximate KS entropy.

First, we discuss the theoretical background of KS Entropy. Consider a dynamical system represented as $(\Omega, \mathcal{B}, P, T)$. We establish that $(\Omega, \mathcal{B}, P)$ is a probability space with a mapping function or transformation function $T : \Omega \rightarrow \Omega$ satisfying $\mu(T^{-1}(B))$ for all $B \in \mathcal{B}$. In other words, Ω is the set of states of the system, and P is the probability distribution of these states. The mapping function T specifies the state of system $T(\omega)$ at time $t+1$ if the system was in state $\omega \in \Omega$ at time t.

Consider a finite partition $\alpha = C_1, C_2, ..., C_k$ of the dynamic system Ω such that $\alpha \subset \mathcal{B}$. The Shannon Entropy of such partition is given as :

$$H(\alpha) = -\sum_{i=1}^{k} P(C_i) log P(C_i)$$

For $A = 1, 2, ...k$, consider a partition $\alpha_{\updownarrow}$ of Ω formed by set $A^{(l)}$ of words $a_1, a_2, ..., a_{(l)}$ of length (l) . Then we have:

$$C_{a_1, a_2, ..., a_{\updownarrow}} = \left\{ \omega \in \Omega | (\omega, T(\omega), ...T^{o\updownarrow-1}(\omega)) \in C_{a_1} \times C_{a_2} \times, ..., C_{a_{\updownarrow}} \right\}$$

Where T^{ok} denotes the $k-fold$ concatenation of T and T^{o0} is the identity map. The probability distribution of these words $P(C_{a_1, a_2, ..., a_{\updownarrow}})$ represent the information om complexity of system. As the length of words $\updownarrow \to \infty$, we get the entropy rate of α as :

$$EntroRate(\alpha) = \lim_{\updownarrow \to \infty} \frac{1}{k} H(\alpha_k) = \lim_{\updownarrow \to \infty} (H(\alpha_k) - H(\alpha_{\updownarrow-1}))$$

In other words, the above equation represents the mean information provided by each symbol or word.

However, we want a complexity measure that is not dependent on a fixed partition. Therefore, The Kolmogorov-Sinai Entropy of a system with regard to a finite or countably infinite partition α is given as:

$$KS = \sup_{\alpha} EntroRate(\alpha),$$

Where the supremum is considered overall finite and countably infinite partitions α. However, in absence of a feasible partition known as *generating partition* [38] that contains all information of the system, it is difficult to determine KS Entropy.

2.5.1 Relationship Between KS Entropy and Ordinal Pattern Entropy Methods

Keller et al. [6] in their study proved the inequality relationship between KS Entropy and ordinal pattern entropy methods. In this sub-section, we discuss the proof in terms of PE, but the concept is valid in the general context of ordinal pattern methods.

We again, consider a dynamical system represented as $(\Omega, \mathcal{B}, P, T)$. Then the Permutation Entropy for the system is defined as :

Definiton 1. *For* $m \in N, PE^X(m) = \frac{1}{d} H(\alpha^X(m))$ *is called the Permutation Entropy of order m with respect to* X. *Moreover, by the Permutation entropy with respect to* X *, we understand* $PE^X = \lim sup_{m \to \infty} PE^X(m)$

The above definition of PE is derived from [7]. Given that if T is a piece-wise monotone interval map, then $PE^{id} \leq KS$ holds. Where, Ω is and interval and *id* is identity on Ω. If, we have well behaved increments of entropy of successive ordinal partitions such that the following exists,

$$\lim_{m\to\infty}(H(\alpha^X(m+1)) - H(\alpha^X(m)))$$

then by using Stolz–Cesàro theorem, we get the desired inequality:

$$\begin{aligned}
KS &\leq \lim_{m\to\infty} inf(H(\alpha^X(m)_2) - H(\alpha^X(m))) \\
&\leq \lim_{m\to\infty}(H(\alpha^X(m+1)) - H(\alpha^X(m))) \\
&= \lim_{m\to\infty}(H(\alpha^X(m)) - H(\alpha^X(m-1))) \\
&= \lim_{m\to\infty}\frac{1}{m}\sum_{i=1}^{m}(H(\alpha^X(i)) - H(\alpha^X(i-1))) \\
&= \lim_{m\to\infty}\frac{1}{m}H(\alpha^X(i))) = PE^X,
\end{aligned}$$

The above inequality sheds some light on the relationship between KS Entropy and all the ordinal pattern-based entropy methods.

Chapter 3

Reverse Weighted Dispersion Entropy

This chapter presents a novel pattern-based entropy method: Reverse Weighted Dispersion Entropy (RWDE). In section 3.2, we theoretically explain the RWDE algorithm, and then in section 2.5 we prove that RWDE provides an upper bound of the intrinsic complexity of time series.

3.1 Introduction

All the entropy methods discussed in the previous chapter: Weighted Permutation Entropy (WPE), Dispersion Entropy (DE), Reverse Permutation Entropy (RPE), and Reverse Dispersion Entropy (RDE); are derived from Permutation Entropy (PE) by adding an extra characteristic of time series for computing entropy. In this section, we introduce Reverse Weighted Dispersion Entropy (RWDE), which incorporates all the characteristics of the aforementioned entropy methods in a single entropy method. Therefore, RWDE takes into account four characteristics of time series for computing entropy: ordinal patterns, amplitude, variance, and distance information. As a new complexity measure for analyzing time series, RWDE takes PE as its theoretical basis and combines the advantages of WPE, DE, and RDE. The flow chart of PE and RWDE is shown in Figure 3.1. We can observe similarities between steps for both the entropy methods.

3.2 Algorithm

The specifics of the steps (as shown in Figure 3.1) of RWDE are as follows:

1. *Mapping time series to c classes*:
   Consider a time series $X = \{$x(i), i=1,2,...,N$\}$ with N values. We map X to $Y = \{$y(i), i=1,2,...,N$\}$ using NCDF, where $y(i)$ ranges from 0 to 1. Then, We map Y to $Z =$

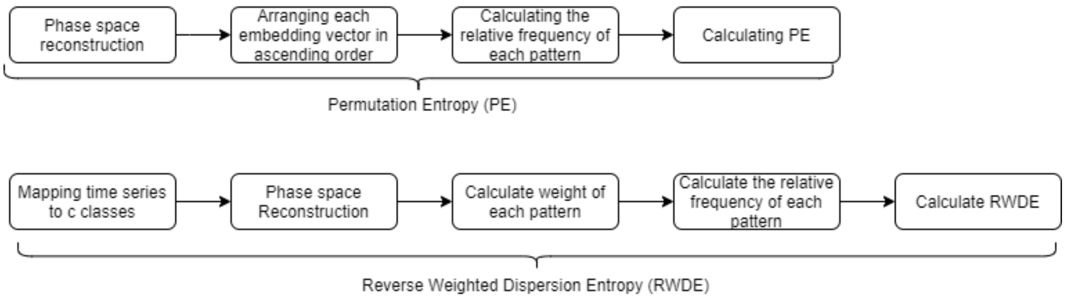

Phase space reconstruction
Arranging each embedding vector in ascending order
Calculating the relative frequency of each pattern
Calculating PE
Permutation Entropy (PE)
Mapping time series to c classes
Phase space Reconstruction
Calculate weight of each pattern
Calculate the relative frequency of each pattern
Calculate RWDE
Reverse Weighted Dispersion Entropy (RWDE)

Figure 3.1: Flow chart of Permutation Entropy (PE) and Reverse Weighted Dispersion Entropy(RWDE). Figure inspired from [48].

{z(i), i=1,2,...,N} using a linear mapping function $round(c \cdot y(i) + 0.5)$. Where c is the number of classes and $z(i)$ is a positive integer in the range $[1, c]$. This step exactly same as the first step of Dispersion Entropy.

2. *Phase space reconstruction*:
   We construct Z into L embedding vectors with the same time delay τ and embedding dimension m, respectively. Here we consider that time delay is unity, i.e., $\tau = 1$. The matrix consisting of all embedding vectors can be represented as follows:

$$\begin{bmatrix} z_1 & z_2 & z_3 & \cdots & z_{1+(m-1)} \\ z_2 & z_3 & z_4 & \cdots & z_{2+(m-1)} \\ & & \cdots\cdots\cdots\cdots & & \\ z_i & z_{i+1} & z_{i+2} & \cdots & z_{i+(m-1)} \\ & & \cdots\cdots\cdots\cdots & & \\ z_L & z_{L+1} & z_{L+2} & \cdots & z_{L+(m-1)} \end{bmatrix}$$

   Where the number of embedding vectors L is equal to $N - (m-1)$

3. *Calculate the relative weight of each pattern*:
   The amplitude of the switches plays a key role in predictability, so weighting the scale of ordinal changes is important for quantifying predictive structure accurately. To accomplish this, RWDE takes into account the weight of a permutation:

$$\omega(x_i^m) = \frac{1}{m} \sum_{j=i}^{i+m} (x_j - \bar{x}_i^m)^2$$

   where x_i^m is a sequence of values $x_i, \ldots, x_{i+m}$ or the embedding vector of length m and $\bar{x}_i^m$ is the arithmetic mean of those values.

4. *Calculate the relative weighted frequency or probability of each pattern*:

Since the embedding dimension and the number of classes are m and c respectively, there exist c^m possible patterns. Each of the embedding vectors can be mapped to a dispersion pattern π. Using the calculated weights in the previous step, we calculate the weighted probability of each pattern as follows:

$$P_w(\pi) = \frac{\sum_{i \leq T-m} \omega(x_i^m) \cdot \delta(x_i^m, \pi)}{\sum_{i \leq T-m} \omega(x_i^m)}$$

5. *Calculate RWDE*:

RWDE is expressed as:

$$RWDE(X,m,c) = \sum_{i=1}^{c^m} (P_w(\pi_i) - \frac{1}{c^m})^2 = \sum_{i=1}^{c^m} P_w(\pi_i)^2 - \frac{1}{c^m}$$

We just take the squared Euclidean distance of the observed pattern frequencies from the uniform pattern frequencies $\frac{1}{c^m}$ in the space of all pattern distribution. when $P_w(\pi_i) = \frac{1}{c^m}$, the value of $H_{RWDE}(X,m,c)$ is 0 (minimum value). When there is only one dispersion pattern, that is $P_w(\pi_i) = 1$, the value of $H_{RWDE}(X,m,c)$ is $1 - \frac{1}{c^m}$ (maximum value). Therefore, the normalized RWDE can be expressed as:

$$RWDE_{NE}(X,m,c) = \frac{\sum_{i=1}^{c^m} P_w(\pi_i)^2 - \frac{1}{c^m}}{1 - \frac{1}{c^m}}$$

In Chapter 5, we redo the simulation experiments performed in RDE [48] and DE [62] research studies. For now, we reuse the recommended parameters in these studies: $m = \{2,3,4,5,6\}$ and $c = \{4,5,6,7,8\}$.

Example: Consider a time series of seven values:

$$x = \{9, 8, 1, 12, 5, 3, 1.5, 8.01, 2.99, 4, 1, 10\}$$

We set the order $m = 3$ and classes $c = 2$. First, we map the time series to c classes using NCDF function to get:

$$x = \{0.82, 0.75, 0.21, 0.94, 0.52, 0.05, 0.241, 0.75, 0.35, 0.43, 0.11, 0.87\}$$

Similar to DE we generate the partition space in step 2 and then we compute the relative weight of each dispersion pattern using $\omega(x_i^m) = \frac{1}{m} \sum_{j=i}^{i+m} (x_j - \bar{x}_i^m)^2$. In the fourth step we compute the weighted frequency of each pattern. We get the following distribution :

$$P(\pi_{11}) = 5.06, P(\pi_{12}) = 0, P(\pi_{13}) = 0.548,$$

$$P(\pi_{21}) = 22.25, P(\pi_{22}) = 0.25, P(\pi_{23}) = 0,$$

$$P(\pi_{31}) = 12.25, P(\pi_{32}) = 18.55, P(\pi_{33}) = 0.25$$

Finally, we compute the distance of the observed pattern frequencies from the uniform pattern frequencies:

$$RWDE(X,m,c) = \sum_{i=1}^{c^m} P_w(\pi_i)^2 - \frac{1}{c^m} \approx 0.249$$

In this Section 2.5, we established a relation relationship between PE and KS Entropy and showed that PE provides aa upper bound for KS Entropy. RWDE also provides an upper bound to KS Entropy, as RWDE is theoretically completely derived from Permutation Entropy.

Chapter 4

Context Of ING

In this study we focus on applying entropy methods for quantifying complexity of real-world cloud metrics. For this, we consider cloud metric data from a large scale private cloud operational at ING bank. This chapter presents the context of ING related to this study. Fist, we discuss some background on IT system monitoring. In section 4.2, we introduce the cloud platform used at ING. Further, in section 4.3 we discuss how the cloud platform at ING is monitored and how system metrics are collected. Finally, in section 4.4 we present the different time-series system metrics used further in this study.

4.1 Background on IT System Monitoring

Tech organizations heavily invest money and efforts in IT infrastructure or cloud services to ensure high-quality services to their customers. To ensure stability and reliability of services it is important to monitor the state of cloud infrastructure [19][5]. Most importantly, parameters such as performance, power usage, security, etc. need to be monitored on a cloud platform to make sure Quality of Service parameters (QoS) specified in the Service Level Agreement (SLA) are met [40]. Cloud monitoring is important for i) managing hardware and software resources; ii) incident management [1].

Incident management is an important aspect of cloud monitoring and QoS. Generally, when a cloud monitoring tool detects an incident or service violation it sends an alert to the on-call engineer. After an alert is raised, the on-call engineer needs to find the root cause of the incident and fix the problem quickly. Often finding the root cause of incidents is a difficult problem and takes time, which leads to service downtime. Many times, engineers simply restart the service component as a quick fix, and the root cause is analyzed later. Some of the popular cloud monitoring tools such as Amazon CloudWatch, Netdata.cloud, and Datadog automate such procedures. Lou et al. [51], in their case study at Microsoft, proposed three ways to combat IT incidents:

1. Identification of incident alerts from system metrics
2. Mining suspicious execution patterns from logs

3. Leveraging previous actions for similar incidents

In this study, we focus on identifying informative system metrics for applications such as IT incident prediction. Raising incident alerts by monitoring system metrics leads to early detection and helps in reducing the mean time to restore service.

System metrics of a cloud platform are raw measurements of resource usage of systems in the cloud. System metrics are imprints of behavior and health of cloud platform. They are typically collected from operating systems, applications, and networks. Examples include, CPU load, disk read operations, number of request to an application per second, number of packets received by system etc. Figure 4.1 shows different system metrics collected across different layers of a cloud stack such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) [1].

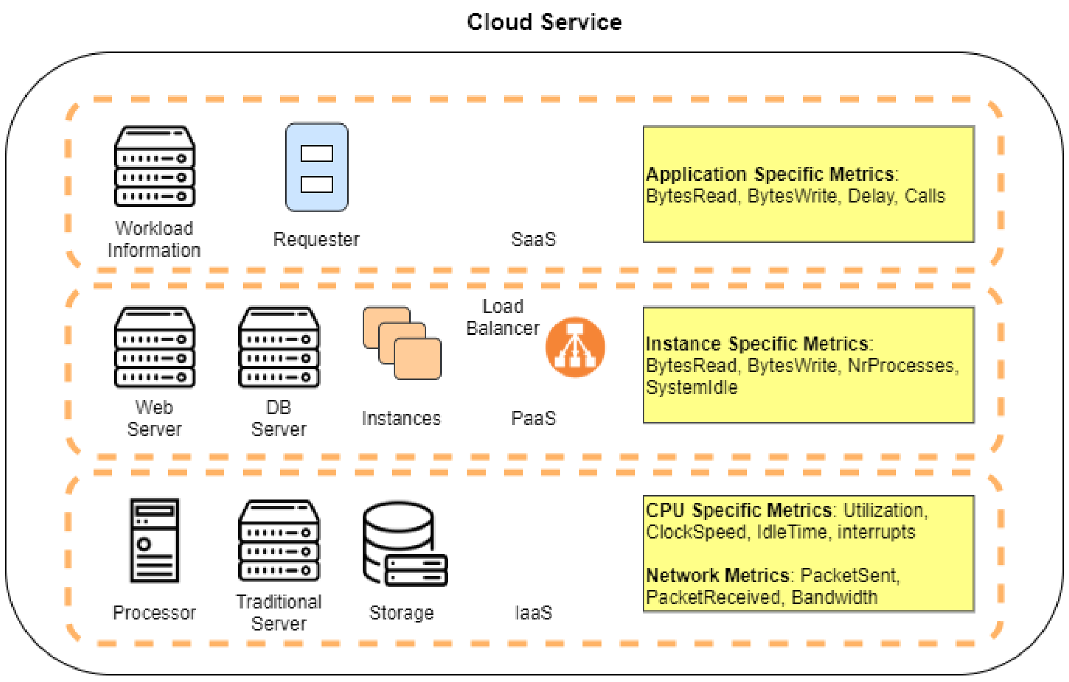

Cloud Service
Workload Information
Requester
SaaS
Application Specific Metrics: BytesRead, BytesWrite, Delay, Calls
Web Server
DB Server
Instances
Load Balancer
PaaS
Instance Specific Metrics: BytesRead, BytesWrite, NrProcesses, SystemIdle
Processor
Traditional Server
Storage
IaaS
CPU Specific Metrics: Utilization, ClockSpeed, IdleTime, interrupts
Network Metrics: PacketSent, PacketReceived, Bandwidth

Figure 4.1: System metrics collected across different cloud platform layers [1].

