Network monitoring and performance assessment: from statistical models to neural networks

Daniel Perdices Burrero

Máster en Investigación e Innovación en las TICs

MÁSTERES DE LA UAM

2019 - 2020

Escuela Politécnica Superior

Universidad Autónoma de Madrid

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

NETWORK MONITORING AND PERFORMANCE ASSESSMENT: FROM STATISTICAL MODELS TO NEURAL NETWORKS

Máster en Investigación e Innovación en Tecnologías de la Información y las Comunicaciones

Autor: Daniel Perdices Burrero

Tutor: Jorge E. López de Vergara Méndez
Co-tutor: Javier Ramos de Santiago
Departamento de Tecnología Electrónica y de las Comunicaciones

Septiembre 2020

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

MASTER'S THESIS

NETWORK MONITORING AND PERFORMANCE ASSESSMENT: FROM STATISTICAL MODELS TO NEURAL NETWORKS

Master's Degree in Information and Communication Technologies Research and Innovation

Author: Daniel Perdices Burrero

Advisor: Jorge E. López de Vergara Méndez
Co-advisor: Javier Ramos de Santiago
Department of Electronics and Communications Technologies

September 2020

NETWORK MONITORING AND PERFORMANCE ASSESSMENT: FROM STATISTICAL MODELS TO NEURAL NETWORKS

Author: Daniel Perdices Burrero
Advisor: Jorge E. López de Vergara Méndez
Co-advisor: Javier Ramos de Santiago

High Performance Computing and Networking Research Group
Departament of Electronics and Communications Technology
Escuela Politécnica Superior
Universidad Autónoma de Madrid

September 2020

Resumen

Resumen En los últimos años, las redes de comunicaciones han empezado a jugar un papel fundamental en muchos ámbitos. Las empresas han evolucionado también en torno a Internet, aprovechando esta capacidad de difusión. No obstante, esta dependencia en las redes y en las infraestructuras supone también un problema para las empresas. Los sistemas de comunicaciones se han vuelto un elemento crítico del negocio, suponiendo perdidas económicas graves en caso de interrupción del servicio.

En este contexto, se requiere proveer de modelos que permitan dimensionar y caracterizar las redes de comunicaciones. Centrándose en el modelado, uno tiene distintas opciones: desde aproximaciones clásicas basadas en estadística hasta alternativas más recientes basadas en aprendizaje profundo y aprendizaje automático. En este trabajo, se quiere realizar un estudio de los diferentes modelos en contextos distintos, prestando atención a las ventajas y desventajas para proporcionar la mejor solución en cada caso.

Para cubrir el mayor espectro, se han estudiado tres casos: fenómenos no sensibles al tiempo, donde prestaremos atención al sesgo y la varianza de los modelos, fenómenos dependientes del tiempo, donde será importante extraer las tendencias de las series, y procesamiento de texto, empleando modelos para procesar atributos obtenidos mediante DPI. Para cada uno de estos casos, se han estudiado alternativas y propuesto soluciones que además de probarse con datos sintéticos, se ha probado con datos reales, mostrando el éxito de las propuestas.

Palabras clave redes de comunicaciones, modelado de datos, análisis de datos funcionales, redes neuronales.

Abstract

Abstract In the last few years, computer networks have been playing a key role in many different fields. Companies have also evolved around the internet, getting advantage of the huge capacity of diffusion. Nevertheless, this also means that computer networks and IT systems have become a critical element for the business. In case of interruption or malfunction of the systems, this could result in devastating economic impact.

In this light, it is necessary to provide models to properly evaluate and characterize the computer networks. Focusing on modeling, one has many different alternatives: from classical options based on statistic to recent alternatives based on machine learning and deep learning. In this work, we want to study the different models available for each context, paying attention to the advantage and disadvantages to provide the best solution for each case.

To cover the majority of the spectrum, three cases have been studied: time-unaware phenomena, where we look at the bias-variance trade-off, time-dependent phenomena, where we pay attention the trends of the time series, and text processing to process attributes obtained by DPI. For each case, several alternatives have been studied and solutions have been tested both with synthetic data and real-world data, showing the successfulness of the proposal.

Keywords computer networks, data modeling, functional data analysis, neural networks.

Agradecimientos

This chapter is intentionally in Spanish.

En primer lugar, el agradecimiento institucional a la Universidad Autónoma de Madrid por la ayuda para el inicio de los estudios de máster, así como la de fomento de la investigación (que tuve finalmente que rechazar por mi situación laboral), que me ayudaron a afrontar las tasas. Me gustaría además agradecer a naudit High Performance Computing and Networking por permitirme compatibilizar mi labor de analista con los estudios, así como proporcionar datos reales para todo este proyecto a través de la Cátedra UAM-naudit en redes, sistemas y servicios de altas prestaciones.

Durante este máster que he cursado en dos años compatibilizado con el Máster Universitario en Matemáticas y Aplicaciones, he tenido la oportunidad de aprender bastante sobre temas que, aunque no me apasionaban al principio, al final me han ayudado a ver otros temas más afines a mi área de otra manera. Esto se debe a los profesores que se han esforzado en transmitir no solo los conocimientos sino la motivación y su visión experta de áreas como el procesamiento de texto, el aprendizaje automático o las redes neuronales. Por todo ello, gracias.

En estos tiempos de pandemia, hay que mencionar también la capacidad de adaptación de los docentes, ya que incluso en situaciones muy contrarias para dar las clases online, todos sin excepción han puesto todos sus medios para ello. Además, esto es aún más meritorio tras haber oído de situaciones (en estudios de grado) en las que se ha sustituido un profesor de universidad por un mero y vago PDF.

Me gustaría mencionar a mis compañeros, aunque la verdad es que hacer el máster en dos años hace que pierdas a muchas de tus amistades al pasar al segundo año. Me gustaría mencionar a dos compañeros que, como yo, han estado estos dos años: Adrián y Ainhoa.

Me gustaría agradecer a mis compañeros del C113 también por estos dos años: Tobías, Mario, Rafa, David, entre muchos más. En especial a David, por ayudarme tanto durante el máster y antes de él no solo a ser un mejor investigador sino también a no morir con la burocracia de un doble máster. También, agradecer todo el apoyo a todos los profesores del grupo por la ayuda en muchos momentos, así como las charlas a la hora de comer. También, quiero dar las gracias a mis compañeros de naudit: Germán, Guillermo, Paula, Pablo, José Fernando, ..., que han estado ahí cuando se les necesitaba para aprender aún más de las redes y servicios de las empresas.

En particular, me gustaría además agradecer a mis directores Jorge E. López de Vergara y Javier Ramos el apoyo y paciencia durante estos meses. Con el doctorado, espero ahorraros a todos más dolores de cabeza y ser cada vez mejor.

Tampoco me quiero olvidar de mi familia, mi padre, mi madre y mi perrita Milka, que con todo esto del teletrabajo ha sido difícil, pero hemos podido todo seguir con nuestras vidas y quehaceres.

Por último, te quiero dar las gracias, Ana, por estar ahí a mi lado, por aguantar dolores de cabeza, por pasar una pandemia juntos. Este año que viene te toca a ti pasar por esto, espero poder ser al menos un 1% de lo que tú significas para mí.

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

AE Autoencoders.

CBoW Continuous Bag of Words.

CNN Convolutional Neural Network.

CNN Recurrent Neural Network.

DNS Domain Name System.

DPI Deep Packet Inspection.

FDA Functional Data Analysis.

FPCA Functional Principal Component Analysis.

IDS Intrusion Detection System.

IPS Intrusion Prevention System.

IT Information Technologies.

k-NN k-Nearest Neighbors.

KPI Key Performance Indicator.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

MLP Multilayer Perceptron.

MSE Mean Square Error.

PCA Principal Component Analysis.

PV-DBoW Distributed Bag of Words Model of Paragraph Vectors.

PV-DM Distributed Memory Model of Paragraph Vectors.

RTT Round Trip Time.

TF-IDF Term Frequency - Inverse Document Frequency.

TLS Transport Layer Security.

TTL Time-To-Live.

Introduction 1

1.1 Motivation

Network environments and modeling techniques are continuously evolving. Nowadays, network infrastructures developed from highly static elements to dynamic, reconfigurable and even virtualized devices. Besides, companies and countries invest more and more resources in Information Technologies (IT) and, thus, the infrastructure is becoming a more critical element of the whole business.

On the side of modeling techniques, there are several modern approaches that are gaining momentum. In statistics, functional data analysis is one the most important recent trending topics. In machine learning, deep learning is one the most important advancement in the decade and it has changed the way of seeing other areas as image, video and text processing.

In this context, network monitoring emerges as a necessity, where it is responsible of dealing with the precise performance assessment to optimize cost and efficiency. In this light, it is also clear that it is an open area to brand new modeling procedures. Models need to provide characterization of central behaviors or uncertainty to handle the daily operations of any network. In previous work [Perdices et al., 2018], we have already concluded that, even in a simple case, one single model cannot be the only choice, no matter how generic it is due to the bias-variance trade-off.

Consequently, we consider in this work a journey through different modeling techniques to cope with different Key Performance Indicators (KPIs), paying attention to both complexity and performance, exposing the advantages and disadvantages of each one. We have divided this task into three different stages depending the situation:

1. **Time-unaware models**: as a continuation of [Perdices et al., 2018], we evaluate parametric models in terms of complexity and goodness of fit as well as we extend it with projections to avoid overfitting and to evaluate metrics more closely related to real-life environment. As an example, we will evaluate the delay introduced in a network segment by a network device inside an enterprise datacenter network.

2. **Time-aware models**: for metrics with a strong trend, it is necessary to provide models that take into account the time. This means that instead of considering random variables, we should look at stochastic processes. Autoregressive models

and recurrent neural networks are already a common approach to time-series modeling, but, in this case, we want to characterize global behaviors. So, instead of using these options that rely only in local behaviors, we use Functional Data Analysis (FDA), so samples will be functions or curves and our model will be, as it was early mentioned, a stochastic process.

3. **Models for text-based registers**: up to this point, we have just considered coarse-grained monitoring, this means that many events are inherently undetectable due to the data recollection or the aggregation procedure. To complete this journey, we approach also the fine-grained monitoring and the modeling of unstructured data. In particular, we want to employ Domain Name System (DNS), Transport Layer Security (TLS) or HTTP registers obtained in passive monitoring network probes to characterize the webpages the user is navigating through. Clearly, this has a price: we need to perform Deep Packet Inspection (DPI) to extract the data. Nevertheless, we can get a deep insight into user's behavior.

1.2 Objectives

The main of objective of this document is to design, develop and validate a methodology to build models for different metrics and KPIs that are interesting during network operation. Therefore, we propose different techniques that will be covered thoroughly in the document.