4.2 ING Private Cloud (IPC)

In 2015, ING started to build a private cloud known as the ING Private Cloud (IPC). The idea behind the IPC was to provide an ideal environment for the standardization of application platforms, data analytics, API platforms, and much more. The goal was to streamline processes by introducing simpler, more automated processes on top of the standardized infrastructure, thus making the cloud a key enabler for simplified banking systems.

In the short term, the Private Cloud has helped ING to rapidly scale standard platform services resulting in a faster time to market for new initiatives and increased security reliability through a simpler, integrated design. The Private Cloud also provided standardization of the infrastructure services catalog, flexible on- and off-boarding of workloads, a significantly lower cost of operation, and the creation of a platform that responds quickly to changing customer and market demands.

IPC provides self-service capabilities for engineers to order, build, and manage their environment. Its purpose is to provide an automated self-service IT delivery process. The IPC introduced a new relationship between consumer, i.e., local application team or DevOps team and provider, i.e., the Global Cloud Engineering team, in which the consumer is empowered with larger responsibilities. At present, IPC consists of more than 30 services deployed on over ten thousand Virtual Machines (VM) in 13 countries.

4.3 Monitoring in IPC

Monitoring in the ING Private Cloud is a shared responsibility that allows the DevOps team to be in control of their stack while the Infrastructure is monitored by the Global Cloud Engineering team. Monitoring is essential to understand the health state of ING's infrastructure and applications. Besides health state monitoring it is also important to diagnose and analyze incidents that occurred and metric values originating from the managed platform in order to determine the root cause of failures and to predict potential performance/capacity bottlenecks. However, thousands of metrics are collected by monitoring applications. The problem we want to solve is identifying the most informative metrics among them. The more informative metrics can be stored for a long period of time, while less informative metrics can be stored for relatively smaller period of time.

The IPC provides standard Linux and Windows services via the ING Private Cloud Self-Service Portal, including resource and availability monitoring. For Linux machines, vROps is included while SCOM [1] is included for Windows machines. A tool like vROps[2] of VMWare monitors the different infrastructure layers for both Windows and Linux. Application DevOps teams are responsible to monitor the layers that are not included in the service by using monitoring tools. In this study, we focus on system metrics collected from IPC. The IPC uses a range of monitoring tools like Kibana[3], Graphite[4] and Grafana[5]. The IPC uses monitoring tools to collect the following type of data at different layers :

[1]https://docs.microsoft.com/en-us/system-center/scom/welcome?view=sc-om-2019
[2]https://www.vmware.com/nl/products/vrealize-operations.html
[3]https://www.elastic.co/kibana
[4]https://graphiteapp.org/
[5]https://grafana.com/

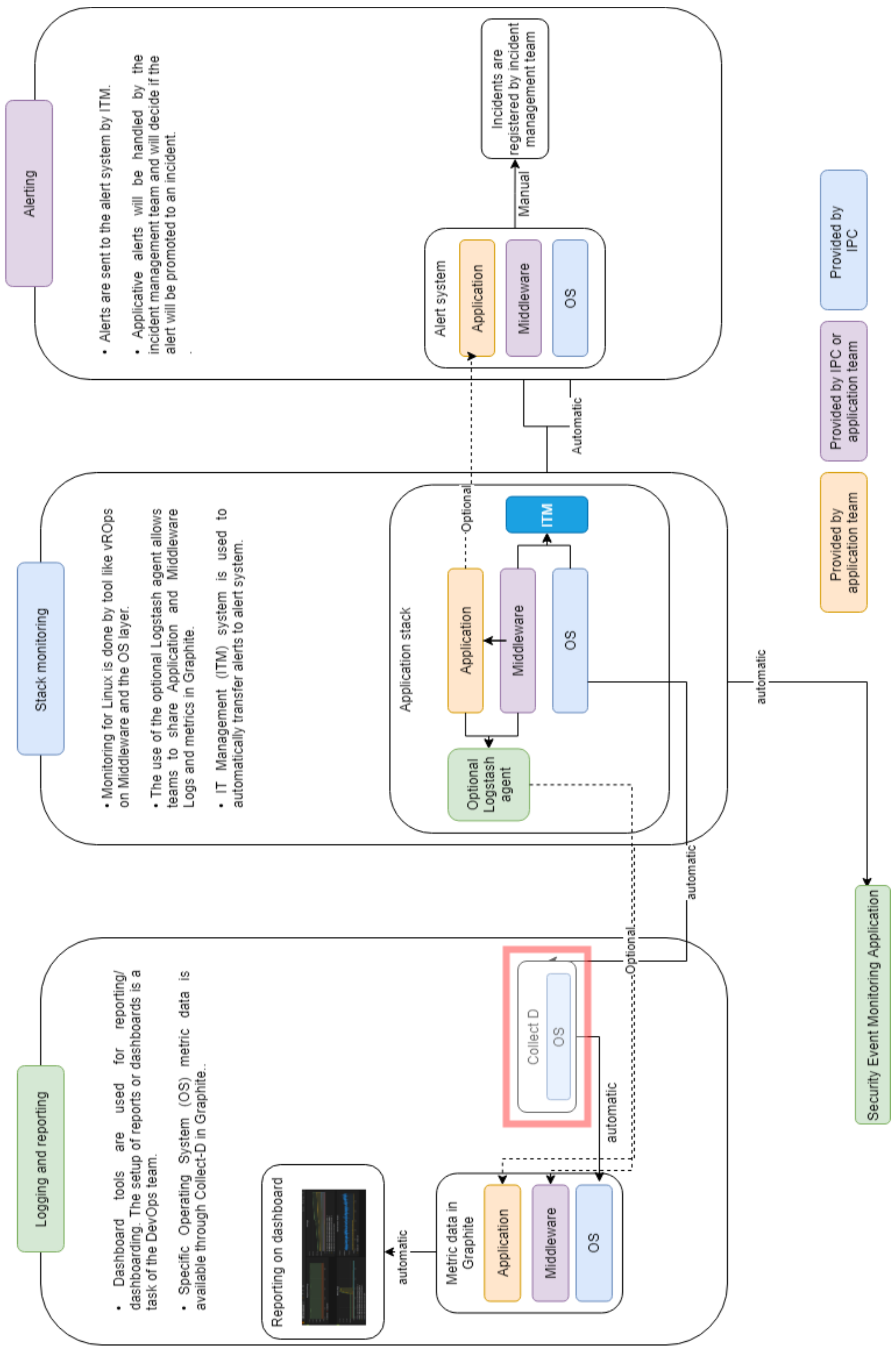
- **Log Data**: The monitoring solution offers the DevOps team log analytics functionality for their application logging data. It helps to gather unstructured log data and recognize application log patterns and trends over a selected period. These log patterns can be used as a baseline, which then can help visualize unexpected behavior. The next great value is that the monitoring solution can simplify and reduce the time for root cause analysis because all application logs are accessible for analysis. The log analytic functionality itself can be accessed by a graphical and REST interface such as *Kibana*. The underlying monitored technical products are known as the *Elastic Stack* and collected data is known as the *Elasticsearch data*.

- **Cloud Metrics**: The monitoring solution also offers the DevOps team metric data graphing and analytics functionality to monitor the application and infrastructure metrics. The collected metrics are stored in a data lake for a specified period of time. On the first day, the metrics are collected at a small time resolution/interval. Over time, the metrics are aggregated with a granularity of minutes and hours. Most importantly, the gathered metrics can be used to recognize application metric patterns and trends over a selected period. These patterns can be used as a baseline, which then can help visualize unexpected behavior. The metric gathering functionality itself can be accessed by a graphical dashboard and web interface such as *Grafana*.

Figure 4.2 represents a simplified overview of the monitoring architecture for a deployed VM at ING. Due to confidentiality reasons, only relevant parts and an abstract overview of the architecture is shown in the Figure 4.2.

4.4 Collected System Metrics

In this study, we primarily focus on time-series system metrics or cloud metrics and attempt to quantify the complexity of these metrics. The raw system metrics are collected and stored by Graphite. We focus on system metrics collected by *CollectD*, highlighted with a red box in Figure 4.2. CollectD is a daemon that periodically collects application and system performance metrics. For this study, we use seven system metrics collected for a year from 4 random VM's. The table 4.1 describes the different system metrics present in the database used in this study. The monitoring tool collects some more metrics but they are redundant to the metrics the shown in Table 4.1. For instance, the tool also collects *disk_time_read* which represents the average amount of time it took to do a read operation, which is very much dependent on the number of disk operations done at a time.



- Alerts are sent to the alert system by ITM.
- Applicative alerts will be handled by the incident management team and will decide if the alert will be promoted to an incident.

- Monitoring for Linux is done by tool like vROps on Middleware and the OS layer.
- The use of the optional Logstash agent allows teams to share Application and Middleware Logs and metrics in Graphite.
- IT Management (ITM) system is used to automatically transfer alerts to alert system.

- Dashboard tools are used for reporting/dashboarding. The setup of reports or dashboards is a task of the DevOps team.
- Specific Operating System (OS) metric data is available through Collect-D in Graphite..

Figure 4.2: Overview of monitoring tools deployed at different layers of infrastructure for collecting system metrics.

Table 4.1: Description of system metrics collected and stored in database used for this study

Measurement Component	Measurement Type	Description
CPU	cpu.system	Percent CPU time spent running in the kernel. This value reflects how often processes are calling into the kernel for services (e.g to log to the console). A sustained high value for this metric may be caused by: 1) A process that needs to be re-written to use kernel resources more efficiently, or 2) A userspace driver that is broken.
	cpu.user	Percent CPU time spent running in userspace. If this value is high: 1) A process requires more CPU to run than is available on the server, or 2) There is an application programming error which is causing the CPU to be used unexpectedly.
	cpu.idle	Percent CPU time spent not in any other state.
Disk	disk.io_time	Amount of time spent doing IO in ms.
	disk_ops.read	The number of disk read operations.
	disk_ops.write	The number of disk write operations.
Network	packets.tx	Count of packets transmitted by the network interface.

Chapter 5

Evaluation on Synthetic Time Series

In Chapter 2, we discussed ordinal pattern based entropy methods and in Chapter 3 we introduced a novel entropy method called Reverse Weighted Dispersion Entropy (RWDE). This chapter presents disparate experiments to facilitate the empirical evaluation of RWDE. We compare RWDE with other entropy methods using simulated synthetic time series data.

5.1 Experimental Setup

In Chapter 2, we introduced Permutation Entropy (PE) and all the entropy methods derived from PE. In this section, we try to compare all the before-mentioned entropy methods with the novel entropy method proposed in this study, i.e., RWDE . To evaluate the performance of RWDE, we set-up 4 simulations of synthetically generated time-series data. In each simulation, we try to target a specific property of entropy methods such as sensitivity, stability, robustness. These simulation experiments follow the exact same setup as experiments performed to compare entropy methods in Reverse Dispersion Entropy (RDE) [48].

In particular, we replicate the experiments performed in [48] and use these experiments as a standard framework to evaluate RWDE's performance. However, we do not replicate simulation 5 from [48], as we believe it is fundamentally similar to simulation 4. We also add Spectral Entropy (SE) [54] to the pool of entropy methods being compared as a representative of non ordinal pattern based entropy methods. In simple terms, SE can be defined as the Shannon Entropy of Power Spectral Density (PSD) [67]. Moreover, disparate from [48], we perform an independent T-test as a significance test in simulation 1 and 2.

Open-source Contribution. All the ordinal pattern based entropy methods (PE and all the entropy methods derived from PE) used in this study are made available as an open-source contribution, in the form of a GitHub repository named OrdinalEntroPy [55]. OrdinalEntroPy is a Python 3 package providing several time-efficient, ordinal pattern based entropy algorithms for computing the complexity of one-dimensional time-series. Moreover, all the simulation experiments performed in this chapter are available as python code on GitHub [56]. The following simulation experiments in this section are focused on characterizing RWDE. With these experiments we try to answer our first research question **(RQ1)**:

RQ1: How does RWDE perform compared to other entropy methods?

5.2 Simulation 1: Sensitivity

In our first simulation, we assess the *sensitivity* of RWDE. The sensitivity of an entropy method can be defined as the ability to quickly detect or respond to slight changes in signals.

Experimental Setup. To evaluate the sensitivity of RWDE method, we consider a mutation signal similar to the one considered in [48] [45]. Mathematically, the synthetic signal (mutation signal) is produced as follows:

$$\begin{cases} y = x + s \\ x = \begin{cases} 50, & \text{if } t = 0.498 \\ 0, & \text{otherwise} \end{cases} \\ s = randn(t) & (0 \leq \text{ t } \leq 1) \end{cases}$$

```python
import numpy as np
np.random.seed(42)

def simulation1():
  # initialize time series
  time_series = np.zeros(1000)
  # generate random noise
  noise = np.random.normal(0, 1, 1000)
  time_series[498] = 50
  time_series = time_series + noise
  return time_series
```

Listing 5.1: Python code for Simulation 1.

In the above equations, y represents the final synthetic signal which is sampled at 1 kHz frequency. That means a data point is recorded at every 0.001-second interval for 1 second (total 1000 data points) to obtain the time series. The synthetic signal y is composed of white Gaussian noise s and deterministic impulse signal x. The Python code for generating the synthetic signal y is shown in Listing 5.1.

The resultant signal y or simulation 1 is shown in Figure 5.1. To calculate the seven entropies (SE, PE, WPE, RPE, DE, RDE, and RWDE), we use a sliding window of the length of 80 data samples with a jump of 10 samples for the next window (i.e., 70 overlapping samples). We set the embedding dimension m to 2 and delay τ to 1 for all the entropy methods. Note that in this simulation, we do not use the recommended/best embedding dimension (but we will do so in simulation 2 described in the next section). We want to test the ability of entropies to detect the mutation signal, independent of parameters. For DE, RDE, and RWDE we set the number of classes c to 6. We also perform an independent T-test on mean entropy values of different windows to confirm a difference between entropy values when the entropy method encounters a spike.

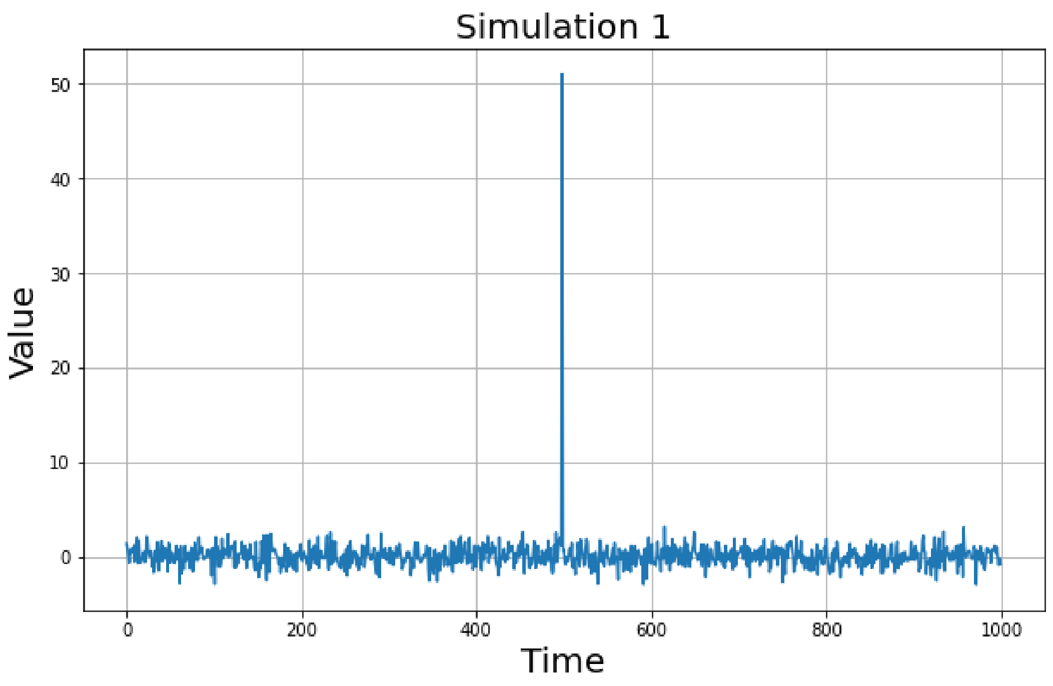


Figure 5.1: The time-domain waveform of mutation signal y in simulation 1.

Results. Figure 5.2 shows the seven calculated entropies of simulation 1. Table 5.1 shows the entropy value of the six methods for windows 41 to 50 (windows consisting of impulse signal x are 42-49). We can observe from Figure 5.2 and Table 5.1 that SE, DE, RDE, and RWDE show a significant increase and decrease when the windows contain impulse signal x. For the non-reverse entropy methods, a high entropy value (close to 1) indicates a complex time series, which is the normal convention for interpreting entropy values. Whereas, for reverse entropy methods, a high entropy value (close to 1) indicates a deterministic or non-complex time series and a low value (close to 0) indicates a complex time-series. For further comparison, we compare the means of the seven entropies and their variation ratios as shown in Table 5.2. A is the mean of the 82 windows excluding the windows consisting of the impulse signal, B is the mean of 8 windows consisting of the impulse signal, and the variation ratio is defined as the ratio of maximum to minimum of A and B. We can observe from Table 5.2, for SE, PE, WPE, and RPE the variation are very small (from 1.0004 to 1.16). For DE, RDE, and RWDE there is a significant difference between A and B, since the variation ratios are greater than 2 for all of them.

Table 5.1: The seven entropies in the windows from 41 to 50 of simulation 1.

Window	41	42	43	44	45	46	47	48	49	50
SE	0.047	0.456	0.456	0.458	0.458	0.458	0.454	0.458	0.458	0.090
PE	0.997	0.994	0.990	0.997	0.999	0.998	0.998	0.997	0.997	0.985
WPE	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.999	0.994
RPE	0.004	0.007	0.012	0.004	0.000	0.001	0.001	0.004	0.004	0.019
DE	0.919	0.415	0.382	0.408	0.428	0.396	0.442	0.455	0.433	0.965
RDE	0.016	0.287	0.330	0.288	0.263	0.304	0.252	0.225	0.272	0.006
RWDE	0.055	0.456	0.461	0.457	0.457	0.459	0.457	0.456	0.457	0.039

To back our results statistically, we also include an independent T-test performed on A and B for each entropy method in Table 5.2. As shown in Table 5.2, SE, DE, RDE, and RWDE have p-values less than 0.01. This Indicates that averages of entropy values without impulse signal are significantly different from the averages of entropy values with impulse signal for these entropy methods. Hence, they are sensitive to information present in these time series.

The simulation results show that SE, DE, RDE, and RWDE are sensitive and can detect information in mutation signals.

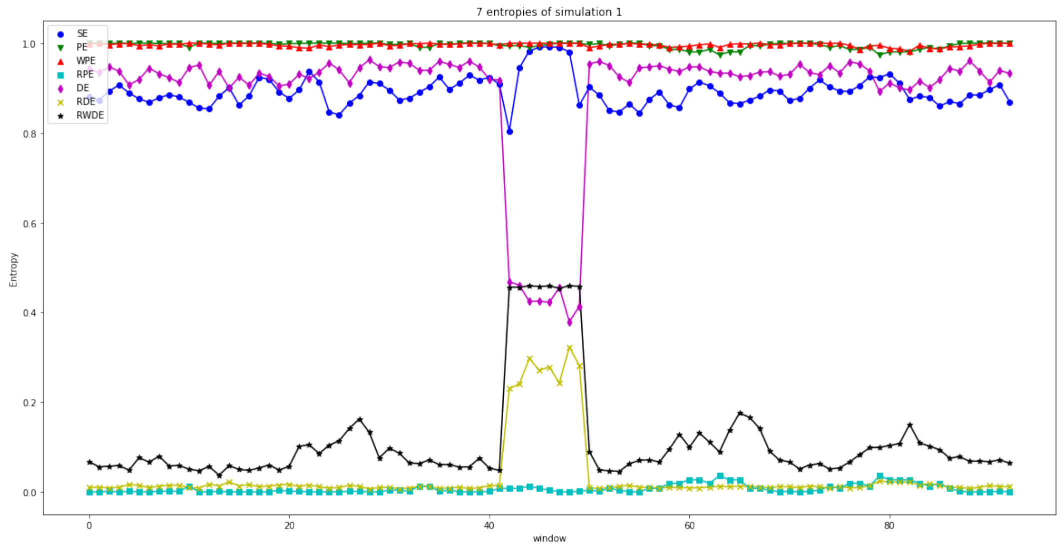


Figure 5.2: Comparison of the seven calculated entropies of simulation 1.

Table 5.2: The means of the seven entropies, variation ratios and their independent T-test.

Parameters	SE	PE	WPE	RPE	DE	RDE	RWDE
A (Means of 82 windows)	0.8859	0.9972	0.9966	0.0038	0.9370	0.0110	0.0698
B (Means of 8 windows)	0.9435	0.9967	0.9999	0.0044	0.4205	0.2782	0.4579
Max(A,B)/Min(A,B)	1.0649	1.0004	1.0033	1.1638	2.2282	25.0840	6.5588
independent/Welchs T-test	5.804e-8	0.7055	0.0262	0.7048	4.422e-87	6.931e-85	6.427e-64

5.3 Simulation 2: Using Recommended Parameter Settings

In simulation 1, we did not use the recommended embedding dimension for calculating the entropy with WPE, DE, RDE, and RWDE, because we wanted to check the ability of entropy methods to detect mutation signals independent of parameters. In simulation 2, we use the recommended parameters considered in [48].

Experimental Setup. We consider the same mutation signal considered in simulation 1 (shown in Figure 5.1). The number of data points and the window size are kept the same. The embedding dimension m is changed from 2 to 4. Also, we set the number of classes c to 4, keeping in mind the length of window T (80) over which we compute the entropy should be greater than $c^m (= 4^4 = 64)$. That is, $T > c^m$.

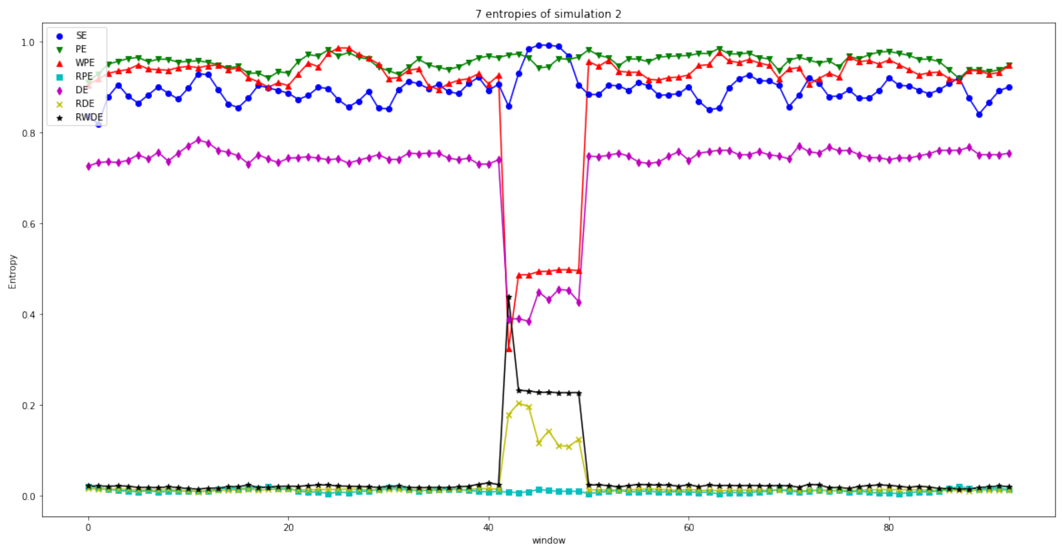


Figure 5.3: Comparison of the seven calculated entropies of simulation 2.

Table 5.3: The means of the seven entropies and variation ratios.

Parameters	SE	PE	WPE	RPE	DE	RDE	RWDE
A (Means of 82 windows)	0.8859	0.9504	0.9229	0.0126	0.7479	0.0130	0.0224
B (Means of 8 windows)	0.9435	0.9361	0.4686	0.0149	0.4088	0.1434	0.2577
Max(A,B)/Min(A,B)	1.0649	1.0152	1.9691	1.1775	1.8291	10.9786	11.4705
independent/Welchs T-test	9.03e-10	0.4138	9.63e-68	0.3527	2.29e-78	1.76e-52	1.04e-49

Results. The seven calculated entropies with new parameters on simulation 2 are shown in Figure 5.3. Similar to observations in simulation 1, DE shows a significant drop in entropy value, while RDE and RWDE show a steep increase in entropy values. However, unlike simulation 1, with the recommended parameters WPE was able to detect the mutation signal as well. This observation is in line with results from Li et al. [48]. The results indicate that WPE is sensitive to embedding dimension for detecting mutation signals.

Similar to the Table 5.2, Table 5.3 shows the seven entropies and their variation ratios. A is the mean of the 82 windows excluding the windows consisting of the impulse signal, B is the mean of 8 windows consisting of the impulse signal, and the variation ratio is defined as the ratio of maximum to minimum of A and B. We can observe from Table 5.3, For PE and RPE the variation are very small (from 1.0152 to 1.177). For WPE and DE, the variation ratios are close to 2 (1.9691 and 1.8291 respectively). Finally, for RDE and RWDE there is a significant difference between A and B,and the variation ratios are 10.9786 and 11.4705, respectively. Spectral Entropy remains unchanged, as it is not dependent on the embedding dimension. The independent T-test on A and B for each entropy method also show that under the recommended parameters WPE, DE, RDE, and RWDE can detect the mutation signals ($p\text{-}value \leq 0.001$).

The simulation results show that under the recommended parameters only WPE, DE, RDE, and RWDE can detect the mutation signals. Moreover, RWDE has the highest variation ratio and has a better performance compared to the other six entropies in detecting the mutated signal.

5.4 Simulation 3: Robustness to noise

Most of the real-time signals or ”real world” signals are noisy. For an entropy method to be reliable for computing complexity of time series, it should be robust to noise.

Experimental Setup. In simulation 3, we verify the robustness of RWDE to noise. We again consider the same mutation signal as in simulation 1 (i.e., Figure 5.1). However, we modify the signal y with different SNRs (Signal to Noise Ratio) by adding white Gaussian noise to x. All other parameters with respect to the signal are the same as in simulation 1 and 2. For the entropy methods, we set the embedding dimension *m* to 3 and the number of classes to 6 for DE, RDE, and RWDE.

Results. Figure 5.4 shows the seven entropies of synthetic signal *y* varied with different SNRs from -10 dB to 10 dB. Each data point on each entropy line is the mean of 100 calculations for each SNR. As the SNR is increased (i.e., the noise decreases) at each step, we expect the entropy methods to indicate decreasing complexity of the signal in the resulting entropy value.This is because the amount of information increases as the noise decreases. We can observe in Figure 5.4, the SE, PE and RPE show no change entropy in value for varying SNRs, and are very close to 0 and 1 respectively. On the other hand, with the increase in SNR, the entropy values of DE and WPE monotonically decrease. Lastly, we can observe that the values of RDE and RWDE monotonically increase with the increase of SNR. Note that, in this simulation we expect to observe a decrease in the influence of noise on complexity as the SNR increases.

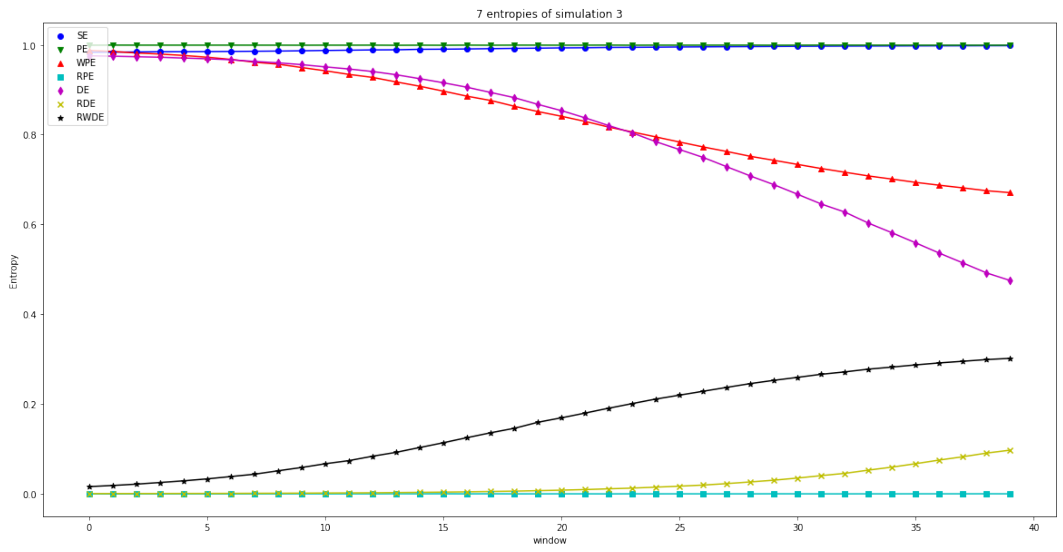


Figure 5.4: The six entropies of synthetic signal *y* under increasing signal-to-noise ratios (SNRs). As the window moves to right, the SNR of the signal increases.

For further comparison of the seven entropies, Table 5.4 shows the entropies of seven methods under varied SNRs of signal along with their variation ratios. A and B are the entropies of each method under 10 dB and -10 dB respectively. The last row shows the variation ratio for each method which is Max(A, B)/Min(A, B); the ratio of maximum to minimum of A and B. As shown in the above Table 5.4, for SE, PE, WPE and RPE, there are low differences between A and B, and hence the variation ratios are close to 1; 1.0001,

Table 5.4: The six entropies under -10 dB and 10 dB and their variation ratios.

Parameters	SE	PE	WPE	RPE	DE	RDE	RWDE
A (10dB)	0.9980	0.9989	0.6707	0.0007	0.4734	0.0981	0.3023
B (-10dB)	0.9834	0.9990	0.9873	0.0006	0.9758	0.0011	0.0161
Max(A,B)/Min(A,B)	1.0148	1.0001	1.4720	1.1076	2.0609	85.1351	18.7321

1.472, and 1.107 respectively; RDE and RWDE have a high variation ratio of 85.13 and 18.73 respectively.

The simulation results show that only DE, RDE, and RWDE show significant variation to varying SNR's. RWDE performs way better than PE, WPE, RPE, and DE. However, RDE can reflect the difference better under different SNR's than RWDE and the other five entropies.

5.5 Simulation 4: Stability

Lastly, in simulation 4, we test the stability of RWDE with synthetic signals. For an unvarying time series, we expect an entropy method to converge to an entropy value quickly and showcase low variance.

Experimental Setup. For simulation 4, we consider a cosine signal wave of different lengths with a frequency of 100 Hz (i.e., 100 data points for each second). A cosine signal is purely deterministic, hence not complex. For all the six entropy methods, we set the embedding dimension *m* to 3 and the number of classes *c* to 6. Figure 5.5 shows the cosine signal with 100 Hz frequency. We consider a cosine signal with 12000 sampling points (120 seconds) similar to [48]. We do not consider Spectral Entropy in this simulation, as it is always constant for sinusoidal wave.

Results. Figure 5.6 shows the six entropies for the cosine signal of 100 Hz frequency. For the first entropy measure, we set the data length to 2,000 data points. Then, for every further entropy measure point, we add 100 data points until the 12,000th data point. As the cosine signal is purely deterministic and non-linear, we expect the entropy methods to converge to an entropy value quickly rather than deviating from an entropy value. Therefore, an entropy method with a lower standard deviation in the entropy measurements has better stability than an entropy method with a higher standard deviation.

In Figure 5.6, we can observe that six entropies deviate in varying degrees and converge with the increase of data length; the entropy values of WPE mostly range between 0.370 to 0.3875, the variation (maximum entropy - minimum entropy) of WPE (0.013) is of the same order of magnitude as that of PE (0.019) and RPE (0.026). Furthermore, the variation of DE (0.0068), RDE (0.0094) and RWDE (0.0034) are smaller than those of WPE, with RWDE having the least variation.