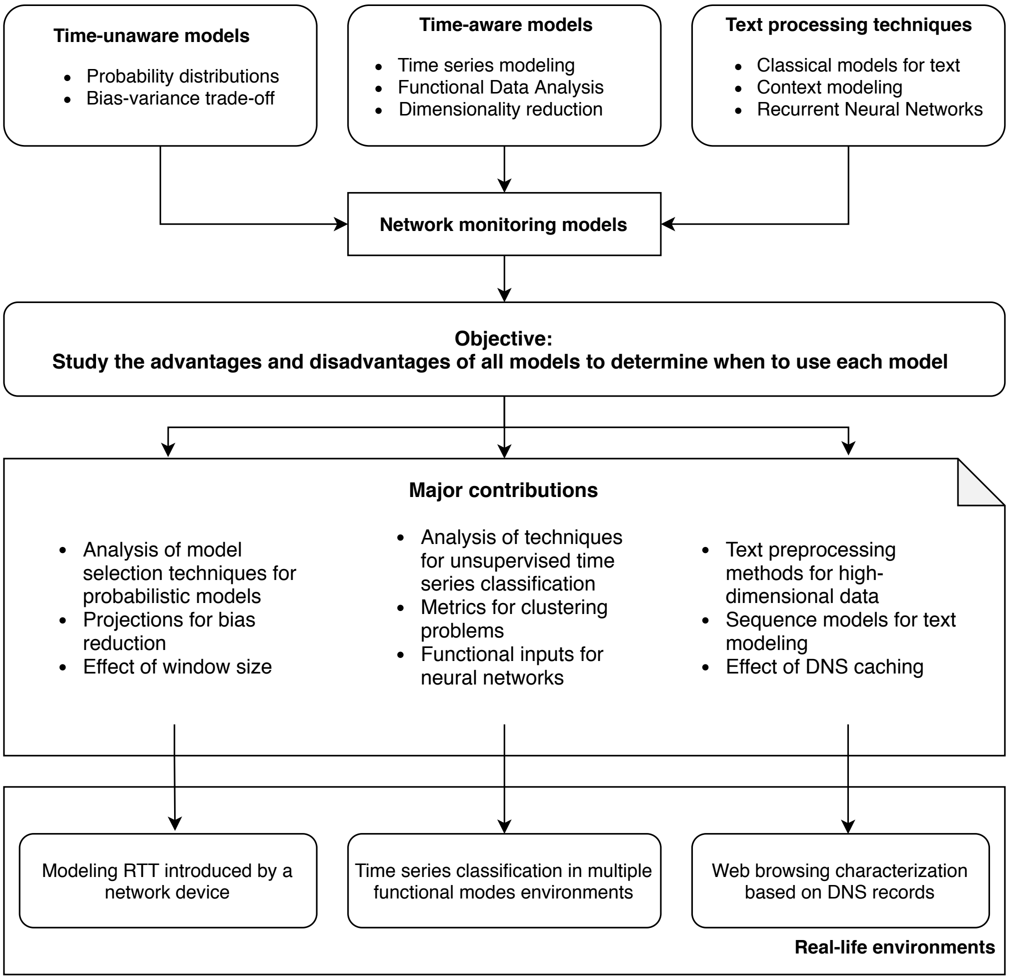

Time-unaware models
• Probability distributions
• Bias-variance trade-off
Time-aware models
• Time series modeling
• Functional Data Analysis
• Dimensionality reduction
Text processing techniques
• Classical models for text
• Context modeling
• Recurrent Neural Networks
Network monitoring models
Objective:
Study the advantages and disadvantages of all models to determine when to use each model
Major contributions
• Analysis of model selection techniques for probabilistic models
• Projections for bias reduction
• Effect of window size
• Analysis of techniques for unsupervised time series classification
• Metrics for clustering problems
• Functional inputs for neural networks
• Text preprocessing methods for high-dimensional data
• Sequence models for text modeling
• Effect of DNS caching
Modeling RTT introduced by a network device
Time series classification in multiple functional modes environments
Web browsing characterization based on DNS records
Real-life environments

Figure 1.1: Diagram of the objectives and contributions of this project.

The partial objectives correspond to the previous parts and they are:

1. Study of time-unaware models.
2. Study of time-aware models.
3. Study of text-based models.

A diagram that illustrates this can be found in Figure 1.1. The central topic in this thesis, instead of being building a model for everything, is to determine when to use each model through its advantages and disadvantages. The figure also gives an early overview of the contributions, which will be detailed through the whole document.

1.3 Project scheduling

Per normative restriction, this project has a limit of 300 hours. Table 1.1 shows the tasks with their time dedication.

Table 1.1: Time plan for the project

ID	Task	Hours
T1	**Time-unaware models**	**70**
T1.1	Analysis of the state of the art	20
T1.2	Trade-off bias-variance and projection methods	30
T1.3	Use case: characterization of latency introduced by a network device	20
T2	**Time-aware models**	**90**
T2.1	Analysis of the state of the art	20
T2.2	Characterization of time series using FDA	20
T2.3	Characterization of time series using neural networks	20
T2.4	Use case: network time series modeling inside a datacenter	35
T3	**Text processing models**	**90**
T3.1	Analysis of the state of the art	20
T3.2	TF-IDF, Doc2Vec and embedding for sequence modeling	35
T3.3	Use case: identification of web browsing trends for ISP clients	35
T4	**Dissemination of the results**	**50**
T4.1	Thesis writing	40
T4.2	Preparation of the defense	10
	Total hours	300

For the first part of the project, we center the objective on the bias-variance trade-off. We want to select models not only in terms of goodness of fit but also taking into account complexity so that the model generalizes. As a practical use case, the latency introduced by several network devices is measured and modeled.

In the second part, time plays a key role. In this case, we expose a generic model that we want to use and different ways of estimating its parameters. For the estimation, the procedures range from FDA approaches to neural networks. As an application, we model the time series of aggregated flow counts per network segment to identify the different trends that exist.

In the last part, we approach a totally different problem. Now, data is formed of text sequences. Usually, text sequences are projected to an embedding space where we study the solution using classical methods. Depending the topology of the embedding, different results will be obtained and, thus, it opens the question of which one to use. This use case is to identify the web browsing domain from the sequence of subdomains observed when monitoring DNS traffic.

1.4 Organization of the document

The document is structured in five more chapters. Chapter 2 contains a light introduction to Neural networks and FDA so that later in the document main concepts are already explained. Next three chapters are focused on the previous topics, i.e. Chapter 3 is about time-unaware models, Chapter 4 introduces the time as a relevant variable and exposes methodologies with FDA and Chapter 5 shows how can we use text-processing techniques and DPI to comprehend the users' behavior. Last, Chapter 6 summarizes the main conclusions of the whole project and next steps to be taken in the future.

Preliminaries

2

2.1 Introduction

This chapter provides a short introduction to FDA and neural network concepts that are used along the document. The first part of the chapter is a summary of concepts related to FPCA theory necessary for chapter 4. The last part consists on a high-level description of the theory of artificial neural networks (hereinafter, neural networks) and recurrent neural networks, which are necessary to understand models and techniques of chapters 4 and 5. With respect to chapter 3, basic probability concepts are required but they are covered in section 2.3 of [Perdices, 2018].

2.2 Functional Data Analysis

First of all, we must define what we call functional data. We will said that some data is functional if it is a random variable that takes values in a functional space such as $L^2([0,T])$. In this kind of spaces, several techniques that work for multivariate data can also be extended to the functional case.

The next two results allow in fact to define a concept akin to the eigenvalue-eigenvector decomposition of a symmetric linear operator. What we want here is to provide the theory that allow Principal Component Analysis (PCA) to work in more general spaces.

Theorem 2.2.1 (Mercer's theorem)**.** *Let K be a continuous symmetric non-negative definite kernel. Then there is an orthonormal basis $\{e_n\}_{n\geq 1}$ of $L^2[a,b]$ consisting of eigenfunctions with nonnegative eigenvalues $\{\lambda\}_{i\geq 1}$ such that K has the following representation:*

$$K(s,t) = \sum_{j=1}^{\infty} \lambda_j e_j(s) \overline{e_j(t)},$$

where the convergence of the series is both absolute and uniform.

Mercer's theorem provides a way of decomposing the covariance operator of a process so that the process in terms of these basis elements, in the same way that a symmetric, positive definite matrix can be diagonalized. This result is known as Karhunen-Loève Theorem and it is fundamental for Functional Principal Component Analysis (FPCA).

Theorem 2.2.2 (Karhunen-Loève Expansion)**.** *Let $\{X(t);\ a \leq t \geq b\}$ be an L^2 process with continuous covariance function K and mean $m = 0$. Let $\{e_n\}_{n\geq 1}$ be an orthonormal basis for the space of eigenfunctions with nonzero eigenvalues of the kernel K and $\{\lambda_n\}_{n\geq 1}$ the corresponding eigenvalues. Then $X(t)$ can be expressed as*

$$X(t) = \sum_{n=1}^{\infty} Z_n e_n(t), \qquad a \leq t \leq b$$

where $Z_n = \langle X(t), e_n(t)\rangle$ are orthogonal random variables with $E(Z_n) = 0$ and $E(|Z_n|^2) = \lambda_n$ and this series converges in L^2 uniformly in t.

Chapter 4 will provide more details of when to use these results to understand FPCA.

2.3 Neural Networks and Deep Learning

2.3.1 Description

Most of the deep learning courses introduce the topic from a historical point of view. Nowadays, deep learning and neural networks are part of the state of the art of many fields, so they need no motivation. The main question then is: why are they so popular? The answer is that these models are usually able to resemble human thinking, adjust the data as much as you decide and overfitting strategies.

Despite the benefits, neural networks are yet being explored in the state of the art of machine learning and there are yet many breakthroughs to come that will stir up other areas as it happened with Convolutional Neural Network (CNN) and image processing, and with Recurrent Neural Network (CNN) and text processing.

Neural networks are composed on layers and neurons. First, we define both

Definition 2.3.1 (Neuron and linear layer)**.** *A standard neuron or neuron with input size k is just an affine operator*

$$\begin{aligned} y:\ & \mathbb{R}^k \to \mathbb{R} \\ & x \to y(x) = wx + b \end{aligned}$$

where w is a vector in $\mathbb{R}^k$ and b_0 is a constant called the bias. W and b are called weights of the neuron.

A linear layer, also called dense layer, with input size k and output size d is a set of d independent neurons. It can be seen also as another affine operator

$$\begin{aligned} Y:\ & \mathbb{R}^k \to \mathbb{R}^d \\ & x \to Y(x) = Wx + b \end{aligned}$$

where W is a $d \times k$ matrix and b is a vector of size d.

Linear layers are the essential building block for neural networks. Nevertheless, the composition of linear operators is just another linear operator; so, we need extra elements. These elements are the activation layers. They allow neural networks to adapt to some particular data such as discrete probability distributions or to build non-linear operators when composed with linear layers.

Definition 2.3.2 (Activation layer)**.** *An activation layer with input dimension k is just a function f from $\mathbb{R}^k$ to $\mathbb{R}^k$.*

Examples of activation layers are:

1. Sigmoid function: classical function whose range goes from 0 to 1 and that near 0, its values is 0.5, so it can model a probability.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

2. Hiperbolic tangent: classical function also which main difference is just that it has 0 mean and the range is $[-1, 1]$

3. Softmax function: a useful activation layer that normalize the outputs so that the outputs are all positive and add to 1.

$$\sigma(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_i e^x} \tag{2.2}$$

4. Rectified Linear Unit (Relu): just a function that eliminates negative outputs.

$$\sigma(x) = \max(0, x) \tag{2.3}$$

Some of them (e.g. the first one and the second one) were deduced from the biological conditions of a neuron. Others, as the second one and third one, were just proposed to deal with processing the data so data is under some restrictions, in this case being a probability distribution. Besides them, researchers proposed other ones, as the fourth one, to cope with the disadvantages of other ones, in this case the decay of the gradient for deep networks.

Figure 2.1 shows the shape of some of these functions. As we see, sigmoid and tanh are smooth functions whereas ReLU is non differentiable at 0. As we will cover next, derivatives play a important role in the training process.

2.3.2 Training procedure: feed-forward and back-propagation

Once models are defined, we need to specify how the models are trained, this is, how we obtain the parameters and weights of the different layers. This procedure is performed in two steps: the feed-forward step and the back-propagation step.