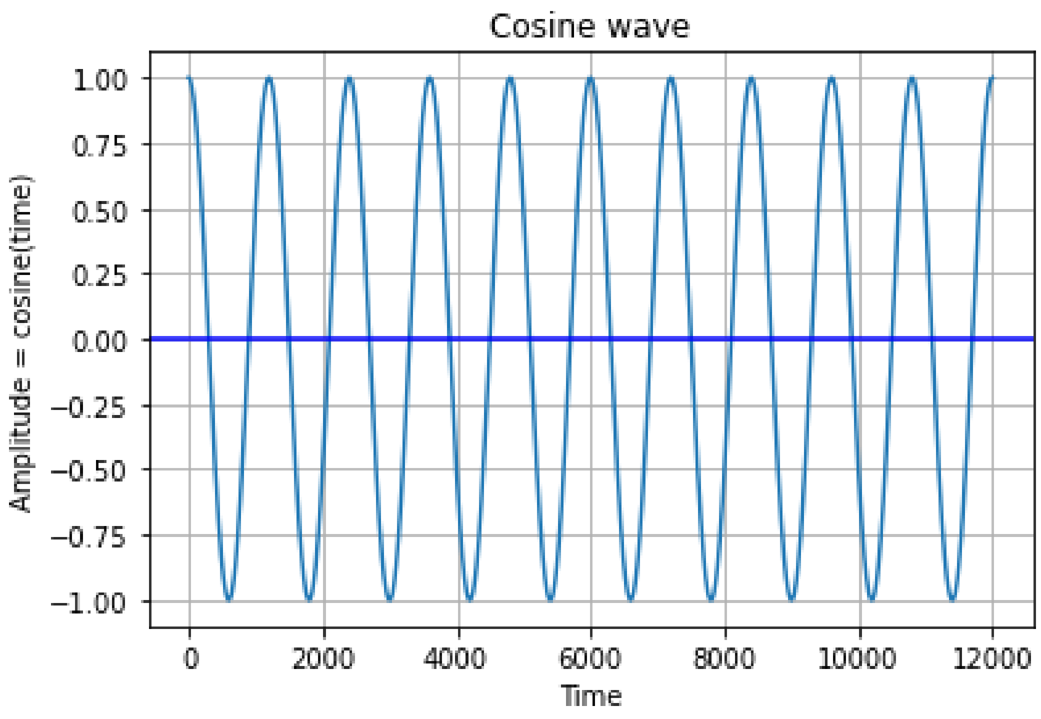


Figure 5.5: Cosine signal for simulation 4.

Table 5.5: The mean and standard deviation of six entropies for the cosine signal of different lengths.

Parameters	PE	WPE	RPE	DE	RDE	RWDE
Mean	0.3921	0.3853	0.4009	0.3371	0.1860	0.1666
Standard Deviation	0.0023	0.0028	0.0046	0.0010	0.0014	0.0006
Variation (Max-Min)	0.013	0.019	0.026	0.0068	0.0094	0.0034

The mean and standard deviation of six entropies for the cosine signal of different lengths are shown in the above Table 5.5.

As shown in results (Table 5.5), RWDE has the smallest standard deviation compared with the other five entropies. The stability testing results indicate that RWDE has better stability than the other five entropies under different lengths of cosine signal data.

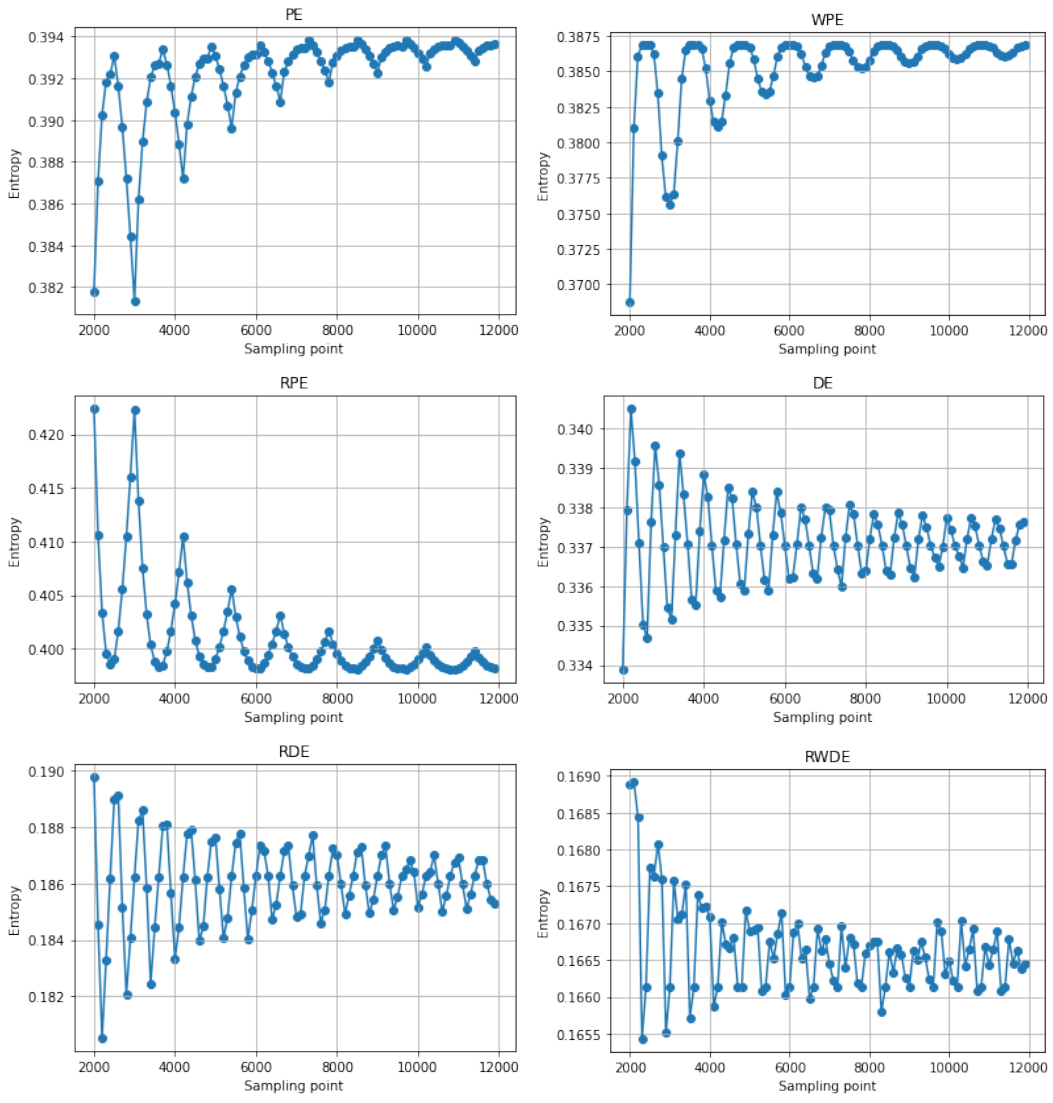


Figure 5.6: The six entropies of cosine signal with the 100 Hz Frequency.

Chapter 6

RWDE Application on ING System Metric Data

In Chapter 5, we compared RWDE with other ordinal pattern based entropy methods and established RWDE's dominance over other entropy methods. However, the data we considered in Chapter 5 was completely synthetic. In this chapter, we use all the ordinal pattern based entropy methods on cloud metric data from ING Private Cloud (IPC). In particular, we report the performances of the entropy methods on the cloud metric data and try to establish a relationship between entropy and forecasting error of various forecasting models on the cloud metric data. At ING, over ten thousand Virtual Machines (VMs) providing over 30 global services, are monitored continuously. As earlier explained in Section 4.3, Graphite collects metric data from applications and infrastructure. The Table 4.1 shows all the system metrics used in the experiments of this section.

6.1 Parameter Sensitivity of RWDE

All the ordinal pattern based entropy considered in this study have one thing common: embedding dimension m. All of them are depedent on embedding dimension and are sensitive to change in embedding dimension. Obviously, a better entropy method would be more insensitive to change in embedding dimension. In order to test the sensitivity of RWDE with respect to embedding dimension (m) we perform an experiment to compare the entropy values of the six entropy methods for different values of embedding dimension for a system metric. We use a system metric rather than a deterministic simulated time series to compare the entropy methods on real-world time series. With this experiment, we try to answer our second research question (**RQ2**):

RQ2. what is the effect of parameters on RWDE?

Experimental Setup. Figure 6.1, shows the recorded time-series data of disk write operations per minute of a week (7 days=10,080 minutes) for a particular virtual machine.

Further, to calculate the six entropies for this time series, we use a sliding window of the length of 1440 data samples (1,440 minutes = 1 day) with a jump of 360 samples (6 hours) for the next window (i.e., 1080 overlapping samples). Therefore, we get 24 different windows ((10080-1440)/360 = 24) on which we compute the entropy. We set the length of sliding window to 1 day, because we can see observe a daily seasonality in Figure 6.1.

For all the entropy methods, we vary the embedding dimension m in the range [3,4,5,6,7,8]. Also, we set the number of classes c to 3, keeping in mind the length of window T (1,440) should be greater than c^m.

We also perform Augmented Dickey Fuller (ADF) test [65] and KPSS (Kwiatkowski-

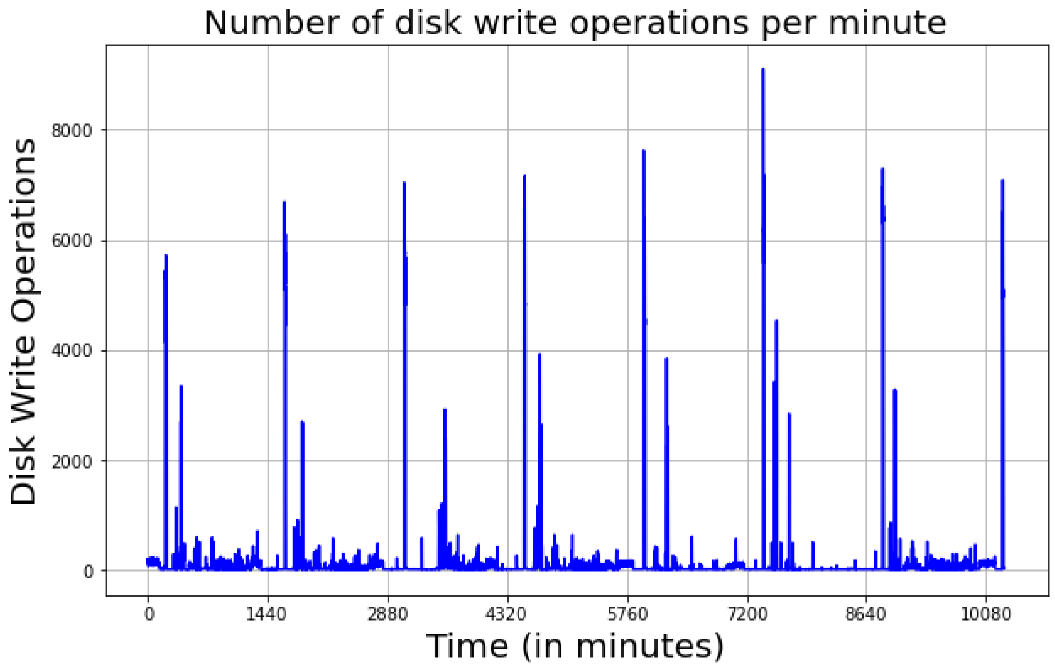


Figure 6.1: Time series of disk write operations per minute of a particular virtual machine in a certain week.

Phillips-Schmidt-Shin) Test [41] to understand if the system metric in consideration is stationary or non-stationary. Table 6.1 shows the result of the two aforementioned tests. In case of the ADF test, the null hypothesis is: A unit root is present in a time series. Unit root is a stochastic process in a time series. Presence of unit root results in a systematic pattern that is unpredictable. In case of KPSS test, the null hypothesis is: The time series is trend stationary. A trend-stationary time-series is a stochastic process from which an underlying trend can be removed, leaving a stationary process. The null hypothesis for the two tests are opposite to each other. Based on the results in Table 6.1, we cannot reject the null hypothesis in case of ADF test (p-value > 0.05), whereas we can reject the null hypothesis in case of the KPSS test (p-value < 0.05). Therefore, both tests suggest that the system metric

Table 6.1: Result of ADF test and KPSS test for disk write operations metric.

Results of Dickey-Fuller Test:	
Test Statistic	-2.344059
p-value	0.13020
Lags Used	56
Number of Observations Used	10080
Critical Value (1%)	-3.430492
Critical Value (5%)	-2.861603
Critical Value (10%)	-2.566803
Results of KPSS Test:	
Test Statistic	13.283094
p-value	0.010000
Lags Used	56
Number of Observations Used	10080
Critical Value (1%)	0.739000
Critical Value (5%)	0.463000
Critical Value (10%)	0.347000

Table 6.2: Entropy values of the six entropy methods for different values of embedding dimension m for the disk writes metric.

Entropy	Embedding Dimension (m)						Deviance =
Method	m=3	m=4	m=5	m=6	m=7	m=8	(max(e) - min(e))/min(e)
PE	0.9936	0.9860	0.9752	0.9312	0.8164	0.6795	0.4622
WPE	0.7938	0.6806	0.5800	0.4840	0.3963	0.3303	1.4032
RPE	0.0047	0.0037	0.0020	0.0012	0.0008	0.0007	5.7142
DE	0.1956	0.1842	0.1761	0.1693	0.0671	0.0643	2.0419
RDE	0.1690	0.1201	0.0929	0.0755	0.0540	0.0483	2.4989
RWDE	0.6987	0.6607	0.6253	0.5927	0.5611	0.5499	0.2705

is non-stationary. Similar to the disk write operations metric, most of the system metrics considered in this study are non-stationary. However, note that PE is applicable on all types of time series, be they deterministic, stochastic, stationary, or non-stationary [32][16].

Results. Figure 6.2, shows the six calculated entropies of disk write operations time series. We can observe that the entropy value change over time.

Table 6.2 shows the entropy values (e) of the six entropy methods for different values of embedding dimension for a disk write operations metric. The last column shows the deviance, which is simply the difference between maximum entropy value and minimum

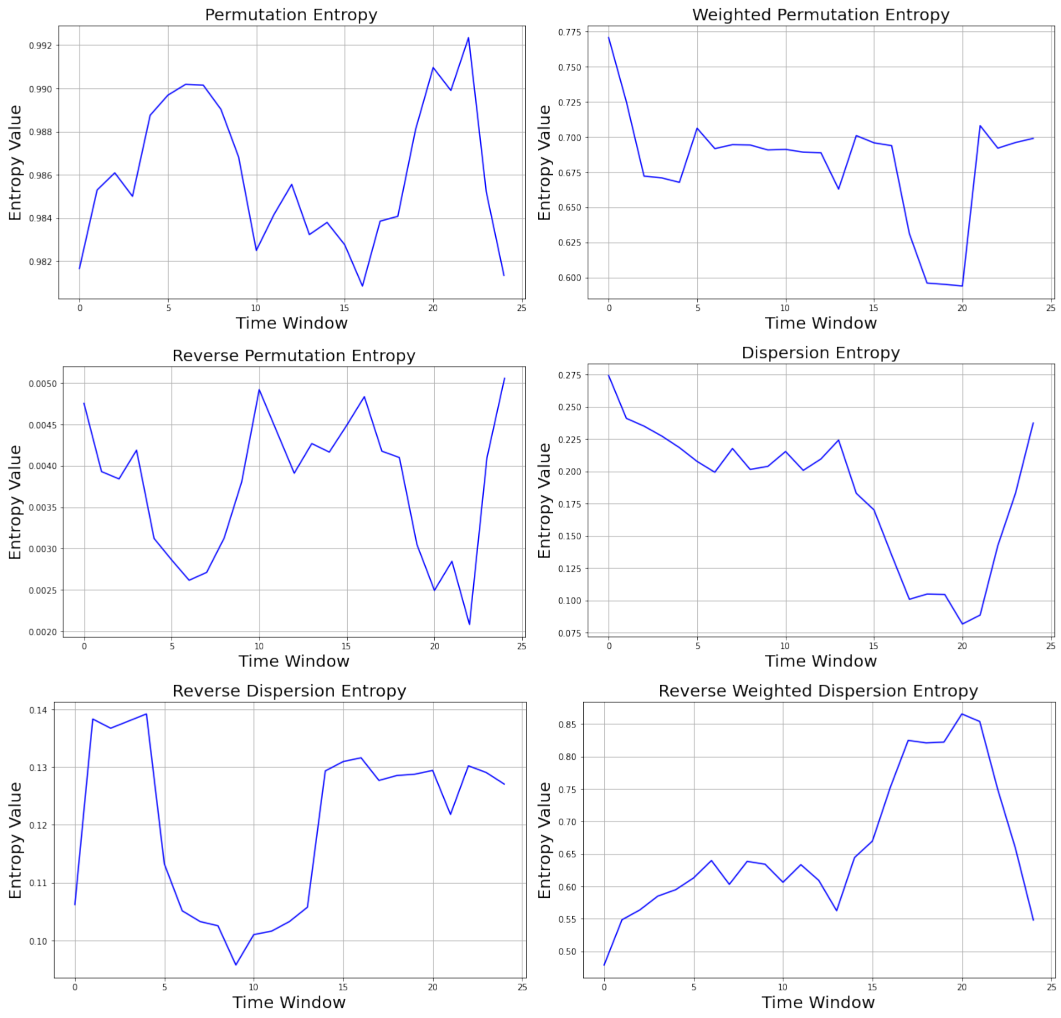


Figure 6.2: Six calculated entropies of disk write operations time series shown in Figure 6.1 for $m = 4$

entropy value divided by the minimum entropy value. We can observe that RWDE is least sensitive to change in embedding dimension.