Both steps are repeated per batch, a subset of the sample that implies the update of the parameters. Batch size is usually a hyperparameter that suppose a trade-off between time and convergence. Smaller batch sizes will likely converge faster, especially in the first steps, whereas bigger batches sizes will require more steps to achieve the same accuracy. On the other hand, smaller batch sizes do not take advantage of vectorization and, thus, the parallelization will be fine-grained instead of coarse-grained. So, it results in larger training times.

For a batch, the feed-forward step consists in computing the output of each layer based on the input or the output of other layers and storing the result in the layer

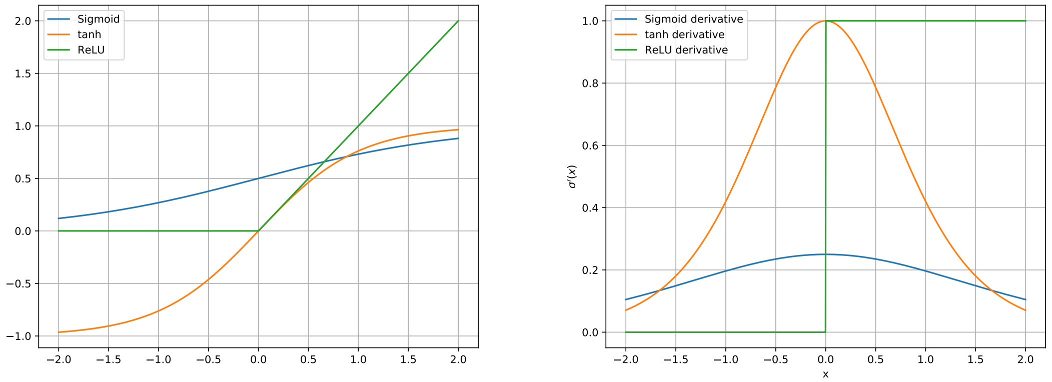


Figure 2.1: Activation layers and their derivatives for $x \in [-2, 2]$

structure as it will be used in the next step. While this task is trivial, it is necessary to the next step.

The back-propagation step is far more complicated. First, all the parameters are trained through modifications or adaptions of gradient descent, i.e. if the weights are w, we have an iterative formula

$$w_{k+1} = w_k + \gamma \nabla \varepsilon(w_k), \tag{2.4}$$

where γ is the learning rate (a hyper-parameter) and $\varepsilon(w_k)$ is the cost function.

Learning rate is usually between 0.0001 and 0.01 but it depends on the magnitude of the gradient. Here is where normalization and pre-processing can help to bound the magnitude of the gradient. Also, modified methods are employed where this learning rate is modified along the iterations implementing concepts as momentum [Kingma and Ba, 2015] or regularizations.

On the other hand, cost functions are usually a fixed parameter of our model. In this case, training process aims at minimizing ε, which is usually the mean of the loss function, i.e.

$$\varepsilon(w) = \frac{1}{N} \sum_{i=1}^{N} \text{loss}(f_w(x_i), y_i), \tag{2.5}$$

where x_i are the inputs of our model, f_w is the neural network as a function of the inputs and y_i the targets, which can be either categorical (classification problems) or numerical (regression problems).

Although loss functions are pretty open, one must keep in mind that we are doing gradient descent, so we need that the cost function, and thereby the loss function, is differentiable. This makes unfeasible to use accuracy or precision, so instead one must use the cross-entropy

$$\text{loss}(\hat{y}, y) = -\sum_{l \in \text{Supp}(y)} p(l) \log(q(l)) = -\sum_{l \in \text{Supp}(y)} \mathbb{1}_{\{y=l\}} \log(\hat{P}(Y = l)), \tag{2.6}$$

where p and q are the original distribution ($p(l) = P(Y = l)$) and the estimated distributions ($q(l) = \hat{P}(Y = l)$) respectively.

For regression problems, Mean Square Error (MSE) is usually the favorite one

$$\mathrm{loss}(\hat{y}, y) = \frac{1}{d}\sum_{j=1}^{d}(y_j - \hat{y}_j)^2, \tag{2.7}$$

but many other possibilities as Mean Absolute Error (MAE)

$$\mathrm{loss}(\hat{y}, y) = \frac{1}{d}\sum_{j=1}^{d}|y_j - \hat{y}_j| \tag{2.8}$$

are possible even if they are not differentiable everywhere.

The last ingredient to completely understand the back-propagation is the chain rule and the automatic differentiation systems. We talked about the gradient but computing the gradient by hand could be extremely hard for some kind of fancy networks when they get deep and deep. Nevertheless, back-propagation is as easy as the chain rule. Recall that the chain rule is

$$\frac{d\varepsilon}{dw}(\varepsilon(y, \hat{y}_w)) = \underbrace{\frac{d\varepsilon(y, \hat{y})}{d\hat{y}}(\hat{y}_w)}_{(A)}\underbrace{\frac{d\hat{y}_w}{dw}}_{(B*)} = \underbrace{\frac{d\varepsilon(y, \hat{y})}{d\hat{y}}(\hat{y}_w)}_{(A)}\underbrace{\frac{d\hat{y}(f)}{df}(f_w^{(h)})}_{(B)}\underbrace{\frac{df_w^{(h)}}{dw}}_{(C*)}. \tag{2.9}$$

As we see, the gradient can be calculated per layer basis. In this case, always the left-hand side terms are the same derivatives (so they are always the same) and since they are computed from back to front of the network, we called it back-propagation. As we highlighted in the feed-forward step, values as $\hat{y}_w$ and $f_w^{(h)}$ were computed before so they can be just saved in the feed-forward step as we explained.

2.3.3 Validation: hyperparameters tuning

Although neural networks are one of the most powerful models nowadays, they are very sensible to the so-called hyperparameters. These hyperparameters configure the network and the training procedure.

For the configuration of the network, the possibilities are limitless in terms of number of neurons per layer, number of layers, types of layer, etc. This means that, without a methodology, fitting a neural network can be impossible. As a consequence, neural network models are usually built using a iterative approach where different parameters are tested.

Cross-validation procedures do also apply for neural networks, but they require so many computational resource that they are not so common. Normally, random search or grid search are the most frequent options.

2.3.4 Extensions of the classical layers: recurrent neural networks

The boom of the neural networks was initiated with just Multilayer Perceptron (MLP) architectures (sequentially connected linear layers and activation functions). Applications for image processing or text processing caused the invention of new type of layers as convolutional layers and recurrent layers. Since for text processing, the most common ones are recurrent layers, we will study deeply the architecture of most used recurrent neuron: the Long Short-Term Memory (LSTM) cell.

Recurrent neural networks usually implement a feedback loopback where information h_n obtained when processing element of the sequence s_n is propagated to process element s_{n+1}. Left-hand part of Figure 2.2 presents the feedback loopback that is modeling whereas right-hand part of the figure shows the real implementation.

Bear in mind that, although this kind of neurons are implemented as a sequence of cells, they all share the same parameters. Also, since neural networks usually have fixed sizes due to optimizations, we have to fix a length of the sequence or window size k.

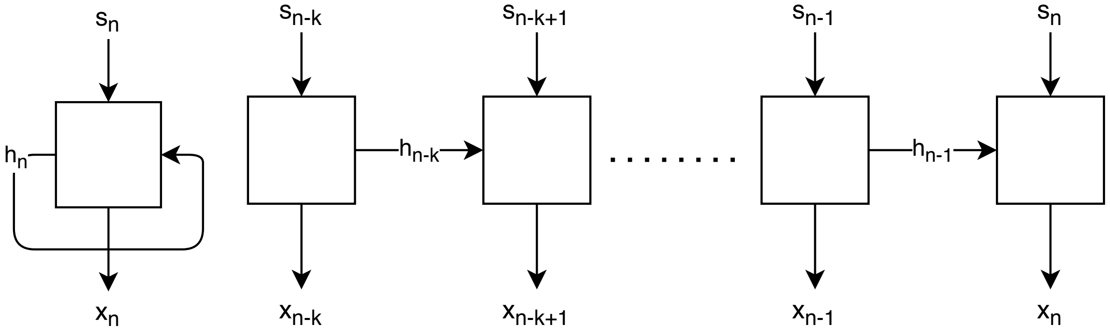


Figure 2.2: Fundamental concept of RNN and its implementation

Once this concept of RNN is clear, the next step is to study particular implementations of this. The most popular one is the LSTM cell. The equations that described the LSTM cell can be found in (2.10)-(2.16)

$$x_n = o_n \cdot \tanh(c_n) \tag{2.10}$$
$$h_n = x_n \tag{2.11}$$
$$c_n = f_n \cdot c_{n-1} + i_n \cdot g_n \tag{2.12}$$
$$o_n = \sigma(W_{so}^t s_n + W_{so}^t h_{n-1} + b_o) \tag{2.13}$$
$$f_n = \sigma(W_{sf}^t s_n + W_{sf}^t h_{n-1} + b_f) \tag{2.14}$$
$$i_n = \sigma(W_{si}^t s_n + W_{si}^t h_{n-1} + b_i) \tag{2.15}$$
$$g_n = \tanh(W_{sg}^t s_n + W_{sg}^t h_{n-1} + b_g) \tag{2.16}$$

As it can be inferred from the equations, state is propagated to the next element of the sequence through h_n and c_n. While c_n represents the long-term state of the sequence, h_n models the short-term memory, consequently they are called LSTM neurons. Besides, f, i, o look like classical neurons and each of them is for different purposes: forget, input and output respectively. As we see, when f is close to zero, it forgets the long-term state. Also, o_n is the output itself weighted and i_n is a hidden layer from the input that it also impacts on the output.

2.4 Conclusion

In the first part of the chapter, we have explored basic concepts of FDA and neural networks so that it serves as a short introduction for readers who are not familiar with them and it also makes the document more self-contained so this chapter can be referenced in the rest of the document without requiring readers to look for the results.

3 Time-unaware modeling: probabilistic models and trade-off bias-variance

3.1 Introduction

Statistical modeling is a key step in many management tasks. The choice of one model over another one can lead to radically different criteria and conclusions. Consequently, this section presents a collection of models that are suitable for situation where there is no seasonality in the data or the model is only employed in short time intervals. Furthermore, this section explains different principles to select a model taking into account both bias and variance. To close this chapter, we present an idea of using projections and the previous models to deal with situations with high variance.

A key example of this kind of metrics that we expect to be time-invariant is hardware capabilities of devices, for instance, the latency or Round Trip Time (RTT) that introduces a firewall in the connections. Using probabilistic models, we fit a distribution to our data so that we have a fair model of our phenomena that will help us to provide a more reliable description of the extreme behaviors that we may observed.

In this chapter, we explain dPRISMA (Distributed Passive Retrieval of Information, and Statistical Multi-point Analysis), a passive monitoring system we designed to study the RTT in multi-hop environments. It employs probabilistic models and projections to provide the most suitable model. Multi-hop analysis allows to locate where performance issues are likely to appear and projections helps to reduce the variance we will observed in single-flow estimates for large periods of time.

3.2 State of the art

We start with a review of statistical models for RTT measurements, to justify the selection of the models in our system. Then, we consider previous results that ground the assumption of validity of this representation, and methods to collapse individual flow estimates and obtain indicators of vantage points' performance. Finally, we focus on other monitoring frameworks that share design principles with our proposal.

The content of this chapter is based on [Perdices et al., 2018, Perdices et al., 2019]

Statistical modeling of network KPIs has deserved much attention, given its importance for network operation. This interest has resulted in a vast amount of literature reporting how different probability distributions represent network measurements, which extends to delay and RTT modeling.