6.2 Ranking Subset of System Metrics

In this section, we compare entropy values of RWDE with the forecasting error of the ARIMA method. ARIMA models are among the most widely used approaches to time series forecasting. ARIMA models aim to describe the autocorrelations in the data by combining autoregressive model and moving average model [34]. The relationship between entropy values and forecasting error can help us intuitively understand if RWDE can quantify the predictability of a time series. We try to get insights for our second research question **(RQ3)**, i.e., What is the relationship between intrinsic predictability and realized predictability?

The entropy of a system/time-series is high if the uncertainty with respect to its outcome is high, i.e., unpredictable. Similarly, the entropy of a system/time-series is low if the uncertainty with respect to its outcome is low, i.e., predictable. Therefore, in this section, we represent the mean entropy value as predictability percentage, i.e., predictability percentage = (RWDE*100)/1, because for normalized RWDE: $0 \leq \text{RWDE} \leq 1$. Moreover, the entropy values are highly influenced by the predictive structure in time series [7]. Therefore, RWDE can be used to rank and select system metrics based on predictability percentage for IT incident prediction tasks or forecasting tasks. Figure 6.3, shows the predictability percentage of a subset of system metrics. We arrange or rank the metrics based on the predictability percentage of each metric.

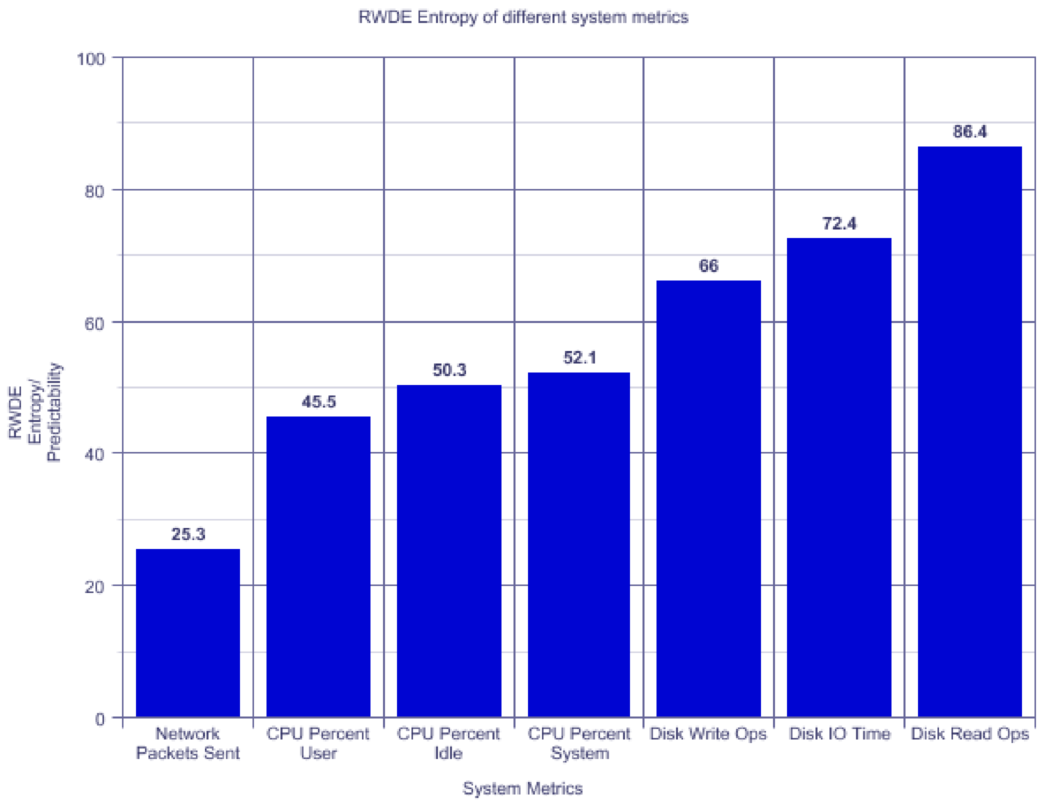


Figure 6.3: Ranking of system metrics based on predictability percentage calculated using RWDE.

In order to test the intuition that RWDE can rank the metrics in the correct order or RWDE can quantify the complexity of time series, we compare RWDE entropy values with the forecasting error of a simple model. In order to study this relationship, we use Autoregressive Integrated Moving-Average Model (ARIMA) to generate a model-based prediction on system metrics considered. We ARIMA because it is very popular and among the most widely used forecasting models.

Experimental Setup. In particular, we use the "auto.arima" [33] model which selects model parameters automatically using well-known methods like the Akaike information criteria and Canova-Hansen test. We consider all the seven system metrics described in Table 4.1 from a random virtual machine. For each metric, we calculate the error of the prediction with respect to the true Time-series continuation of the metric. Specifically, we split each metric time series in the "train" and "test" set: the first 85% (approx. 25 days) is the training set and the last 15% (approx 5 days) as the test set. The training set is used to build the ARIMA model and later the model is used to generate an n-step prediction of the next n-values (15%). The predictions are then compared to the test set to calculate the prediction error.

We use Mean Absolute Scaled Error (MASE) [35], also known as MASE score, as a

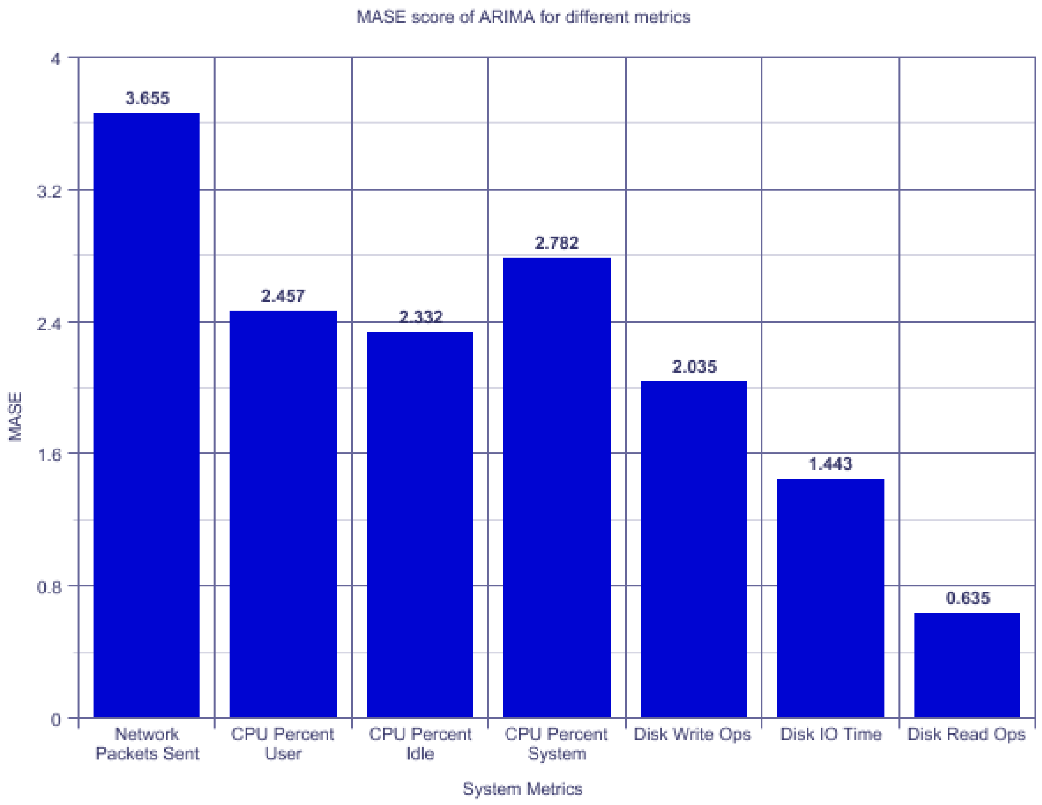


Figure 6.4: MASE score of ARIMA model applied on system metrics.

numerical measure of prediction accuracy. The MASE score is a normalized measure of prediction error. To obtain the MASE score of ARIMA for each metric, we divide the Mean Absolute Error (MAE) obtained from the ARIMA model by MAE obtained from Random-Walk prediction over the training set. A Random-walk model simply uses the previous value observed in time series as a prediction value for the next step. The MASE is calculated as follows :

$$Mean\ Absolute\ Error\ (MAE) = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

$$Mean\ Absolute\ Scaled\ Error\ (MASE) = \frac{MAE_{ARIMA}}{MAE_{Random\_walk}}$$

Therefore, MAE is arithmetic average of absolute errors $|e_i| = |y_i - x_i|$. Where x_i is the true value and y_i is the prediction. A MASE score of less than 1 would imply that that the average prediction error of ARIMA for a given metric, was lower than the average prediction error of the Random-walk model. On the other hand, a MASE score greater than 1 would mean the ARIMA model for a given metric performed worse than the Random-walk model.

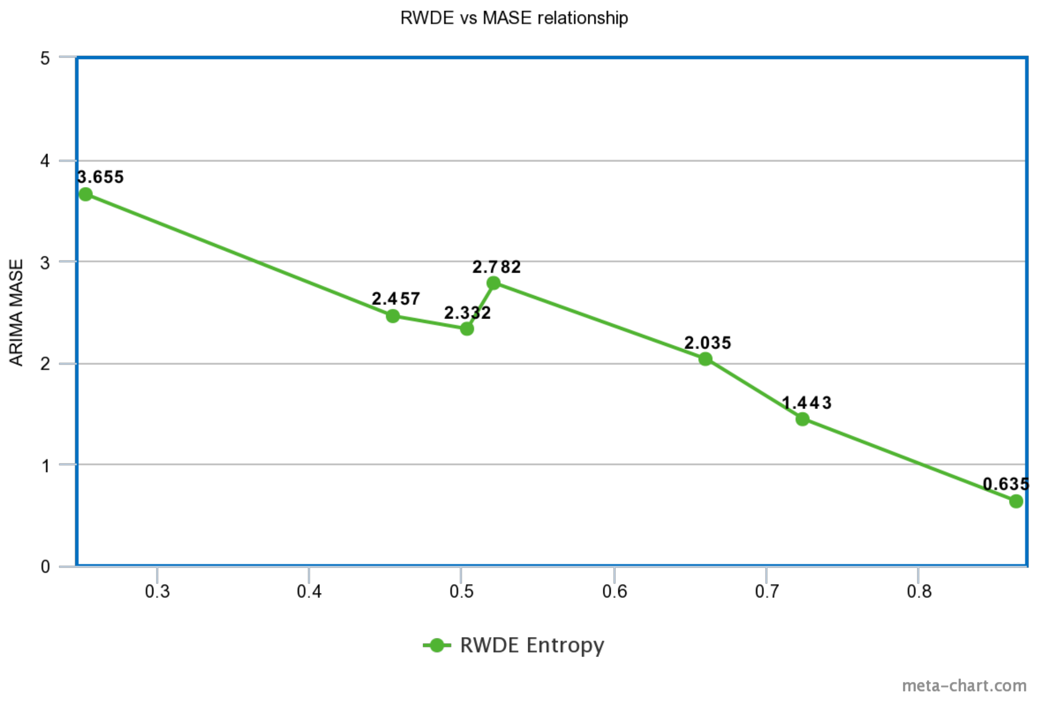


Figure 6.5: Relationship between MASE (model based prediction error) and RWDE

Results. Figure 6.4, shows the MASE scores of the ARIMA model applied to the metrics we considered in ranking the metrics (Figure 6.3). The metrics are arranged in the same order as in Figure 6.3. We can observe that the ARIMA model performs worst on the CPU Network Packets Sent (highest MASE score), whereas it performs best on the Disk Read Operations metric. Therefore, as far as model-based predictability is concerned, according to the ARIMA model, the CPU Percent-System metric is the most complex, and the Disk Read Operations metric is the least complex. This corroborates with the complexity ranking of metrics using RWDE in Figure 6.3 .

Finally, Figure 6.5 shows the relationship between RWDE predictability percentage and MASE. The line graph is directly derived from Figure 6.3 and Figure 6.4. We can observe that there is a linear relation between RWDE and model-based prediction error. Most importantly, the relationship implies that a system metric with a low RWDE predictability percentage is likely to yield a high prediction error when a prediction is applied. Overall, RWDE can help in selecting system metrics with a high predictive structure. This can play important role in selecting the right set of metrics for anomaly detection and clustering application.

However, we need to keep in mind that this relationship is subject to the ”No Free Lunch” theorem [73][74]. Therefore, obviously, ARIMA is not the most suitable model for prediction tasks for all the considered system metrics. Some other forecasting model might showcase better or worse relationship with RWDE. We can observe that the MASE score for CPU Percent-System is higher than expected, which might indicate the unsuitability of ARIMA for the metric. Albeit, the auto-correlation plot (Figure 6.6) for the disk write operation shows a high correlation between initial points which ARIMA would likely capture. Although, we also observe correlation at 1,440 data point which would be missed by ARIMA. However, correlation at 1,440 data point indicates seasonality on day basis. We believe this makes the ARIMA model a good match for the prediction task, if not the best.

RWDE can be a handy and easy method to rank cloud metrics according to their predictability.We believe, RWDE can be used as a feature selection tool for time series forecasting applications.

6.3 Relationship Between Model-Based Predictability and RWDE

In Chapter 3, we defined RWDE as a method to quantify the *intrinsic predictability* of a time series. Most importantly, RWDE is an entropy method and not a forecasting method. In most of the time series forecasting applications, the performance is measured using Forecasting Error (FE) or Mean Absolute Error (MAE), which is completely dependent on the model used. However, we observed some kind of relationship between RWDE and model-based predictability in Section 6.2. In this section, we try to explore this relationship in depth to answer our third research question **(RQ3)**, i.e., What is the relationship between intrinsic predictability and realized predictability? In particular, we try to establish an em-

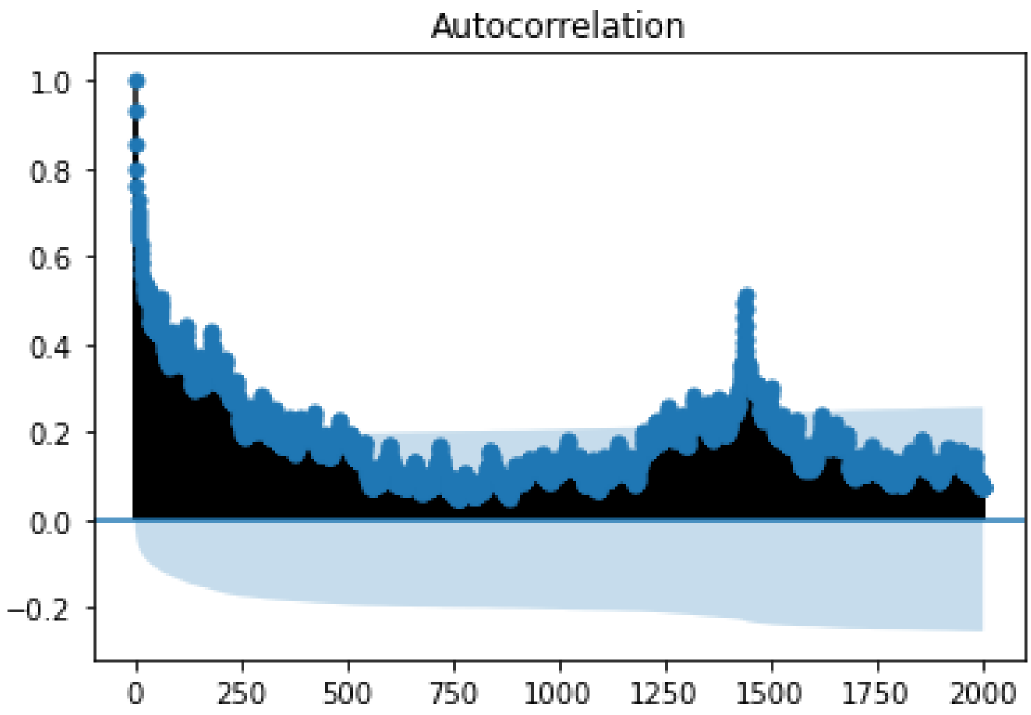


Figure 6.6: Auto-correlation plot of disk write operations metric.

pirical heuristic for model selection or suitability based on the relationship between model based predictability and RWDE. Ideally, there should be some sort of monotonic relationship between entropy method and model-based predictability.