Given their central position in inference, probability theory and empirical research [Mandel, 1984], normal and lognormal models are a common approach when coping with data analysis. However, the research of KPIs in operational networks has exposed that many times they exhibit heavy-tailed behaviors in existing deployments, which grounded the exploration of more complex models able to capture large deviations [Liebeherr et al., 2012, Simmross-Wattenberg et al., 2011, Carisimo et al., 2017]. As we will detail in the following sections, we consider several parametric models (some of them with heavy tails) and compare their performance, taking into account different metrics to optimize the trade-off between goodness of fit and complexity.

In [Papagiannaki et al., 2003], the authors explored which distribution adjusted single-hop delays in computer networks. Their conclusions pointed to a good representation of this KPI with Weibull distributions, as delays presented fair unimodal behaviors. Similar results were reported in [Hernández and Phillips, 2006], while in this latter case multi-modal behaviors were observed (somehow expectable, as that work analyzed end-to-end delays) so mixtures of Weibull distributions provided good fitting to the measurements. Inspired by these results, we explored two additional parametric families, which for some values in the space of parameters lay near Weibull distributions.

On the one hand, we have considered the Generalized Extreme Value (GEV) distribution [Coles et al., 2001], given their suitability to represent variables with large and rare values. Remarkably, GEV distributions generalize Weibull, Gumbel and Fréchet distributions, which motivates the selection of this model. On the other hand, we also introduced Burr Type XII distributions to model RTT, motivated by the relation of this parametric family with Weibull distributions [Tadikamalla, 1980]. The complexity of both models is comparable to Weibull distributions, but their broader flexibility can potentially reduce deviant cases.

Additionally, in recent times α-stable distributions have been applied to model RTT [Carisimo et al., 2017]. This family is very flexible and general, but much more complex that those previously commented. In fact, the fitting of the parameters of α-stable distributions is computationally expensive [Royuela-del-Val et al., 2017, Julián-Moreno et al., 2017] and there is no closed expression for their density function. Remarkably, α-stable distributions appear in the generalized central limit theorem and converge to normal distributions for some values in the space of parameters.

As stated above, the properties of several parametric distributions offer a promising framework to describe RTT and delay components in operational networks. While this is a primary step along data modeling, model fitting requires a time-dependent consideration to guarantee that such representation can exist. In other words, we wonder to what extent measurements are stable—i.e., they follow a given distributional law during observations periods.

RTT can be decomposed, as we develop below, in delay components that affect network traffic along a path. Therefore, a reasonable condition for RTT stability is the stability of those components. Given the importance of such components in overall network performance, several previous works have addressed its modeling and understanding. For instance, a formal model for stochastic components of delay in common network equipment is presented in [Hohn et al., 2004]. In that work, the authors pointed to relevant factors—remarkably, network load and node capacity—that provided a suitable estimator for delay components.

In that same line, we consider that under stationary network load and in the absence of changes in node capacity, delay components can be considered *short-term invariants*: this seems a reasonable condition, as a result of empirical and grounded analysis of network load [Mata et al., 2012]. In this latter work, authors analyzed a method for the detection of abrupt short-time changes in network load, showing that under very general assumptions this indicator can be considered invariant. Then, they applied cluster aggregation to test the model compliance—specifically, multivariate Gaussian model—and detected excursions from the typical behavior.

Following a somehow similar approach, we put together these previous results to define a projection method to pre-filter individual flow estimations in time windows, selecting representatives for central and extreme values. Then, we characterize the typical behavior of projections using the aforementioned models. Additionally, we also introduce a control measurement to assure stability of the estimates using Dictyogram [Muelas et al., 2015]. This method describes the evolution of flow characteristics by accounting the frequency of their values within a set of order statistics. Hence, it provides a flexible evaluation of changes in the distribution with the analysis of its corresponding variation rates.

Beyond improving techniques to retrieve information from measurements, network monitoring and analysis solutions need to suit novel operational architectures. This entails that data capture and deployment processes should evolve towards more scalable and flexible approaches. As an illustrative situation, current trends regarding network slicing and virtual networks on top of shared hardware require this type of approaches to gather data without incurring in high costs—*e.g.*, movement of big data volumes. This matter is not a particularly new concern for network monitoring, and many previous results explored principles that can help to improve current systems.

For instance, the design of cooperative monitoring systems [Xu and Wang, 2008] arose as a promising approach to alleviate the shortcomings of monitoring scalability. These classical ideas can pave the way for improved solutions in the network monitoring scope, as stated in [Bari et al., 2013].

Even more important is that many current network monitoring efforts are focused on how to take advantage of the ever-increasing capabilities of network equipment. This opens the gate to disaggregate network monitoring, moving specific tasks to the most suitable equipment in the network. Turboflow [Sonchack et al., 2018] is a recent proposal that relies on the embedding of flow generation into programmable switches. However, the authors of that work highlight that stateful information may limit the complete implementation of some processes in the network hardware. In the same line, Sonata [Gupta et al., 2016] distributes monitoring tasks to different network elements, providing a query-based API that can be exploited by other modules. Parallel to these proposals, our approach provides high-level analytics after aggregation and correlation of traffic packets or flow records that may be produced by different sources and methodologies.

Finally, and regarding the trends in virtualization and software-definition of networks, we point to other recent works that exploit containers to define flexible monitoring services that can be instantiated on demand and linked to specific applications [Moradi et al., 2017]. The modular design of our system in chapter 3 is totally aligned with these trends, providing a higher decoupling of data gathering and analytics.

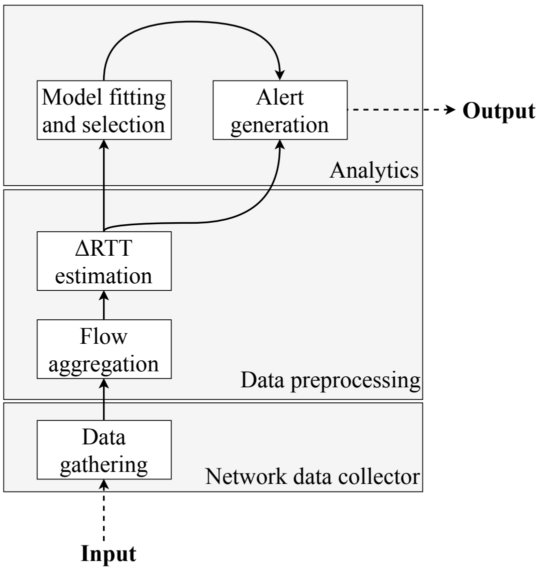


Figure 3.1: Functional modules of dPRISMA.

Such approaches can push network monitoring proposals toward microservice-oriented architectures [Holger et al., 2016].

3.3 Fine-grained modeling

Since we are focusing on the example of the RTT of a network device, we introduce dPRISMA [Perdices et al., 2018] (Distributed Passive Retrieval of Information, and Statistical Multi-point Analysis), a passive monitoring system that measures the RTT increment introduced by a device or a network segment using just the SYN packets in the TCP handshake or, alternatively, Netflow [Claise, 2004] or IPFIX [Aitken et al., 2013] records when there is no sampling in the exporters. It is designed to satisfy these principles:

1. *Distributed and passive data gathering:* the retrieval of information should be distributed among different network elements. Monitoring systems should exploit capabilities of the equipment to improve scalability with a horizontal division of tasks. This can be implemented using several functionalities of common network equipment. For instance, we point to opportunistic retrieval from built-in capabilities (*e.g.*, exploitation of OpenFlow records); existing passive monitoring elements (*e.g.*, NetFlow or IPFIX exporters); and traffic forwarding based on SPAN ports or selective OpenFlow rules.

2. *Correlation of multi-point measurements:* measurements should be exploited to provide contextual data and link observations from different elements. As network issues usually affect complete segments, measurements that encompass only single points can hide the location, extension and nature of the problems. Therefore, correlation of measurements can provide deeper insights into performance issues and network state.

3. *Application of statistical models:* stochastic nature of network measurements requires a suitable statistical modeling. Otherwise, results may not reflect actual network conditions and spurious values can lead to biased decisions. Models should consider a compromise between goodness of fit and complexity, to optimize analytics and prevent unnecessary computational costs.

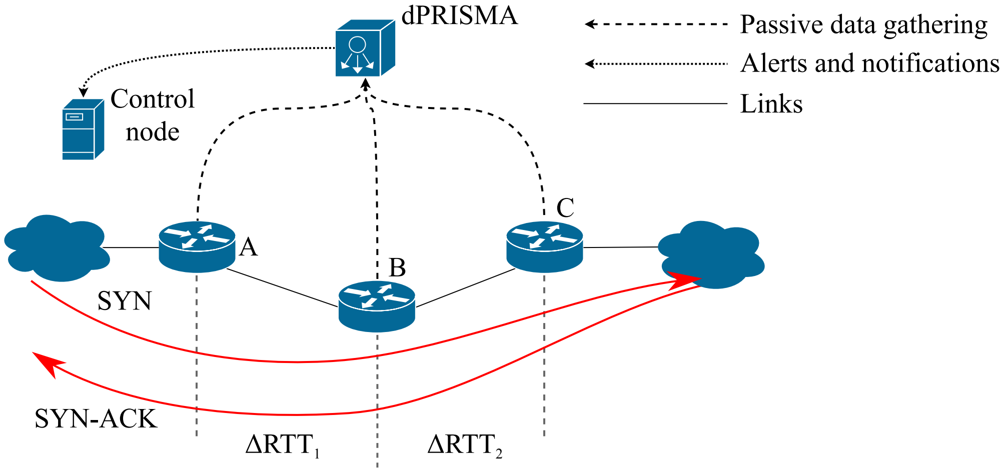


Figure 3.2: Operation of dPRISMA. Red arrows represent a sample connection traversing the three monitored points of the network, distinguishing the different RTT components that are estimated to detect possible bottlenecks.

Next, we describe the main functional components of dPRISMA, which are summarized in Figure 3.1. In the current proof of concept implementation, dPRISMA relies on flow records to conduct the analysis and modeling of RTT. To prevent ambiguities, we clarify that hereafter we refer to *TCP flow* as a set of TCP packets with a common 4-tuple, which traverse a particular vantage point in the network during a specific time interval, as stated in RFC 7011 [Aitken et al., 2013].

Additionally, we synthesize the operation of dPRISMA in Figure 3.2. First of all, passive measurements are gathered from the available vantage points. These measurements are aggregated in dPRISMA and correlated to obtain estimations of RTT and its components—that is, the increments along the network segments defined by vantage points. After that, the system fits and selects the parametric model for measurements, and provides estimations of significant central values—*e.g.*, mean, median and mode— and other order statistics such as extreme values. This leads to flexible and adaptable profiles for alerts, thus providing indicators of performance issues.

In the following, we detail these operations and how they are implemented within the different functional blocks. For our description, we follow a constructive approach that first considers how data are gathered and preprocessed, and then details how they are exploited to build the models.

3.3.1 Data gathering and preprocessing

Flows are collected in several ways. Some examples are Netflow or IPFIX [Claise et al., 2013], and other custom tools that send at least information about when every flow starts. Except for special cases, these timestamps are taken from SYN and SYN-ACK segments, which let us have an estimation of RTT that only requires that both flows of the same connection are sampled. Regarding performance issues in this process, we may distinguish two different situations: flow aggregation in a computing element different to network equipment, and aggregation inside the networking elements. In the first case, it is possible to capture traffic up to 40 Gb/s in commodity hardware—e.g., see [Trevisan et al., 2017, Leira et al., 2019]. In the case of monitoring functions within network equipment, performance issues may appear depending on traffic characteristics and capabilities of specific hardware, while commercial equipment includes support for these operations.

Algorithm 3.1 Flow aggregation.

```
function getSuperflows(flows...)
table $\leftarrow$ InitializeSuperFlowsTable()
for flow in flows do
    if flow is ip and tcp then
        if flow.srcPort < flow.dstPort or
(flow.srcPort = flow.dstPort and
(flow.srcIp < flow.dstIp)) then
            quintuple $\leftarrow$ (flow.srcIp, flow.srcPort, flow.dstIp, flow.dstPort, flow.ipProto)
        else
            quintuple $\leftarrow$ (flow.dstIP, flow.dstPort, flow.srcIp, flow.srcPort, flow.ipProto)
        end if
        table[quintuple].addFlow(flow)
    end if
end for
return(table)
```

dPRISMA estimates RTT by subtracting the start times of two TCP flows that share temporal and spatial localities, and the 4-tuple swapping source and destination addresses and ports. Then, it correlates *equivalent flows*: TCP flows sharing the 4-tuple and time interval but observed in different points of presence. This process is described in Algorithm 3.1.

Once RTT is estimated and correlated, the *equivalent flow* contains information of the flow in several locations. By looking at Figure 3.2, we observe that RTT in hop j is given in (3.1):

$$\text{RTT}_j = \sum_{i=j}^{N} \Delta\text{RTT}_i \tag{3.1}$$

By inverting this linear operator, we obtain an estimation of the component in the network segment between vantage point j and $j+1$ as in (3.2):

$$\Delta\text{RTT}_j = \text{RTT}_j - \text{RTT}_{j+1} \tag{3.2}$$

Note that, in contrast to one-way delay measurements, these estimations do not require clock synchronization, since the RTT_j are absolute values, each one computed with the flow estimates performed in the same vantage point j with its local clock as a single reference time. As this process involves a single clock source, synchronization among different vantage points is not required.

3.3.2 Model selection

Due to the stochastic nature of network measurements, statistical models are needed. In our case, these models are intended to characterize ΔRTT_j behavior, so that frequent events can be distinguished from anomalies or deviant events.

Apart from how challenging model fitting can result, the selection of an optimal model to be used emerges as key matter for systems as ours. For this aim, we have equipped dPRISMA with several criteria, summarized in Table 3.1, to adapt its behavior to a wide range of situations:

Table 3.1: Summary of metrics for model selection.

Metric	Description	Expression
$\mathbf{R}^2$	Only considers fitting.	$1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$
AIC	Considers both fitting and number of parameters.	$2(k - \log(\hat{L}))$
BIC	Considers fitting, number of parameters and sample size.	$\log(N)k - 2\log(\hat{L}))$

1. *Coefficient of Determination (*$\mathbf{R}^2$*)*: A well-known metric of goodness of fit is the coefficient of determination, $\mathbf{R}^2$. This metric is based on a linear fitting of (x_k, y_k), where x_k are the order statistics of the sample and y_k are the corresponding quantiles of the model. If the samples follow the model, there must be a strong linear relation, which entails that $\mathbf{R}^2$ must be close to 1. This is a necessary but no sufficient condition [Kilpi and Norros, 2002], so although this method cannot provide a formal proof of goodness of fit, it can be applied to rule out the parametric models with the lowest values—*i.e.*, to select that with the strongest linear relation between the order statistics of the sample and estimated distribution.

2. *Akaike Information Criterion (****AIC****)*: This a statistical method to compare different models based on two factors: complexity and goodness of fit. It has the expression in (3.3):

$$AIC = 2(k - \log(\hat{L})) \tag{3.3}$$

   where k is the number of parameters of the model and $\hat{L}$ is the maximum of the likelihood function [Akaike, 1974]. It is remarkable that complexity is just evaluated with the number of parameters, and this makes it an optimistic approach.

3. *Bayesian Information Criterion (****BIC****)*: Related to the aforementioned AIC, it introduces an additional component, which is the number of samples. This is intended to reduce overfitting in parametric models, so that the complexity and goodness of fit are balanced [Schwarz, 1978]. It is defined as in (3.4):

$$BIC = \log(N)k - 2\log(\hat{L}) \tag{3.4}$$

   where N stands for the sample size and the rest of variables were described in AIC.

These three criteria allow choosing the most appropriate model based on complexity and goodness of fit, and on the situation and requirements of the other top-level system that use this information. For instance, for real-time applications, simpler models are preferred so the model computation is not a bottleneck in the monitoring system.

3.3.3 Mode estimation

The mode of a sample is a prominent centrality measure that returns the most probable value of a distribution. Given that finding a good parametric model is not always feasible, we also evaluated alternative methods to estimate the mode. We have considered methods for the univariate case —see the analysis in the introduction of [Kirschstein et al., 2016]— and studied both indirect (that is, relying on a non-parametric density function estimation) and direct (essentially, search methods around intervals where the mode is likely to appear) proposals:

3.3.3.1 Estimation through the Kernel Density Estimator (KDE)

This approach arises from the definition of mode. First, the KDE, a PDF estimator, is calculated. The mode is estimated as the maximum of the KDE, $\widehat{\text{Mode}}(X) = argmax_{x \in \mathbb{R}} \hat{f}(x)$. While this method can reveal important details about the density function (*e.g.*, shape or number of modes), it depends on the convergence of KDE to the actual PDF.

3.3.3.2 Half-Sample Mode (HSM) algorithm

The HSM algorithm is a robust and fast method to approximate the mode [Bickel and Frühwirth, 2006]. This algorithm is based on the principle that "the mode is in the smaller interval that contains half of the sample". By applying this idea, we reduce both computations and assumptions, making this approach a good one to use in many situations.

3.4 Projections to reduce the variance

In this section, we introduce a modification of the aforementioned system called adPRISMA (Advanced Distributed Passive Retrieval of Information, and Statistical Multipoint Analysis) [Perdices et al., 2019]. This system adds a fourth capability:

4. *Robust data processing:* model fitting needs to include methods to extract relevant information from time-varying measurements. That fact entails a compromise between the granularity of detectable events and resiliency against noisy or isolated excursions.

In order to fulfill this requirement, we performed aggregations of single-flow estimates, described in next subsection. Bear in mind that bias-variance trade-off will be presented since, the more we aggregate, the more we reduce the variance but also, the higher the bias is.

3.4.1 Aggregation of single-flow estimates

Given the high variance of single-flow-based estimates, we envisaged an aggregating procedure to better characterize vantage point modeling. In other words, as adPRISMA is intended to provide indicators for issues at network elements or segments, spurious variance in flow behavior could lead to biased conclusions. To separate this latter type of deviant situations from sustained changes in the vantage points' behavior, we have introduced a windowed filtering of single-flow observations.

We recall that adPRISMA tries to obtain a model for the distribution of $\{\Delta\text{RTT}_i\}$ for separate network segments. Therefore, it requires some stability on the fitted distribution. In this regard, we detected two main issues that may appear because of the flows' stochastic behavior. On the one hand, changes on the underlying distribution can lead to sub-optimal adjustments—e.g., a sustained change on the expectation of $\{\Delta\text{RTT}_i\}$. On the other hand, the convergence to a fair approximate of the distribution depends on the number of observations, as stated in the Glivenko-Cantelli theorem [Wellner et al., 1977].

This entails a bias-variance trade-off—i.e., the balance between how far the estimated model to the theoretical distribution is, and how the model fits the observations. adPRISMA copes with this matter by clustering single-flow observations in time windows. We distinguished two possible strategies to proceed with this aggregation.

On the one hand, the first strategy is intended to extract a *central* representative within each time window. This would lead to robust models for typical behaviors sustained in time. This can be accomplished using a projection as in (3.5):

$$\Delta\mathrm{RTT}(t) = \underset{x\in\mathbb{R}}{\arg\min} \sum_{j\in\mathbb{J}_t} (d(\Delta\mathrm{RTT}_j, x)),\ t \in \mathbb{T} \tag{3.5}$$

where $t \in \mathbb{T}$ represents the time-domain partition, $\mathbb{J}_t$ the index set of $\{\Delta\mathrm{RTT}_i\}$ within each element of the time-domain partition, and $d(\cdot,\cdot)$ a distance—e.g., any L^p distance. The projection can lead to different *centroids*, such as the median (L^1) or average (L^2) for the observations within the interval. With this, adPRISMA introduces a variance-reduction procedure that can attenuate the effect of flow distortions—i.e., isolated extreme values do not affect model fitting.

On the other hand, the second strategy pursues the determination of boundaries for extreme values. This is accomplished by using order statistics of the observations within each time window as in (3.6):

$$\Delta\mathrm{RTT}_p(t) = \inf\left\{ x : p \le \frac{1}{|\mathbb{J}_t|} \sum_{j\in\mathbb{J}_t} \mathbb{1}_{[0,x]}(\Delta\mathrm{RTT}_j) \right\},\ t \in \mathbb{T} \tag{3.6}$$

where $t \in \mathbb{T}$ represents the time-domain partition, $\mathbb{J}_t$ the index set of $\{\Delta\mathrm{RTT}_i\}$ within each element of the time-domain partition, $\mathbb{1}_A(t)$ the indicator function of set A—i.e. its value is 1 if $t \in A$ and 0 otherwise—and $p \in [0,1]$ indicates the selected probability level. This approach is useful to define and model extreme values' bounds with sensitivity to trends along time.

Bias control on projections is accomplished with the study of convergence to a robust empirical estimate of the theoretical distribution function. Dictyogram [Muelas et al., 2015] offers a basis for quantitative criteria to determine whether the time-domain partition suffices to a reasonable convergence—i.e., if the number of observations offer a fair representation of the distribution. Dictyogram maps time-depending study of the distribution of a flows' characteristic, such as the ΔRTT, onto the analysis of the number of flows lying on categories defined in terms of order statistics of the specific characteristic.

To do so, once the values corresponding to a grid of probability levels $\{x_k\}_{k=1,\dots,N}$ are selected, then flows can be partitioned using the intervals in (3.7):

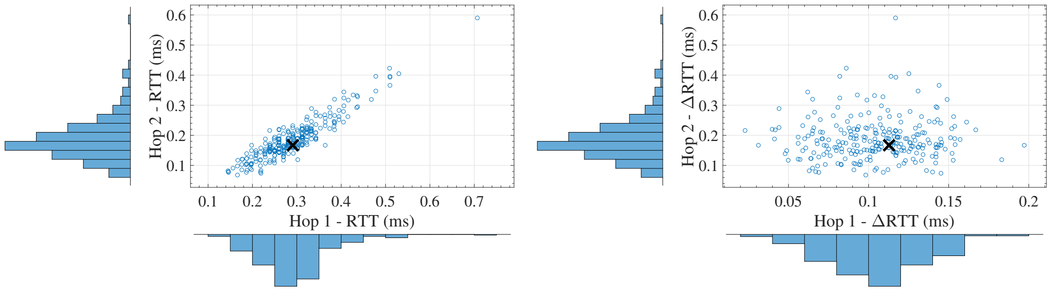
$$\mathcal{I}_k = \begin{cases} [0, x_1] & \text{if } k = 1 \\ (x_{k-1}, x_k] & \text{if } 1 < k < N+1 \\ (x_N, \infty) & \text{if } k = N+1 \end{cases} \tag{3.7}$$

These intervals induce a set of time series with the number of flows within each interval and time window, which we denote as $f_k(t)$ in (3.8):

$$f_k(t) = \sum_{j\in\mathbb{J}_t} \mathbb{1}_{\mathcal{I}_k}(\Delta\mathrm{RTT}_j) \tag{3.8}$$

Using these series, we can define a relative measure of variation along time as presented in (3.9):

$$d[f_k(t)] = \frac{\sum_k |f_k(t) - f_k(t-1)|}{\sum_k f_k(t)},\ t \in \mathbb{T} \tag{3.9}$$



(a) Scatter plot of hops 1 and 2 for RTT.