> **RQ3. What is the relationship between intrinsic predictability and realized predictability?**

Experimental Setup. For the experiments to establish an empirical heuristic, we again consider the seven system metrics we considered in Section 6.2. We consider these metrics from four different random virtual machines, taking the total number of metrics to 28. The objective of these experiments is to explore the relationship between model-based predictability and RWDE. For prediction strategies, we consider three types of models:

- Naive Prediction Strategy: A simple prediction strategy that calculates the mean of all previous observations to generate the next step forecast.
- Regression Based Prediction Strategy: ARIMA (auto.arima) [33].
- Non-linear Prediction Strategy: Facebook Prophet [68].

Our experimental setup is very similar to Garland et al. [26]. In [26], the authors used WPE with Naive, ARIMA and Lorenz Method of Analogues (LMA) for establishing a heuristic. In this experiment, we try to replicate it with RWDE.

To keep the experiments independent of parameter selection, we use auto.arima and Facebook Prophet. For both the methods, we don't need to set any parameters. The parameters are selected by the methods themselves using sophisticated criteria. Also, for the Naive prediction strategy, we do not need to set any parameter.

Similar to Section 6.2, we consider metrics recorded for 30 days. We split each metric time series in the training set (85%, approx. 25 days) and test set 15% (approx 5 days). We calculate RWDE entropy values for all system metrics and MASE scores for all system metrics using the three mentioned prediction strategies. In case of RWDE, we expect low MASE score for high RWDE and vice versa. We use the Spearman's Rank Correlation Coefficient [42] test to assess if there is a monotonic relationship between the RWDE and model-based predictability. We further repeat the same experiments with other entropy methods to verify if any other entropy method forms significant relationship with model-based predictability.

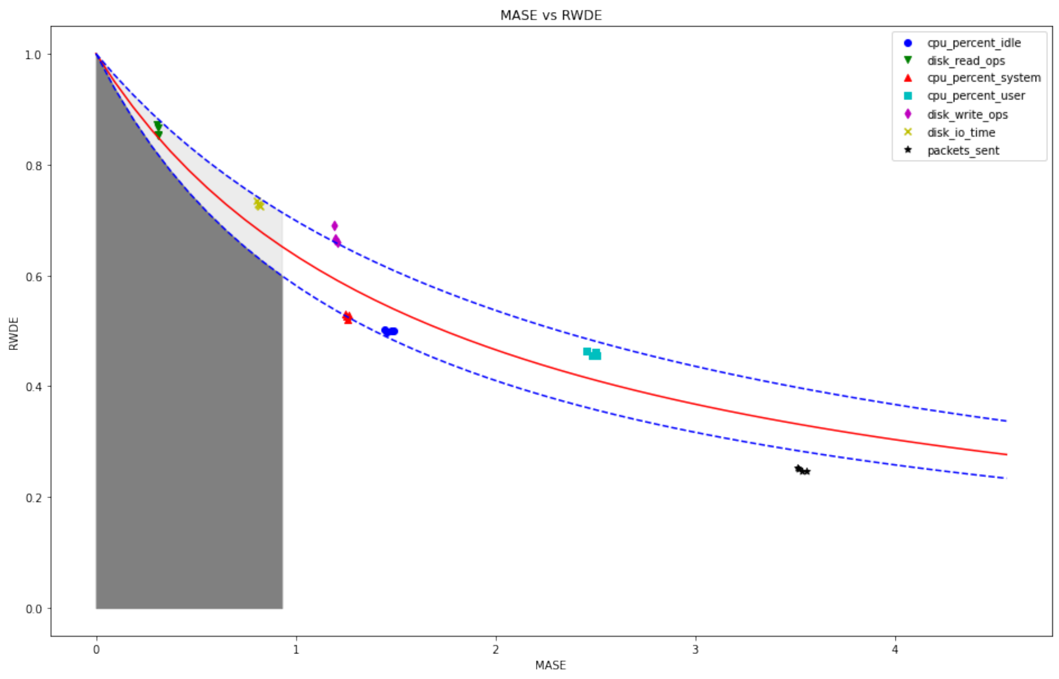


Figure 6.7: RWDE vs MASE score of best fit model for each system metric.

Results. Figure 6.7, shows the plot for RWDE entropy values versus best MASE score (i.e., MASE score of the best fit model out of the three) for each system metric. We can clearly observe an inverse trend between RWDE and MASE. Further, we try to fit a curve to this relationship. Obviously, linear regression or fit won't be suitable. Because, for RWDE = 1, MASE should be ideally equal to zero. Therefore, a suitable curve should pass from the point (0,1). Moreover, as RWDE tends towards zero, MASE tends to large values. So, we expect RWDE and MASE to reach an asymptote. After considering these points, we fit an inverse logarithmic function of form $y = 1/(a\log(bx+1)+1)$ (coefficient values a = 7187.86621, b = 0.00008). Where $y =$RWDE and $x =$MASE. In Figure 6.7, the solid red curve shows the fit and the dashed blue curves are $\pm$ one standard deviation of the fit. The system metrics lying in the light grey area between blue curves and MASE $<$ 1 are the best predictable metrics considered in this study. Moreover, the dark grey represents the ideal region for a system metric. A metric lying in the dark grey area would indicate a high predictive structure. The shaded regions are limited to MASE$<$ 1, because MASE$>$1 would mean the random walk is a better alternative than the model in consideration.

We believe Figure 6.7 can be used as a heuristic for determining a suitable prediction strategy or model (linear or non-linear) for a time series. By heuristic, we mean the relationship graph between RWDE and MASE, which can be used as by a practitioner to select a suitable forecasting method. Obviously, with only seven system metrics we do not exhaustively cover all the spaces of the RWDE-MASE relationship. Again, the goal of these experiments is not determining the best fit model for a time series. Rather the goal is to explore the RWDE-MASE relationship to gain insights in determining if a linear prediction strategy or a non-linear prediction strategy would be suitable for a time series.

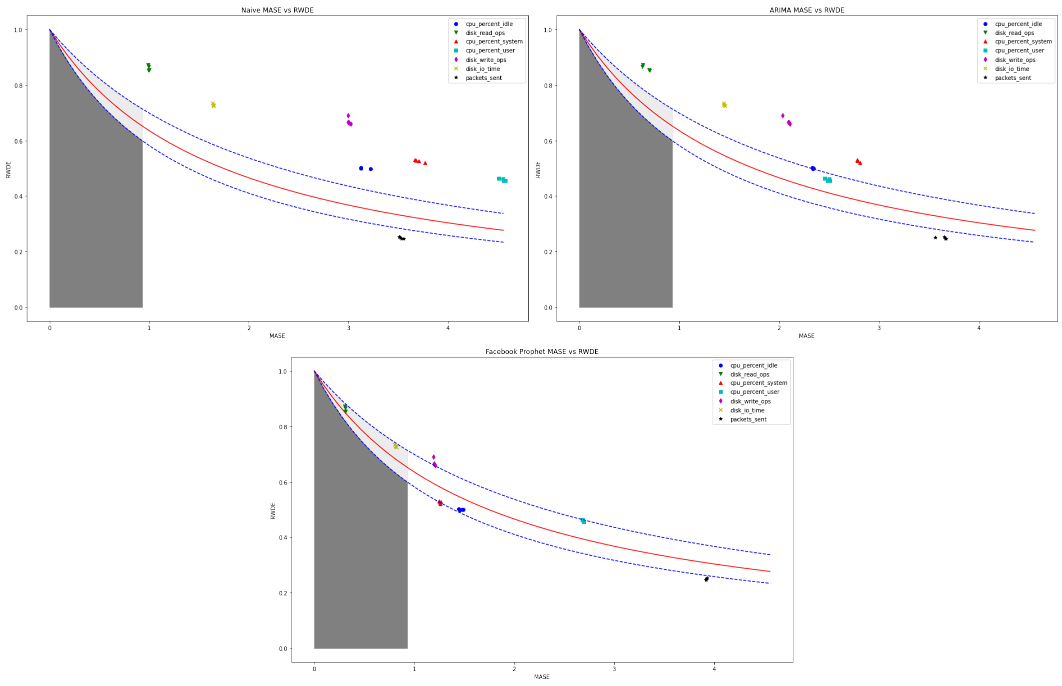


Figure 6.8: RWDE vs MASE for all the prediction strategies.
The curves in each plot are same as in Figure 6.7

The Figure 6.8, shows the RWDE vs MASE plots for all the prediction strategies for each system metric. Obviously, the RWDE entropy values do not vary across different experiments. Moreover, the variance between the clusters of same system metrics is low for all of them. Most importantly, we can observe that most the system metrics fall outside the grey area. Which insinuates complex nature of these system metrics. Only disk read operations and disk IO time metrics fall in the grey area. For all other system metrics, the heuristic suggests that the prediction strategies are not suitable. Rather they are unnecessarily complex in that a simple random-walk method outperforms these sophisticated methods. Essentially, the heuristic can easily determine if a method is unable to exploit the structure present in the time series, if RWDE vs MASE value of the time series lies outside the grey area. On the other hand, if the RWDE vs MASE value for the time series lies in the light or dark grey area would suggest that the prediction method is suitable.

Overall, Facebook Prophet performs better than the other two prediction strategies. For the naive method, none of the predictions are not helpful (Random-walk is better). For auto.arima, we can observe that disk read operation metric has a very low MASE score. However, it does not fall into the grey area as the heuristic is aware that Facebook Prophet performs better on the metric. Therefore, the heuristic can identify mismatch of a model for a time series. However, Facebook prophet does not always perform better than the other two methods. auto.arima and naive perform better than Facebook prophet for cpu percent user and network packets sent respectively.

The heuristic we built is just representative of the methods we considered. Obviously, a more sophisticated heuristic can be built similarly with possibly better prediction methods. Moreover, the system metrics we used in the experiments do not cover the RWDE vs MASE space exhaustively, which can result in sub-optimal results. However, with the seven chosen metrics, we try to make the heuristic as representative as possible.

Table 6.3: Spearman's Correlation coefficients between best fit MASE score and entropy values of seven entropy methods.

Entropy Method	Spearman's correlation	P-value
SE	0.6121	0.0673
PE	0.2148	0.2722
WPE	0.5167	0.1006
RPE	0.2550	0.1908
DE	0.2485	0.2022
RDE	0.1283	0.5149
RWDE	0.8801	6.744e-10

In Figure 6.5, we can clearly observe a negative monotonic relationship between RWDE and MASE scores. Table 6.3 shows the Spearman's Correlation Coefficient between the seven entropy values and MASE scores, and the corresponding P-values. Spearman's Correlation Coefficient determines if there is a monotonic relationship between two variable. The monotonic relationship does not necessarily need to be linear (In case of RWDE, it is inverse logarithmic). Two variables form perfect monotone function, if the Spearman's Correlation Coefficient is +1 or -1. In Table 6.3, we can observe that RWDE and MASE have almost a perfect Spearman's Correlation Coefficient (0.8801) with 99.9% confidence that correlation has not occurred by chance (P-value=6.744e-10). For, all the other entropy methods, none of them have a significant Spearman's Correlation Coefficient (P-value$\leq$0.01). Therefore, unlike RWDE, it is difficult to represent the relationship between other entropy methods and MASE scores with a monotonic function.

Chapter 7

Discussion

In this chapter, we discuss our main findings and compare them with related work in Section 7.1. In Section 7.2, we consider the limitations that RWDE possesses. We also discuss the threats to validity of this study in Section 7.3. Finally, in Section 7.4, we discuss the promising directions and future work.

7.1 Main findings

We first revisit the main findings from Chapter 5 and discuss these main findings with respect to the research question we posed in Section .

RQ1. What is the difference between the performance of RWDE and other entropy methods?

In our experiments with the synthetic signal, we evaluated the entropy methods on three factors : sensitivity, robustness, and stability. In most of the related work presented before, these three factors have been considered to compare entropy methods [48] [62] [17]. Li et al. [48] also considered *distinguishability* as a factor to compare entropy methods. Distinguishability is considered the ability of the entropy method to distinguish between different types of time series data. However, Bandt [8] and Rostaghi et al. [62] argued that entropy methods are methods to quantify the intrinsic complexity of time series rather than classification methods. Moreover, distinguishability does not relate to the accuracy of complexity quantification entropy methods because two different types of signals could have the same intrinsic complexity. Hence, in this study, we do not consider distinguishability as a factor for the comparison of entropy methods.

Sensitivity. In simulation 1 (Section 5.2) , we considered a mutation signal to evaluate the ability to detect information (spikes) in data. We statistically found that RWDE was sensitive towards sudden spikes in time series data, along with DE and RDE. We observed RWDE is sensitive to information presence and change and does not miss any information for entropy calculation. We believe, sensitiveness of RWDE makes it a good candidate for anomaly detection tasks/applications.

Robustness. In simulation 3 (Section 5.4), we considered time-series of varied signal to noise ratio (SNR) or information to noise ratio to evaluate the reliability of entropy method. We found that RWDE is significantly robust to noise compared to other entropy methods. However, the RDE performed better than RWDE under noisy conditions. We suspect that RWDE gave lower weights to new patterns because the change in information was gradual rather than sudden.

Stability. Finally, in simulation 4 (Section 5.5), we considered a deterministic and non-linear cosine signal to evaluate the stability of the RWDE method. We found RWDE to be the fastest to converge to an entropy value compared to other entropy methods . Most importantly, RWDE had the lowest standard deviation while converging to an entropy value. The results of the stability test indicate that RWDE has better stability than the other five entropies.

The main difference between most of the other methods and RWDE was that RWDE considered Weighing (”Weighted”) and distance to white noise (”Reverse”) in entropy calculation. The difference in performance in comparison to other methods can be mainly attributed to these two entities in RWDE calculation. We also consider the following two sub-research questions to conclude the differences between the performance of RWDE and other entropy methods :

RQ1.1 what is the effect of the weighted entropy method?
In simulations 1, 2, and 3 we clearly observed that only weighted and dispersion entropy methods were able to detect spikes in data and change in information density. Especially, in simulation 2 (Section 5.3) WPE was able to detect spikes in data with the right parameters whereas PE was not able to detect the spike. This primarily signifies the role of weights in entropy calculation. Most importantly, the intent behind weighing patterns is to give more importance to changing trends in data and overpower the noise in data. This clearly visible in all the simulations, as RWDE is sensitive to information change in data. We believe the amplitude of the switches plays a key role in the predictability of time series, so weighting the scale of ordinal changes in time series is important for quantifying the predictive structure accurately. Therefore, we infer that weighing ordinal patterns contributes to the sensitivity of the entropy method.

RQ1.2 what is the effect of adding distance to noise
In our experiments, we considered RPE, RDE, and RWDE which include distance to white noise in the calculation of entropy value. The reverse entropy methods simply calculate the squared euclidean distance of observed pattern distribution from uniform distribution $(1/m!)$ in the space of all pattern distributions. Therefore, reverse entropy methods are much simpler to understand and faster compared to logarithmic (Shanon) based entropy methods. Bandt [8], in his empirical study, showed that reverse entropy methods have better statistical properties than logarithmic entropy methods. In studies [46][47][48], it

has been shown that reverse entropy methods have more stable performance compared to logarithmic entropy methods. Our results from stability test in simulation 4 (Section 5.5) corroborates with results in these previous studies. On average reverse entropy was more stable than logarithmic entropy methods. Therefore, we believe adding distance to noise in RWDE calculation enhances its stability.

RQ2. What is the effect of parameters on RWDE?

For this research question, we consider only two parameters related to RWDE: embedding dimension m and the number of classes c. We do not consider window length and delay parameters related to entropy methods because those parameters largely depend on time series data and the application of the entropy method rather than the performance of the entropy method itself [61]. Even though we consider both embedding dimension and number classes, we actually are considering only embedding dimension. Because setting up the number of classes is completely dependent on embedding dimension with the rule : length of time series $(T) > c^m$. Therefore, once we fix the embedding dimension parameter, the number of classes is set automatically. Therefore, in this section, we mainly focus on the sensitivity of RWDE with respect to the embedding dimension.

Determining the exact range of embedding dimension m has been treated as a theoretical problem in studies [39][7]. However, most of the entropy studies recommend setting embedding dimension between 3 and 8 [62][48]. Ideally, we would want embedding dimension to have minimal effect on entropy value. Obviously, we expect embedding dimension to affect entropy value calculation of RWDE. Rather, we are interested in determining the extent of effect relatively, i.e., we can determine the entropy method with the least sensitivity to the embedding dimension.

RQ3. What is the relationship between intrinsic predictability and realized predictability?