(b) Scatter plot of hops 1 and 2 for ΔRTT.

Figure 3.3: Results for $Dataset_1$. The × shows the intersection of the modes.

Table 3.2: Estimated mode of ΔRTT_1 and ΔRTT_2 in $Dataset_1$, for each of the methods.

	ΔRTT_1				ΔRTT_2			
Model	**Mode**	**R^2**	**AIC**	**BIC**	**Mode**	**R^2**	**AIC**	**BIC**
KDE	0.1080ms	-	-	-	0.161ms	-	-	-
HSM	0.1124ms	-	-	-	0.167ms	-	-	-
Normal	0.1042ms	0.990	**-4754.719**	**-4747.567**	0.7553ms	0.05	-1728.580	-1721.429
Lognormal	0.0861ms	0.905	-4711.002	-4703.850	0.1223ms	0.295	-4093.103	-4085.950
GEV	0.1055ms	0.992	-4750.049	-4739.321	0.1251ms	0.651	-4251.231	-4240.504
Burr Type XII	0.1063ms	**0.995**	-4754.618	-4743.890	0.1546ms	0.708	-4275.132	-4264.402
α-stable	∼0.1042ms	0.991	-4750.721	-4282.579	∼0.1568ms	**0.970**	**-4296.883**	**-4282.579**

which accounts for the cumulative relative variation of the number of flows in each category.

Then, bursts in $d[f_k(t)]$ are equivalent to abrupt variations in the Empirical Cumulative Distribution Functions (ECDFs) in adjacent time windows. In other words, the stability of this function offers a quantification of ECDFs' stability.

3.5 Results

In the first stage of experiments, we analyzed flow records from a data center network with adPRISMA to assess its outcomes in an actual case study. This dataset, hereinafter denoted $Dataset_1$, has the following characteristics:

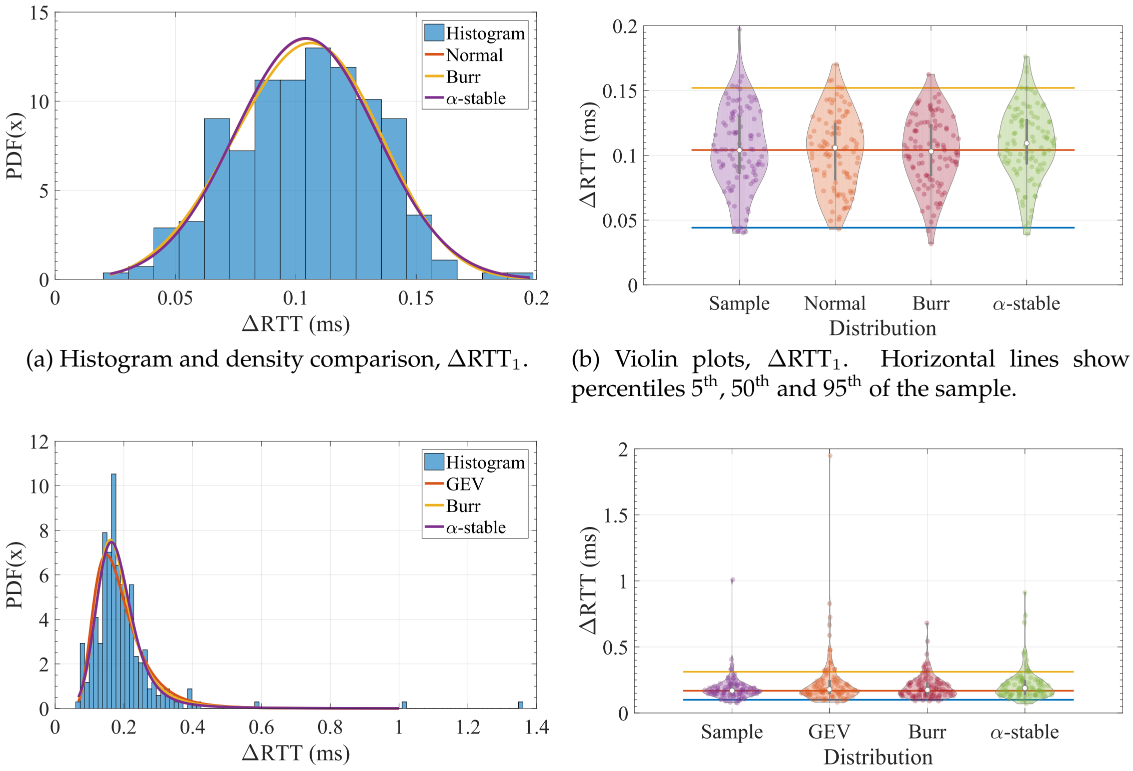
- It includes real traffic traces of core and service switches, load balancers and virtual machines in operation, gathered from an Internet Service Provider (ISP) data center network.

- It was captured using the management software of two vantage points, so no special equipment was completely dedicated to network monitoring.

Due to the presence of some outliers in the second hop of the dataset, some preprocessing was applied to visualize and plot the data. As some of the destinations of the connections are virtual machines, the outliers were likely caused by the hypervisors managing virtual machines.

Finally, the second experimental stage was intended to assess the validity of fixed distribution models along longer observation periods. For this purpose, we consider a second real dataset ($Dataset_2$) retrieved from an enterprise datacenter.

In contrast to $Dataset_1$, $Dataset_2$ is composed of roughly seven working hours of traffic captured in two vantage points deployed to monitor the performance of an in-between operational firewall during working hours in one day. This network presents heavy



(a) Histogram and density comparison, $\Delta\mathrm{RTT}_1$.

(b) Violin plots, $\Delta\mathrm{RTT}_1$. Horizontal lines show percentiles 5th, 50th and 95th of the sample.

(c) Histogram and density comparison, $\Delta\mathrm{RTT}_2$.

(d) Violin plots, $\Delta\mathrm{RTT}_2$. Horizontal lines show percentiles 5th, 50th and 95th of the sample.

Figure 3.4: Comparison among models and observation for $\Delta\mathrm{RTT}_1$ and $\Delta\mathrm{RTT}_2$ in *Dataset*$_1$.

traffic load during the peak hour around 9:00 AM, and some other minor peak moments during the rest of the day. Whereas *Dataset*$_1$ lasted for only some few minutes, this latter case represents a scenario where single-flow estimates might not be stable along time. With this, we assessed the principles that grounded the projection methodology to obtain node indicators from individual flow estimates—trying to capture a fair representation of the global node performance.

3.5.1 Short-term device monitoring

Once we have assessed the accuracy of adPRISMA, we inspect the results obtained during the study of a real data center network. In a similar way to the previous experiments, we present scatter plots of RTT and ΔRTT in Figure 3.3 and summarize the results of model fitting and mode estimation in Table 3.2. Additionally, we include in Figure 3.4 the representation of sample data compared to the three models that provided the best goodness of fit. Figure 3.4a and 3.4c present the comparison among the estimated densities and the normalized sample histogram, and Figures 3.4b and 3.4d depict the corresponding violin plots with some remarkable order statistics—specifically, the median as centrality measure, and the 5th and 95th percentiles for extreme values.

For $\Delta\mathrm{RTT}_1$ (*i.e.*, measurements in the first vantage point), Burr Type XII model obtained the highest $\mathbf{R}^2$, whereas **AIC** and **BIC** suggest that a normal model is also reasonable and much less complex. This behavior is coherent with the insights from Figure 3.4b, where Burr Type XII presents higher accordance with the order statistics of the sample, while the adjusted normal model fairly fits the sample distribution.

However, the behavior of $\Delta\mathrm{RTT}_2$ (*i.e.*, measurements in the second vantage point) is very different. In this case, the preferred model is the α-stable distribution, with better

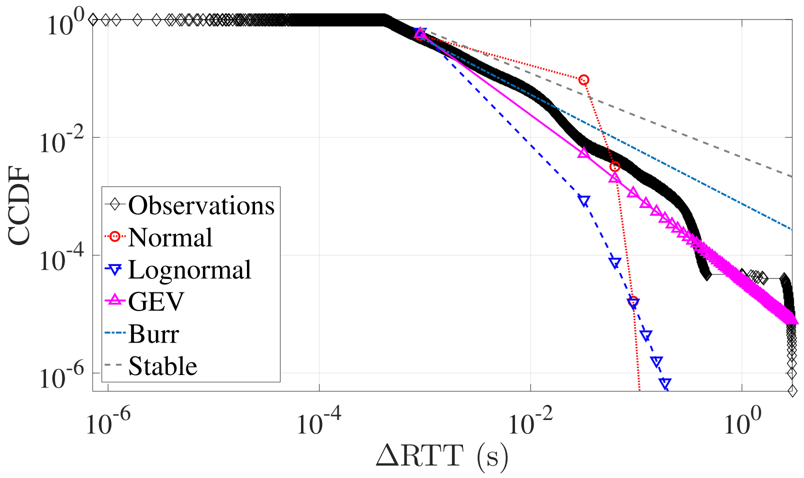


Figure 3.5: ΔRTT distribution, single-flow estimates, *Dataset*$_2$.

scores (either when considering $\mathbf{R}^2$, **AIC** or **BIC**) for any other option. The skewness and tail of $\Delta\mathrm{RTT}_2$ prevent from considering more simplistic models, with poor accuracy in the representation of the shape and order statistics of the sample distribution—see Figures 3.4c and 3.4d for illustration.

This situation exposes two important matters. First, this dataset provides evidences of disparity in the behavior of RTT components among vantage points. That is, we cannot assume the existence of a one-fits-all model for network KPIs, even within the same network. Moreover, our results show that complex models with outstanding performance in some situations can fail where simpler ones achieve good results. Additionally, this analysis shows that RTT components (i.e., ΔRTT) locate and differentiate how traffic is affected when traversing each of the vantage points. This fact is useful to detect situations of local saturation in a network segment that are not detectable with the aggregated RTT.