In Section 6.3, we considered naive, linear (regression-based), and non-linear prediction strategies to represent realized predictability of different system metrics. The RWDE vs MASE plot (6.7) clearly showed an inverse (inverse-logarithmic/exponential) relationship between RWDE and MASE score. This implies a direct positive relationship between intrinsic predictability and realized predictability. Most importantly, this relationship can be used as a heuristic to suggest the suitability of a prediction strategy. The heuristic can identify if a linear or non-linear prediction strategy is a mismatch for a particular time series.

For representing a heuristic using the RWDE vs MASE relationship, we used a limited number of forecasting strategies and system metrics. The results we draw from the heuristic are positive but not generalizable to large extent. However, our findings are consistent with prior studies in [26][31]. In fact, results of this study are a step ahead of results in these studies. Both [26][31] proposed a similar heuristic based on Weighted Permutation Entropy (WPE) but with synthetically augmented data. In this study, we used complex real-world data from a dynamic ING private cloud (IPC) system. There is a clear difference between the data used in [26][31] and this data. The system metrics in this study are way more complex than the synthetic data considered in those studies. The complex time series data compelled the heuristic in suggesting Random-walk method as the most suitable prediction strategy.

For the cloud metrics considered in this study, we established a significant relationship between RWDE and model-based predictability. However, we failed to establish any kind of meaningful relationship between other entropy methods and model-based predictability. This significantly shows the superiority of RWDE over other entropy methods.

In Section 6.1, we compared the entropy values of the six entropy methods for different values of embedding dimension for a system metric. We observed that RWDE is the least sensitive to change in embedding dimension. Other reverse entropy methods perform worse due to nonexistence weights in entropy calculation, as RWDE normalizes the amplitude changes in the time series. This corroborates with results insinuated in studies [22][26] with respect to weighted entropy methods. We believe this result plays important role in the usability of RWDE. Most importantly, lower sensitivity towards embedding dimension makes RWDE suitable for real-world applications.

7.1.1 Which cloud metrics are most informative?

In this study, we considered system metrics from broadly three categories : CPU , Disk, and Network (Table 4.1). The experiments in Section 6.1 clearly showed that system metrics related to the network of the system were most complex with low RWDE and high MASE score. Metrics related to CPU also showed the same characteristics. These system metrics were so complex that sophisticated forecasting methods like auto.arima and Facebook Prophet couldn't capture the underlying structure. Moreover, the simple Random-walk method performed better on these methods. However, most of the disk-related system metrics had some predictive structure present in their time series. Especially disk read operations, disk IO time and disk write operations metric. These metrics often had high RWDE entropy value and low MASE scores. Hence, metrics related to disk operations were most informative. As an organization, ING can focus storing more predictable metrics like disk operation metric for a longer time, compared to less predictable metrics for getting better forecast results.

7.2 Limitations of RWDE

7.2.1 Choice of Parameters for RWDE

Despite being less sensitive to the embedding dimension, one of the major limitations of RWDE is that it is dependent on a set of parameters. One can argue that selecting appropriate parameters for entropy methods is more tedious than selecting minimal parameters for model-based methods. Finding the right set of parameters for RWDE could be a strenuous task. We consider the four parameters related to RWDE:

- Window length: Most of the Model-based complexity measures are independent of window length because one can feed all the data at once. However, in a real-time application, one might need to consider windows of data which makes RWDE suitable for observing changing complexity. In this study, we considered window length equal to one day for system metrics, because recorded metrics of a day are mostly similar to the next day. Choice of window length is mostly dependent on the dynamic system and its seasonality rather than the entropy method. However, the choice of the window plays an important in selecting the values of embedding dimension and number of classes.

- Delay: In [61], Riedl et al. suggested choosing delay τ as the period length of the main oscillation of the dynamic system in consideration. In this study, every system metric considered was recorded (some aggregated) at one-minute granularity. In order to evade any further data aggregation or loss, we kept delay always equal to 1. However, depending on the dynamical system and application, the delay must be set with some domain knowledge.

- Embedding Dimension (m): In all the entropy methods including RWDE, selecting an appropriate value for the embedding dimension is the most difficult task. Zambrano et al. [2] in their study recommended selecting an embedding dimension (m) satisfying length of time series $N > 5m!$. Which typically results in $3 \leq m \leq 8)$ for most of the applications. They also recommended selecting a value of m, for which the average normalized entropy is highest.

- Number of classes (c): The value of c is mostly dependent on m, given the recommended setting $T > c^m$. That said, the value of c should be larger than one because at $c = 1$ there is only one dispersion pattern. Most importantly, if we set c too small, data points with large amplitude difference would get assigned to the same class. Whereas a large value of c might result in data points with small amplitude difference getting assigned to different classes, resulting in high sensitivity to noise for RWDE.

7.2.2 RWDE & Linear Time Series.

As much as RWDE is effective on non-linear time-series for quantifying the complexity of time series, it fails to capture linear structure in a time series. Entropy calculation in RWDE is primarily based on ordinal patterns present in a time series. Given that there is only a trend in linear time series and no repetition of patterns, RWDE fails to understand the structure in time series. However, in Section 6.1 we observed that RWDE is able to understand the trend if there is a pattern present. Fortunately, most of the real-world dynamic systems generate complex non-linear time series data.

In Figure 7.1, the figure on the left shows a modified simulation 1 (Section 5.2) with a simple, almost deterministic (due to small Gaussian noise) linear trend at the end of the time series. The figure on the right (7.1), shows the RWDE calculation for the before mentioned time series. We can observe that RWDE detects the linear structure with a slight jump in entropy value. However, one would expect the RWDE entropy value to move close to one because of the almost deterministic nature of the linear structure.

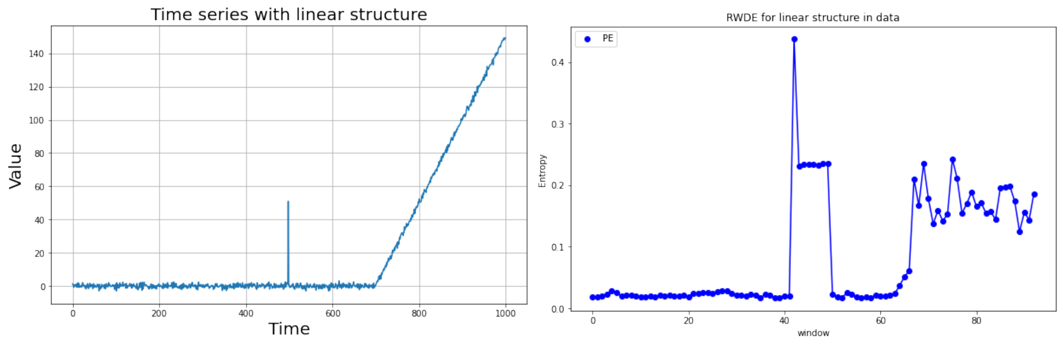


Figure 7.1: On the left, a simulated time series similar to simulation with linear structure at the end. On the right, RWDE values for the time series on left.

7.3 Threats to Validity

In this study, while carrying out experiments we observed some threats to validity of this study. In this section, we discuss these observed threats to validity.

Internal Validity. Threats to internal validity focus on forces or factors that are not part of independent variable, affecting the results drawn in this study. One of the factor that could have affected the empirical analysis carried out his study is experimenter bias. In this study, experimenter bias can inadvertently affect the results due individual behavior towards observed data. Moreover, author was part of the organization (ING) whose data is considered in this study. To counter this bias, the data analysis and interpretation of the results were supervised by two other colleagues in qualitative research.

In this study, we used the Mean Absolute Scaled Error (MASE) score as forecasting error to represent realized predictability of a time series. The MASE score represents the Mean Absolute Error (MAE) of a model relative to the MAE of random-walk algorithm. However, the MASE score might not be the perfect fit for every time series. A random-walk algorithm on highly oscillating time series would always result in wrong prediction and hence a highly skewed MASE score. Secondly, we divided the time series in training and test data sets. A signal could be quite different in these two data sets due to non-stationarity (regime shift), which can result in skewed MASE score. System metrics considered in this study exhibit nonstationarity to some extent. To counter this bias we eluded oscillating linear time series and highly nonstationary time series data from the experiments.

External validity. External threats to validity are concerned with factors that are not part of the experiment. In this study, it is mainly concerned with generalization of the results. In this study, we focus on only simulated and system metric data from only one organization. This makes it important to focus on limitation of generalizability of results. We tried to make the data used in the study as comprehensive and representative as possible. For representing a heuristic using the RWDE vs MASE relationship we used limited number of forecasting strategies and system metrics. The results we draw from the heuristic are optimistic but not generalizable to a large extent. More exploration using sophisticated methods using domain knowledge and exhaustive data from different organizations would be needed to establish such claim. The conclusions of this study cannot be generalized to any other organization.

7.4 Future Work

Based on our study in this thesis work, we identified some directions for future research work. In Chapter 3, we introduced RWDE as a generalized entropy method for quantification of time series complexity. This makes RWDE a contender for all the existing applications using entropy methods. In this section, we present research directions related to applicability of RWDE in IT monitoring and model quantification. We believe, progress in these directions can be crucial for full utilization of RWDE in large IT-driven organizations.

RWDE for anomaly detection and clustering. Our results in Section 5.2 show that RWDE can detect spikes in time-series data. RWDE's ability to detect spikes and change in information makes it suitable feature for anomaly detection application in IT-monitoring. A handful of studies have used entropy for anomaly detection and attacks on cloud systems [71][52]. However, these studies used Shannon's entropy rather than sophisticated entropy methods such as Permutation Entropy or RWDE. For anomaly detection in cloud systems, multiple correlated system metrics need to be considered along with some feature engineering. How can RWDE used for correlated multi-variate time series data? How can it be used for anomaly detection in a multi-variate data scenario? We believe the answer to these questions can help us use RWDE to its full potential and to avail high business impact. Moreover, RWDE can be used for clustering virtual machines for managing resources [15]. The overall entropy of a virtual machine is good representation of the load and condition of VM. VM's can be clustered based on entropy method for resizing VM's and managing resources to scale cloud computing. In general, with this study, we like to argue that entropy methods are a good way to understand underlying complexity of cloud systems and entropy methods can play important role in ITOps or AIOps in the future.

Fine grained analysis of RWDE parameters. In this study we acknowledge that RWDE is dependent on different parameters. These parameters might often depend on the dynamic system in consideration. An interesting opportunity for future work is a fine grained analysis of RWDE parameters on data from different real world dynamic system. This would allow a deeper analysis into affect of parameters on RWDE. Such a study could be similar to [61], which did a fine grained analysis of Permutation Entropy parameters on different datasets.

Model specification with regime shift. Our results in Section 6.3 show that the relationship between RWDE and realized predictability can be used as a heuristic for specifying model suitability for time series forecasting. However, in this study, we did not take the non-stationarity of dynamic systems into consideration. Dynamic systems change their state frequently, which could lead to a regime shift in corresponding time-series data. Non-stationarity can result in a change of predictive structure of a time series with time. We can imagine with a different regime in the training and test data set, the realized predictability (MASE) would be skewed . How does RWDE evolve with regime shift in data? How do the regime shifts in data affect the intrinsic vs realized predictability relationship? The answer to this question could help us build an adaptive model specification heuristic. Entropy methods have been used before to detect regime shifts [8][13]. Detecting regime could help in determining model suitability dynamically as the regime shifts, discarding unsuitable models and applying new suitable models over time.

Chapter 8

Conclusion

In this chapter, we revisit the main contribution of this thesis we posed in the Chapter 1. We conclude by discussing the main findings of the thesis. After this overview, we will reflect on the results and draw some conclusions.

8.1 Conclusion

The main purpose of this thesis has been to introduce a new entropy method Reverse Weighted Dispersion Entropy (RWDE) and how the relationship between RWDE and model-based predictability can be used as a heuristic for determining model suitability for time series forecasting. This thesis presented the first attempt in the literature to quantify the complexity of cloud metrics. RWDE is a contribution to the open-source community and available as a Python package.

We carried out simulation experiments to characterize RWDE and evaluate its effectiveness. In each simulation, we targeted a specific property of entropy methods such as sensitivity, stability, and robustness. In comparison to other entropy methods considered, RWDE performed better in detecting mutation signals or spikes in the signal. RWDE also showcased better stability compared to other entropy methods. Lastly, RWDE was also more robust to noise compared to PE, WPE, RPE, and DE. However, RDE was slightly more robust than RWDE. RWDE is derived from Permutation Entropy (PE) by adding a couple of features: weighted entropy and distance to noise. We observed that weighing ordinal patterns in entropy calculation contributes to the sensitivity of the entropy method. Moreover, adding distance to noise in the RWDE calculation enhances its stability. It is also worth mentioning that RWDE is least sensitive to parameter initialization. We observed that RWDE has the least variation to different initial values of embedding dimension, which gives it an edge over other entropy methods. Overall, RWDE can be deemed as an effective approach for quantifying complexity.

We further used RWDE to quantify the complexity of real-world cloud metrics. RWDE can be a handy and easy method to rank cloud metrics according to their predictability.

We believe RWDE can be used as a feature selection tool for time series forecasting applications. Moreover, an organization can store more predictable metrics for a longer time compared to less predictable metrics. In our particular case, we found that disk metrics have the most predictive structure, followed by CPU and network metrics.

Most of the forecasting models fail to capture the underlying predictive structure of a time series if its complexity predominates the information or pattern redundancy. Without knowing the underlying generating process, it is difficult to estimate the inherent predictability of time series. Without an estimation of inherent predictability, it is difficult to determine if a forecasting model can capture the underlying predictive structure. Also, a forecasting model might simply not be a good fit for a time series. It would be a lot easier for practitioners if they can determine wether failure of the forecasting model is due to the absence of predictive structure, or unsuitability of the model for a particular time series.

In this thesis, we argued that the relationship between intrinsic predictability (quantified by RWDE) and realized predictability (quantified by MASE) can be a useful heuristic for determining the suitability of a forecasting method. We observed that there is in fact a relationship between these two. In general, the higher the RWDE (more predictive), the lower the forecasting error. The relationship is almost inverse logarithmic or exponential. Most importantly, if the forecasting error of the model is high or poor but the RWDE of the time series is high (high intrinsic predictability). A practitioner can conclude that the forecasting model is not a good match for the given time series. In general, the heuristic indicates that for very complex time series, simple prediction strategies like random-walk perform better than more sophisticated strategies.

Bibliography

[1] Khalid Alhamazani, R. Ranjan, Karan Mitra, Fethi Rabhi, Prem Prakash Jayaraman, Samee Khan, Adnene Guabtni, and Vasudha Bhatnagar. An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art. *Computing*, 97, 04 2014. doi: 10.1007/s00607-014-0398-5.

[2] José Amigó, Samuel Zambrano, and Miguel Sanjuán. Permutation complexity of spatiotemporal dynamics. *http://dx.doi.org/10.1209/0295-5075/90/10007*, 90, 04 2010. doi: 10.1209/0295-5075/90/10007.

[3] José Amigó, Karsten Keller, and Juergen Kurths. *Recent Progress in Symbolic Dynamics and Permutation Complexity Ten Years of Permutation Entropy*, volume 222. 06 2013. doi: 10.1140/epjst/e2013-01839-6.

[4] Shiryayev A.N. New metric invariant of transitive dynamical systems and automorphisms of lebesgue spaces. In Shiryayev A.N, editor, *Selected Works of A*, volume 27 of *N. Kolmogorov. Mathematics and Its Applications (Soviet Series*. Springer, Dordrecht. doi: 10.1007/978-94-017-2973-4_5. URL `https://doi.org/10.1007/978-94-017-2973-4_5`.

[5] M. Anand. Cloud monitor: Monitoring applications in cloud. In *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 1–4, 2012. doi: 10.1109/CCEM.2012.6354603.

[6] Alexandra Antoniouk, Karsten Keller, and Sergiy Maksymenko. Kolmogorov-sinai entropy via separation properties of order-generated -algebras. *Discrete and Continuous Dynamical Systems*, 34:1793–1809, 05 2014. doi: 10.3934/dcds.2014.34.1793.

[7] C. Bandt and B. Pompe. Permutation entropy: a natural complexity measure for time series. *Physical review letters*, 88 17:174102, 2002.

[8] Christoph Bandt. A new kind of permutation entropy used to classify sleep stages from invisible eeg microstructure. *Entropy*, 19:197, 04 2017. doi: 10.3390/e19050197.

[9] M.S. Baptista, E.J. Ngamga, Paulo R.F. Pinto, Margarida Brito, and J. Kurths. Kolmogorov–sinai entropy from recurrence times. *Physics Letters A*, 374(9):1135 – 1140, 2010. ISSN 0375-9601. doi: https://doi.org/10.1016/j.physleta.2009.12.057. URL `http://www.sciencedirect.com/science/article/pii/S0375960109016260`.

[10] Brian Beckage, Louis J. Gross, and Stuart Kauffman. The limits to prediction in ecological systems. *Ecosphere*, 2(11):art125, 2011. doi: https://doi.org/10.1890/ES11-00211.1. URL `https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1890/ES11-00211.1`.