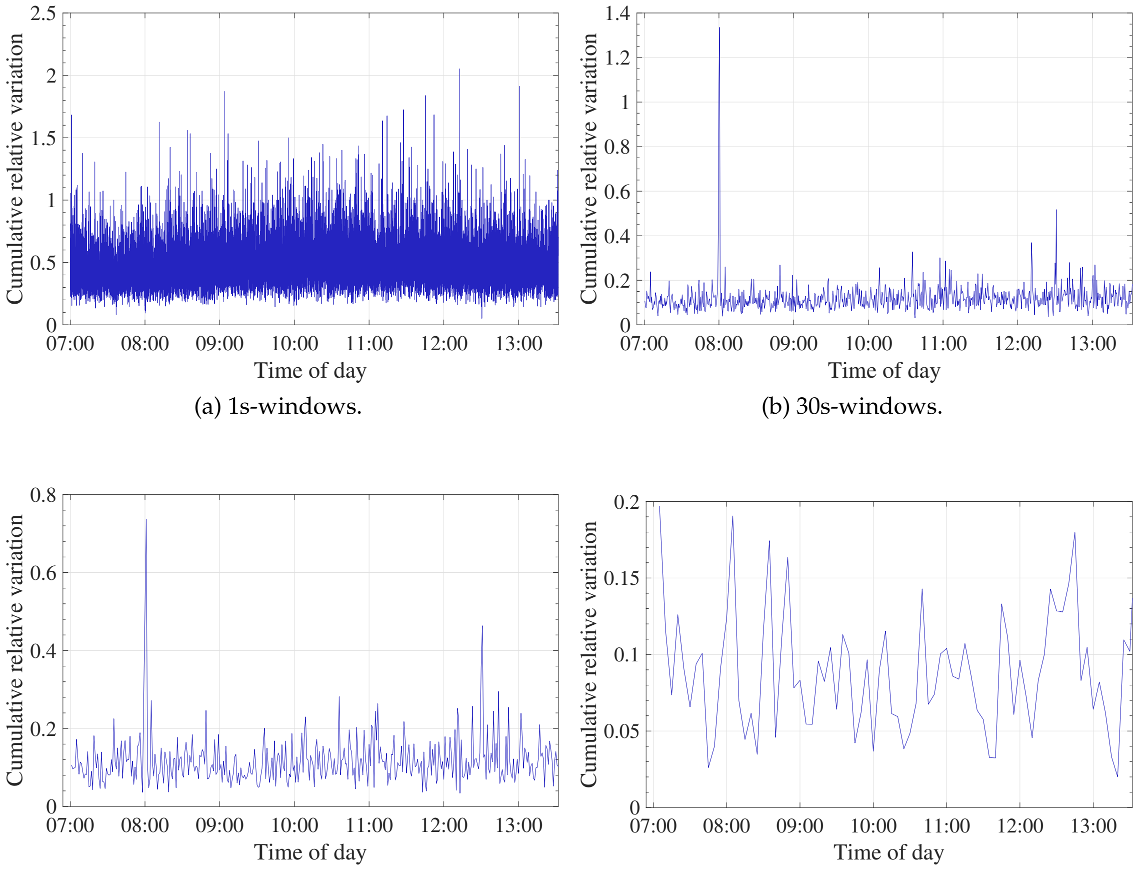
3.5.2 Long-term device monitoring

Once the short data ranges' modeling has provided good results for the characterization of real measurements, we move forward to the evaluation of its outcomes in longer periods. To do so, we considered the ΔRTT extracted from *Dataset*$_2$, which lasts for several hours and exhibits severe extreme ΔRTT values as a result of the firewall operation.

In this case, the situation is completely different due to the bursty nature of the single-flow estimates. Figure 3.5 shows this situation with the Complementary Cumulative Distribution Function (CCDF) of ΔRTT for all the flows in the trace, showing that the aforementioned models cannot fit either central or extreme values. In this scenario, projections within different time windows—1s, 30s, 60s, 300s—can reduce the variance in centrality measures caused by isolated extreme values.

As stated before, Dictyogram enables the definition of quantitative metrics to evaluate ECDFs' stability along time and determine whether a window size may be suitable. Figure 3.6 shows the cumulative relative variation of Dictyogram for the aforementioned window sizes, by applying (3.9). Remarkably, the lower the window size is, the projection effect will become less noticeable—i.e. models of projected observations can be expected to be similar to the single-flow ones.

Hence, robust fitting of central delay values will require larger windows sizes although either excessively coarse or fine-grained pre-filtering can obfuscate significant



(a) 1s-windows.

(b) 30s-windows.

(c) 60s-windows.

(d) 300s-windows.

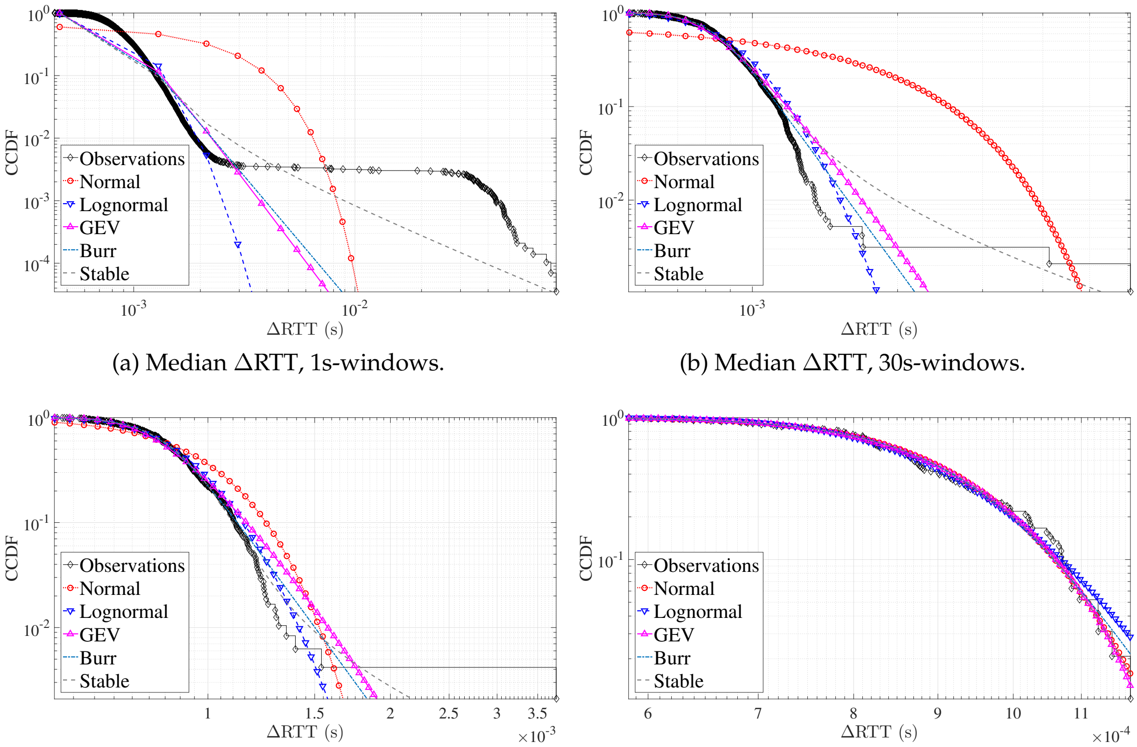
Figure 3.6: Dictyogram cumulative relative variation in *Dataset*$_2$. Effect of window size for computation of flows per decile.

punctual deviations—see the peaks at 8:00 and 12:30 in Figures 3.6b and 3.6c, which are undetectable in Figures 3.6a and 3.6d.

These situations clearly translate into different fitted models after projection, as shown in Figure 3.7. This figure illustrates that the coarser the projection is, the better the model fits: both Figures 3.7a and 3.7b present cases where few extreme observations impoverish the fitting, whereas Figures 3.7c and 3.7d display models that fairly fit the data up to 99th percentile.

On the other hand, extreme values' modeling can be tackled using other order statistics instead of the median. This is useful to represent boundaries for network equipment performance improving the detection of service disruptions. Hereinafter, we consider the 95th percentile for illustrative purposes and aiming at the discrimination of the large ΔRTT peaks in our data. Time-based pre-filtering with the later statistical modeling allows the model to capture extreme values with a reduction of over- or under-represented atypical observations. The effect of window size in this procedure is illustrated in Figure 3.8, which shows the convergence to a stable situation with an acceptable fitting of the extreme values and a progressive reduction of the weight of observations near central values.

With this, adPRISMA corroborated its capabilities to reach a comprehensive yet simple description of how network elements behave. Remarkably, this description distinguishes the dynamics of central and extreme values and includes quantitative criteria to balance



(a) Median ΔRTT, 1s-windows.

(b) Median ΔRTT, 30s-windows.

(c) Median ΔRTT, 60s-windows.

(d) Median ΔRTT, 300s-windows.

Figure 3.7: Results for ΔRTT in *Dataset*$_2$. Time-based aggregation, median projection, with diverse window sizes.

variance—i.e., reducing the effect of bursty measurements—and bias—i.e., considering the ECDFs' variability during the projection stage.

3.6 Discussion

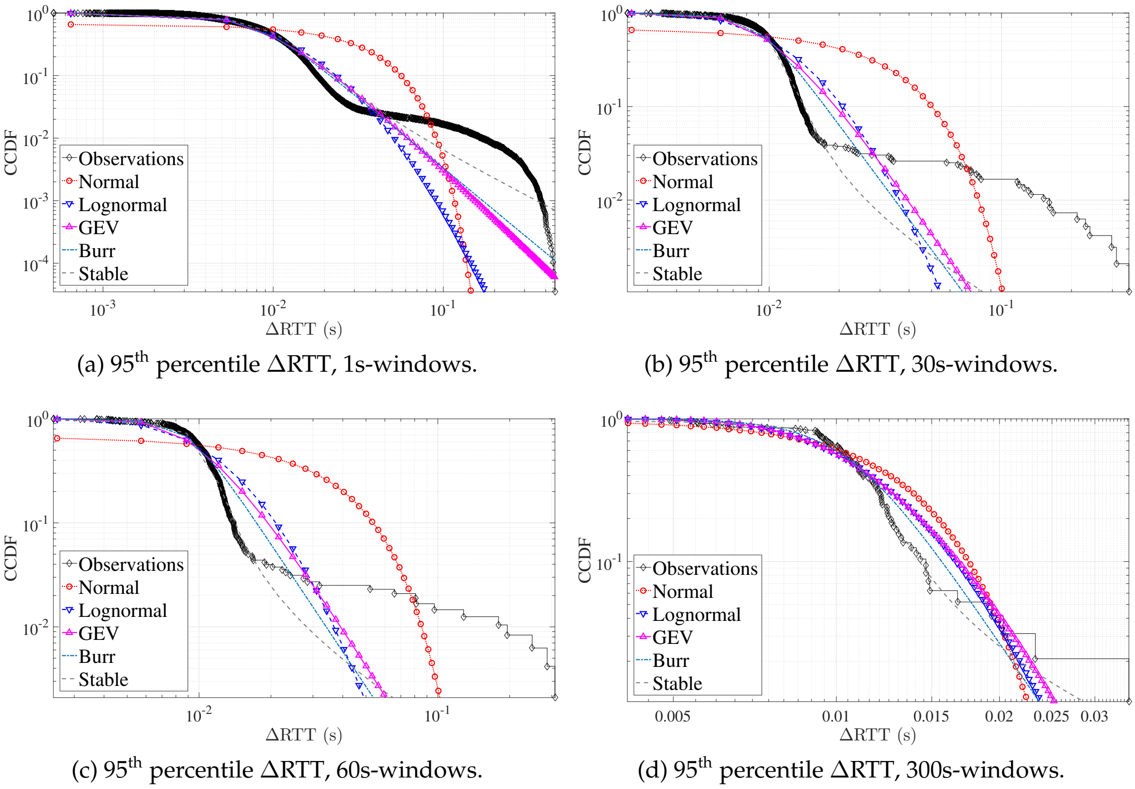
The evaluation of adPRISMA has illustrated the viability of monitoring systems with the desirable characteristics that grounded this work. Our proof of concept and case studies have exposed some remarkable ideas that improve current network management state of the art:

1. *Passive retrieval of relevant information can be distributed:* adPRISMA implements a distributed data gathering strategy, which is useful to improve the scalability of monitoring systems. Then, data aggregation and processing provided meaningful contextual information to characterize the network state comprehensively.

2. *RTT components help to locate where performance issues are most likely to appear:* as shown above, the observations of RTT do not fully characterize the behavior of RTT components. Therefore, the application of strategies such as ours can improve the detection and actuation in case of network issues.

3. *Models that are more complex are not necessarily better:* our evaluation and first case study reveal that simpler models may be better to represent measurements if complexity is included in the selection criteria. That is, slight improvements of goodness of fit may not justify the usage of more sophisticated models.