[11] Jasmin Bogatinovski, Sasho Nedelkoski, Alexander Acker, Florian Schmidt, Thorsten Wittkopp, Soeren Becker, Jorge Cardoso, and Odej Kao. Artificial intelligence for it operations (aiops) workshop white paper, 2021. URL `https://arxiv.org/abs/2101.06054`.

[12] LUDWIG BOLTZMANN. *Further Studies on the Thermal Equilibrium of Gas Molecules*, pages 262–349. 1872. doi: 10.1142/9781848161337_0015. URL `https://www.worldscientific.com/doi/abs/10.1142/9781848161337_0015`.

[13] Yinhe Cao, Wen-wen Tung, J. B. Gao, V. A. Protopopescu, and L. M. Hively. Detecting dynamical changes in time series using the permutation entropy. *Phys. Rev. E*, 70:046217, Oct 2004. doi: 10.1103/PhysRevE.70.046217. URL `https://link.aps.org/doi/10.1103/PhysRevE.70.046217`.

[14] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, 2009.

[15] V. Chavan and P. R. Kaveri. Clustered virtual machines for higher availability of resources with improved scalability in cloud computing. In *2014 First International Conference on Networks Soft Computing (ICNSC2014)*, pages 221–225, 2014. doi: 10.1109/CNSC.2014.6906707.

[16] Zhe Chen, Li Yaan, Hongtao Liang, and Jing Yu. Improved permutation entropy for measuring complexity of time series under noisy condition. *Complexity*, 2019:1–12, 03 2019. doi: 10.1155/2019/1403829.

[17] Zhe Chen, Li Yaan, Hongtao Liang, and Jing Yu. Improved permutation entropy for measuring complexity of time series under noisy condition. *Complexity*, 2019:1–12, 03 2019. doi: 10.1155/2019/1403829.

[18] Yingnong Dang, Qingwei Lin, and Peng Huang. Aiops: Real-world challenges and research innovations. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*, ICSE '19, page 4–5. IEEE Press, 2019. doi: 10.1109/ICSE-Companion.2019.00023. URL `https://doi.org/10.1109/ICSE-Companion.2019.00023`.

[19] S. A. De Chaves, R. B. Uriarte, and C. B. Westphall. Toward an architecture for monitoring private clouds. *IEEE Communications Magazine*, 49(12):130–137, 2011. doi: 10.1109/MCOM.2011.6094017.

[20] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. doi: 10.1017/9781108679930.

[21] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano. Devops. *IEEE Software*, 33(3): 94–100, 2016. doi: 10.1109/MS.2016.68.

[22] Bilal Fadlallah, BD Chen, Andreas Keil, and Jose Principe. Weighted-permutation entropy: A complexity measure for time series incorporating amplitude information. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 87:022911, 02 2013. doi: 10.1103/PhysRevE.87.022911.

[23] Ben D. Fulcher. Feature-based time-series analysis. *CoRR*, abs/1709.08055, 2017. URL `http://arxiv.org/abs/1709.08055`.

[24] Matthias Gander, Michael Felderer, Basel Katt, Adrian Tolbaru, Ruth Breu, and Alessandro Moschitti. Anomaly detection in the cloud: Detecting security incidents via machine learning. volume 379, pages 103–116, 01 2013. ISBN 978-3-642-45259-8. doi: 10.1007/978-3-642-45260-4_8.

[25] L. P. F. Garcia, A. C. Lorena, M. C. P. de Souto, and T. K. Ho. Classifier recommendation using data complexity measures. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 874–879, 2018. doi: 10.1109/ICPR.2018.8545110.

[26] Joshua Garland, Ryan James, and Elizabeth Bradley. Model-free quantification of time-series predictability. *Phys. Rev. E*, 90:052910, Nov 2014. doi: 10.1103/PhysRevE.90.052910. URL `https://link.aps.org/doi/10.1103/PhysRevE.90.052910`.

[27] Anton Gulenko, Marcel Wallschläger, Florian Schmidt, Odej Kao, and Feng Liu. Evaluating machine learning algorithms for anomaly detection in clouds. pages 2716–2721, 12 2016. doi: 10.1109/BigData.2016.7840917.

[28] Tim Gutjahr and Karsten Keller. Ordinal pattern based entropies and the kolmogorov–sinai entropy: An update. *Entropy (Basel, Switzerland)*, 22(1), January 2020. ISSN 1099-4300. URL `https://europepmc.org/articles/PMC7516495`.

[29] Lev Guzmán-Vargas, Alejandro Ramirez-Rojas, and Fernando Angulo-Brown. Multiscale entropy analysis of electroseismic time series. *Natural Hazards and Earth System Sciences*, 8, 07 2008. doi: 10.5194/nhess-8-855-2008.

[30] Jake M. Hofman, Amit Sharma, and Duncan J. Watts. Prediction and explanation in social systems. *Science*, 355(6324):486–488, 2017. ISSN 0036-8075. doi: 10.1126/science.aal3856. URL `https://science.sciencemag.org/content/355/6324/486`.

[31] Yu Huang and Zuntao Fu. Enhanced time series predictability with well-defined structures. *Theoretical and Applied Climatology*, 138, 10 2019. doi: 10.1007/s00704-019-02836-6.

[32] A. Humeau-Heurtier, C. Wu, and S. Wu. Refined composite multiscale permutation entropy to overcome multiscale permutation entropy length dependence. *IEEE Signal Processing Letters*, 22(12):2364–2367, 2015. doi: 10.1109/LSP.2015.2482603.

[33] Rob Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 26, 07 2008. doi: 10.18637/jss.v027.i03.

[34] Rob J Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. otexts, 2 edition, 2018. URL `https://otexts.com/fpp2/`.

[35] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679 – 688, 2006. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2006.03.001. URL `http://www.sciencedirect.com/science/article/pii/S0169207006000239`.

[36] M. S. Islam and A. Miranskyy. Anomaly detection in cloud components. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 1–3, 2020. doi: 10.1109/CLOUD49709.2020.00008.

[37] Dario Jozinović, Anthony Lomax, Ivan Štajduhar, and Alberto Michelini. Rapid prediction of earthquake ground shaking intensity using raw waveform data and a convolutional neural network. *Geophysical Journal International*, 222(2):1379–1389, 05 2020. ISSN 0956-540X. doi: 10.1093/gji/ggaa233. URL `https://doi.org/10.1093/gji/ggaa233`.

[38] Karsten Keller, Teresa Mangold, Inga Stolz, and Jenna Werner. Permutation entropy: New ideas and challenges. *Entropy*, 19(3), 2017. ISSN 1099-4300. doi: 10.3390/e19030134. URL `https://www.mdpi.com/1099-4300/19/3/134`.

[39] Andres Kowalski, M.T. Martin, A. Plastino, and Osvaldo Rosso. Bandt–pompe approach to the classical-quantum transition. *Physica D: Nonlinear Phenomena*, 233: 21–31, 09 2007. doi: 10.1016/j.physd.2007.06.015.

[40] Mahendra Kutare, Greg Eisenhauer, Chengwei Wang, Karsten Schwan, Vanish Talwar, and Matthew Wolf. Monalytics: Online monitoring and analytics for managing large scale data centers. In *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, page 141–150, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300742. doi: 10.1145/1809049.1809073. URL `https://doi.org/10.1145/1809049.1809073`.

[41] Denis Kwiatkowski, Peter C.B. Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How

sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1):159–178, 1992. ISSN 0304-4076. doi: https://doi.org/10.1016/0304-4076(92)90104-Y. URL `https://www.sciencedirect.com/science/article/pii/030440769290104Y`.

[42] A. Lehman, N. O'Rourke, Larry Hatcher, and Edward J. Stepanski. Jmp for basic univariate and multivariate statistics: Methods for researchers and social scientists, second edition. page 123, 2013.

[43] Ling Li. *Data Complexity in Machine Learning and Novel Classification Algorithms*. PhD thesis, USA, 2006. AAI3235581.

[44] Xiaoli Li, Suyuan Cui, and Logan Voss. Using permutation entropy to measure the electroencephalographic effects of sevoflurane. *Anesthesiology*, 109:448–56, 10 2008. doi: 10.1097/ALN.0b013e318182a91b.

[45] Yuxing Li. Reverse weighted-permutation entropy: A novel complexity metric incorporating distance and amplitude information. volume 46, page 6688, 11 2019. doi: 10.3390/ecea-5-06688.

[46] Yuxing Li, Li Yaan, Xiao Chen, and Jing Yu. Denoising and feature extraction algorithms using npe combined with vmd and their applications in ship-radiated noise. *Symmetry*, 9:256, 11 2017. doi: 10.3390/sym9110256.

[47] Yuxing Li, Xiao Chen, Jing Yu, and Xiaohui Yang. A fusion frequency feature extraction method for underwater acoustic signal based on variational mode decomposition, duffing chaotic oscillator and a kind of permutation entropy. *Electronics*, 8:61, 01 2019. doi: 10.3390/electronics8010061.

[48] Yuxing Li, Gao, and Wang. Reverse dispersion entropy: A new complexity measure for sensor signal. *Sensors*, 19:5203, 11 2019. doi: 10.3390/s19235203.

[49] Jun Liu, Shuyu Chen, Zhen Zhou, and Tianshu Wu. An anomaly detection algorithm of cloud platform based on self-organizing maps. *Mathematical Problems in Engineering*, 2016:1–9, 04 2016. doi: 10.1155/2016/3570305.

[50] Ana Carolina Lorena, Luís Paulo F. Garcia, Jens Lehmann, Marcilio C. P. de Souto, and Tin Kam Ho. How complex is your classification problem? A survey on measuring classification complexity. *CoRR*, abs/1808.03591, 2018. URL `http://arxiv.org/abs/1808.03591`.

[51] J. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 475–485, 2013. doi: 10.1109/ASE.2013.6693105.

[52] A. S. Syed Navaz, V. Sangeetha, and C. Prabhadevi. Entropy based anomaly detection system to prevent ddos attacks in cloud. *International Journal of Computer Applications*, 62:42–47, 08 2013. doi: 10.5120/10160-5084.

[53] Fumio Ohi and Tatsuya Suzuki. Entropy and safety monitoring systems. *Japan Journal of Industrial and Applied Mathematics*, 17:59–71, 04 2012. doi: 10.1007/BF03167336.

[54] Y. N. Pan, J. Chen, and X. Li. Spectral entropy: A complementary index for rolling element bearing performance degradation assessment. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 223: 1223 – 1231, 2009.

[55] Pradyot Patil. Ordinalentropy. `https://github.com/pradyot-09/OrdinalEntroPy`, 2021.

[56] Pradyot Patil. Ordinalentropy_simulations. `https://github.com/pradyot-09/OrdinalEntroPy_Simulations`, 2021.

[57] Frank Pennekamp, Alison C. Iles, Joshua Garland, Georgina Brennan, Ulrich Brose, Ursula Gaedke, Ute Jacob, Pavel Kratina, Blake Matthews, Stephan Munch, Mark Novak, Gian Marco Palamara, Björn C. Rall, Benjamin Rosenbaum, Andrea Tabi, Colette Ward, Richard Williams, Hao Ye, and Owen L. Petchey. The intrinsic predictability of ecological time series and its potential to guide forecasting. *Ecological Monographs*, 89(2):e01359, 2019. doi: https://doi.org/10.1002/ecm.1359. URL `https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1002/ecm.1359`.

[58] S M Pincus. Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences*, 88(6):2297–2301, 1991. ISSN 0027-8424. doi: 10.1073/pnas.88.6.2297. URL `https://www.pnas.org/content/88/6/2297`.

[59] William Pourmajidi, John Steinbacher, Tony Erwin, and A. Miranskyy. On challenges of cloud monitoring. *ArXiv*, abs/1806.05914, 2017.

[60] Joshua S. Richman and J. Randall Moorman. Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology-Heart and Circulatory Physiology*, 278(6):H2039–H2049, 2000. doi: 10.1152/ajpheart.2000.278.6.H2039. URL `https://doi.org/10.1152/ajpheart.2000.278.6.H2039`. PMID: 10843903.

[61] Maik Riedl, Andreas Müller, and Niels Wessel. Practical considerations of permutation entropy: A tutorial review. *The European Physical Journal Special Topics*, 222, 06 2013. doi: 10.1140/epjst/e2013-01862-7.

[62] M. Rostaghi and H. Azami. Dispersion entropy: A measure for time-series analysis. *IEEE Signal Processing Letters*, 23(5):610–614, 2016. doi: 10.1109/LSP.2016.2542881.

[63] Mostafa Rostaghi, Mohammad Reza Ashory, and Hamed Azami. Application of dispersion entropy to status characterization of rotary machines. *Journal of Sound and Vibration*, 438:291 – 308, 2019. ISSN 0022-460X. doi: https://doi.org/10.1016/j.jsv.2018.08.025. URL `http://www.sciencedirect.com/science/article/pii/S0022460X18305297`.

[64] Edin Sabic, David Keeley, Bailey Henderson, and Sara Nannemann. Healthcare and anomaly detection: using machine learning to predict anomalies in heart rate data. *AI SOCIETY*, 05 2020. doi: 10.1007/s00146-020-00985-1.

[65] SAID E. SAID and DAVID A. DICKEY. Testing for unit roots in autoregressive-moving average models of unknown order. *Biometrika*, 71(3):599–607, 12 1984. ISSN 0006-3444. doi: 10.1093/biomet/71.3.599. URL `https://doi.org/10.1093/biomet/71.3.599`.

[66] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. doi: https://doi.org/10.1002/j.1538-7305.1948.tb01338.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x`.

[67] Petre Stoica and Randolph Moses. Spectral analysis of signals. *Prentice Hall*, 01 2005.

[68] Sean J. Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018. doi: 10.1080/00031305.2017.1380080. URL `https://doi.org/10.1080/00031305.2017.1380080`.

[69] Jiří Tomčala. Acceleration of time series entropy algorithms. *The Journal of Supercomputing*, 75, 03 2019. doi: 10.1007/s11227-018-2657-2.

[70] Khanh Duy Trinh. *An Introduction to Ergodic Theory*, pages 297–309. Springer Japan, Tokyo, 2014. ISBN 978-4-431-55060-0. doi: 10.1007/978-4-431-55060-0_22. URL `https://doi.org/10.1007/978-4-431-55060-0_22`.

[71] GA US) Talwar Vanish (Campbell CA US) Ranganathan Partha (San Jose CA US) Wang, Chengwei (Atlanta. Cloud anomaly detection using normalization, binning and entropy determination, May 2012. URL `https://www.freepatentsonline.com/y2012/0136909.html`.

[72] Sean Wolfe. Amazon's one hour of downtime on prime day may have cost it between $72 million and$99 million in lost sales, Jul 2018. URL `https://www.businessinsider.nl/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7?international=true`.

[73] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. doi: 10.1109/4235.585893.

[74] D. H. Wolpert and W. G. Macready. Coevolutionary free lunches. *IEEE Transactions on Evolutionary Computation*, 9(6):721–735, 2005. doi: 10.1109/TEVC.2005.856205.

[75] M. Zanin, L. Zunino, O. Rosso, and D. Papo. Permutation entropy and its main biomedical and econophysics applications: A review. *Entropy*, 14:1553–1577, 2012.

[76] H. Zhang and S. He. Analysis and comparison of permutation entropy, approximate entropy and sample entropy. In *2018 International Symposium on Computer, Consumer and Control (IS3C)*, pages 209–212, 2018. doi: 10.1109/IS3C.2018.00060.

[77] Long Zhang, Guoliang Xiong, Hesheng Liu, Huijun Zou, and Weizhong Guo. Bearing fault diagnosis using multi-scale entropy and adaptive neuro-fuzzy inference. *Expert Systems with Applications*, 37(8):6077–6085, 2010. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2010.02.118. URL `https://www.sciencedirect.com/science/article/pii/S0957417410001570`.

[78] Luciano Zunino, Massimiliano Zanin, Benjamin M. Tabak, Darío G. Pérez, and Osvaldo A. Rosso. Forbidden patterns, permutation entropy and stock market inefficiency. *Physica A: Statistical Mechanics and its Applications*, 388(14):2854 – 2864, 2009. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2009.03.042. URL `http://www.sciencedirect.com/science/article/pii/S0378437109002593`.

Appendix A

Glossary

List of Acronyms

PE:	Permutation Entropy
SE:	Sample Entropy
SampEn:	Sample Entropy
AppEn:	Approximate Entropy
WPE:	Weighted Permutation Entropy
RPE:	Reverse Permutation Entropy
DE:	Dispersion Entropy
RDE:	Reverse Dispersion Entropy
RWDE:	Reverse Weighted Dispersion Entropy
ARIMA:	Auto Regressive Integrated Moving Average