4. *Projection of flow-based estimates can improve the extraction of node-level KPIs:* second case study presented adPRISMA outcomes when analyzing data lasting for several



(a) 95th percentile ΔRTT, 1s-windows.

(b) 95th percentile ΔRTT, 30s-windows.

(c) 95th percentile ΔRTT, 60s-windows.

(d) 95th percentile ΔRTT, 300s-windows.

Figure 3.8: Results for ΔRTT in *Dataset*$_2$. Time-based aggregation, 95th percentile projection, with diverse window sizes.

hours and in the presence of large peaks in the RTT component under test. This has exposed a moral: since single-flow estimates can produce sub-optimal models with high variance, we need techniques that reduce the variance and allow us the characterization of the regular behavior in the vantage point correctly.

However, some practical issues may arise during the operation of adPRISMA. For instance, random packet sampling in vantage points may harm the fitting of models because of the reduction of mutual observations.

3.7 Conclusion

We have described adPRISMA, a network monitoring system able to provide comprehensive multi-point RTT modeling. It relies on the decomposition of passive RTT values in components that reflect the state of different network segments. adPRISMA is equipped with an automatic model selection algorithm that takes into account goodness of fit and complexity to optimize computational cost of analysis. This fitting also includes projection methods to improve the extraction of KPI trends from single-flow estimates.

Although experimental results have focused on RTT measurements, our methodology can be extended to other performance indicators measured at multiple points—e.g. delay variation or jitter at each vantage point. Specifically, adPRISMA provides a processing engine with a general set of features for measurements: namely, (a) pre-process, correlate and cluster the measurements, (b) segment observations using time or spatial location, and (c) fit models and choose the most suitable one depending on the situation and trade-offs between accuracy and complexity. Furthermore, adPRISMA's design and operation make easier the definition of wide monitoring perspectives, as observations from different vantage points can be simultaneously considered and correlated.

These features turn adPRISMA into a promising framework to enrich network management platforms and tools, given its advantages for the characterization of network KPIs with high adaptability. For instance, high values can be distinguished from atypical values—as seen in the first case study—, and projections with different window sizes can be used—as shown in the second case study—which may be helpful to improve bias-variance trade-offs. Remarkably, both use cases were intended to illustrate how these insights can improve and support the business logic inherent to many management tasks. Hereby, we believe that our work provides evidences of adPRISMA applicability to the monitoring, analysis and modeling of diverse network KPIs.

In sum, the experimental assessment of our proof of concept exposed that it provides promising results both in synthetic scenarios and in field trials with real-world traces gathered from enterprise networks. Additionally, we have released a prototype that is freely available to the community.[1]

[1] `https://github.com/hpcn-uam/adprisma`

4 Time-aware modeling: time series characterization of central behaviors

4.1 Introduction

Network characterization is an important task for alerting and provisioning. However, in strongly time-dependent network KPIs is not immediate to provide a suitable model. In this light, we aim at the modeling of real-world network time series, where several phenomena such as multiple modes or extreme behaviors are frequent, and we want to extract the most from this data.

Nowadays, most of the monitoring systems rely on simple baselines (moving average or moving median with thresholds based on standard deviation) [Vega et al., 2018] that work really well if curves are similar. Consequently, one of the jobs of a network operator is to analyze the different daily behaviors and try to group similar days, such as grouping by the day of the week. Nevertheless, we found out that, in many cases, patterns are really obscured or even not periodical but random instead. This entails our next objective: classify daily trends so that we can still use previous baselines in a more reliable way reducing the cost of manual operators and make it scalable.

In contrast to previous section, the model here is based on stochastic processes and not random variables. Consequently, the natural approach would be extending the previous techniques for the functional case. Nevertheless, that is not always feasible and complex models in this scope will tend to have infinitely many parameters and, indeed, even simple models can present non-trivial issues. In conclusion, we define

$$X_t = \sum_{i=1}^{d} \theta_i X_t^{(i)}, \tag{4.1}$$

as the model for our data, where d is the number fo components, θ_i are some random coefficients that satisfy

1. $\mathrm{supp}(\theta_i) = \{0, 1\}$
2. $\sum_{i=1}^{d} \theta = 1$, i.e. only one of the θ_i can be activated at the same time.

The $X_t^{(i)}$ are stochastic processes defined as

$$X_t^{(i)} = \mu_t^{(i)} + \varepsilon_t^{(i)}, \tag{4.2}$$

where $\mu_t^{(i)} \in L^2([0,T])$ are some deterministic functions and $\varepsilon_t^{(i)}$ are heteroscedastic error terms with $E[\varepsilon_t^{(i)}] = 0$. The objective is to estimate all the parameters: the number of components d, the functions $\mu_t^{(i)}$ and to provide a bound or a soft characterizing $\varepsilon_t^{(i)}$.

From the model, it is simple to foresee the approach to this problem, we will be using unsupervised learning methods to classify the functions into artificial categories, usually called clusters. We will cover several methodologies for the same purpose, commenting on differences, advantages and disadvantages. These methodologies are functional k-means, principal components, functional principal components and autoencoders, where the first one is a direct approach and the rest are all based on projection methods.

4.2 State of the art

Few works have addressed the use of FDA applied to network related data. In [Muelas et al., 2017], authors propose an architecture for network data processing using functional data and evaluate some application cases such as clustering, outlier detection and capacity planning. Furthermore, authors in [Ben Slimen et al., 2017] apply functional data techniques over a set of KPIs to predict potential performance problems in radio cells. Authors in [Ben Slimen et al., 2018] explored co-clustering methods for functional data by an adaption of the expectation-maximization algorithm to functional data, showing the application of co-clustering to mobile networks. Although these works point out some interesting applications of the functional data analysis as well as some fundamentals, they lack the service-oriented approach, the combination with machine learning and the functional multi-modal approach.

On the other hand, in recent years several works have used machine learning and neural network techniques applied to network data. Authors in [Boutaba et al., 2018], present a survey on the application of machine learning techniques to different network areas, such as traffic prediction and classification or network security and routing.

Other approaches, like [Jalalpour et al., 2019], present a traffic monitoring analytics system. Such system uses clustering techniques and autoencoders over flow features extracted from incoming traffic to detect attacks and anomalies with a 76% recall. Similarly, in [Nguyen et al., 2019], authors apply variational autoencoders over NetFlow data to detect and cluster network anomalies. Other works like [Kiran et al., 2020] use autoencoders, isolation forest and PCA techniques to detect anomalous packets in TCP data transfers by means of clustering.

The above works provide valuable insight into the application of machine learning techniques over network data, but they are focused on classifying either individual packets or flow records rather than classifying network aggregates using time series information. These approaches are suitable for anomalies or attacks detection, where some previous knowledge is available, but not for service or behavioral characterization. In this work, we present clustering techniques applied to time series of network aggregates, not suitable for fine-grained network monitoring, but more useful to network provisioning and performance assessment.

Recently, Hidden Markov Models (HMM) and Bayesian networks have also received much attention in network monitoring. Authors in [Chen et al., 2016] present a HMM to predict traffic volume in terms of flow counts using an auto regressive approach.

In [Mouchet et al., 2020], a system is presented to segment time series of network delay using an Infinite HMM. Although the latest approach is helpful to detect different states of the network, they lose the time component, so it is not possible to know when the network will change from one state to other. Our aim here is different: we classify the whole daily time series and not time sub-intervals, since we are interested in characterizing frequent trends of the network operation without losing the time component.

Focusing on service characterization, authors in [Samani and Stadler, 2018] model key service metrics using infrastructure measurements in a cloud environment by means of mixture density neural network. This type of network provides as output model the parameters for a mixture of Gaussian distributions. Using such a model, they can predict SLA conformance. Although this work is somehow similar to our proposal, it focuses mainly in the prediction of conformance using mean values estimated from a distribution. In our case, we provide a model based on a mixture of stochastic processes (not necessary Gaussian processes), whereas they present a Gaussian mixture model for each time, leading to having precise models in each time moment, but not being able to see time-dependent behaviors such as correlation between events.

Finally, while all previous works addressed the classification or characterization problem by using either functional or machine learning approaches separately, regarding the joint use of functional data and neural networks there is little literature. For instance, in [Rossi et al., 2005], a method for using functional data as input for neural networks is presented. In this work, several methods for functional processing such as FPCA or projection on smooth bases are presented. A similar approach is presented in [Rossi and Conan-Guez, 2005], focusing on multi-layer perceptrons. While these works present some interesting foundational ideas that are used in this paper, they rely on simple neural networks that are far from modern deep neural network models. Moreover, there are some contributions of the neural networks to the functional data analysis, more explicitly radial basis function neural networks [Broomhead and Lowe, 1988] exposed a kernel method that it is inherently based on a functional data representation of the data. Nevertheless, its application to networking has been limited [Baras et al., 1997] and it is restricted to this kind of basis representation, where we propose a more general idea of using any functional orthonormal basis representation.

4.3 Functional k-Means

The first approach is to directly solve the problem using a clustering method in the functional setting. Among the many available clustering methods in the state of the art, we chose k-Means [Hartigan and Wong, 1979] because it is one of the simplest methods and it does not rely on distributional assumptions as, for instance, Expectation-Maximization (EM) algorithm does.

The k-means algorithm is described in Algorithm 4.1. It has two main parts:

1. **Initialization:** we need a set of initial centroids to start working. They are normally chosen randomly but the convergence of the algorithm is strongly affected by bad choices of them. In this light, alternative methods as k-Means++ initialization [Arthur and Vassilvitskii, 2007] arise.

   This improvement is as simple as choosing centroids that are far from each other, this is, we choose at random one and the subsequent ones are chosen randomly with weights inversely proportional to the mean distance to other centroids. Another possibility is to perform a random search of the seed, so that the random

Algorithm 4.1 k-Means algorithm: main body

```
function K-MEANS(K, X)
    # 1. Initialization
    centroids ← initialize(X,K)
    # 2. Main loop
    for i in 1:ITER_LIMIT do
        # 2.1 Compute the classes
        for j, x in X do
            classes[j] ← closest(x, centroids)
        end for
        # 2.2 Compute the new centroids
        for k in 1:K do
            centroid ← deepest(X[classes == k])
        end for
    end for
    return classes, centroids
end function
```

seed does not affect the algorithm. Both initializations procedures are described in Algorithm 4.2, in particular, classic initialization is function INITIALIZE1 and k-means++ initialization is INITIALIZE2.

Algorithm 4.2 k-Means algorithm: initializations

```
# Classic initialization
function INITIALIZE1(X, K)
    return random_choice(X,K)
end function
# K-means++ initialization
function INITIALIZE2(X, K)
    centroids ← [random_choice(X,1)]
    for k in 2:K do
        distances ← min(distance(X, centroids), axis=1)
        centroids.append(random.choice(X,1, weights=distances)
    end for
    return centroids
end function
```

2. Main loop: it is divided in two steps. First, the classes are computed by choosing the same class as the closest centroid, i.e. the closest centroid is the representative of the class.

   Second, we update the centroids (2.2). For that purpose, we use a functional depth measure [Cuevas et al., 2007]. These depth measurements generalize the concept of median and quantiles [Gijbels and Nagy, 2017, Nieto-Reyes and Battey, 2016]. A functional depth measure $D(f, P)$ indicates how "deep" the function f is in terms of the distribution of the data, P. In our case, we consider the following half-region depth measure [Zuo and Serfling, 2000, López-Pintado and Romo, 2009, López-Pintado and Romo, 2011]. Given that P is estimated by the samples $\{f_i\}_{i=1}^{N}$, we define our depth measure

$$D(f, P) = \min(SL_P(f), IL_P(f)), \tag{4.3}$$

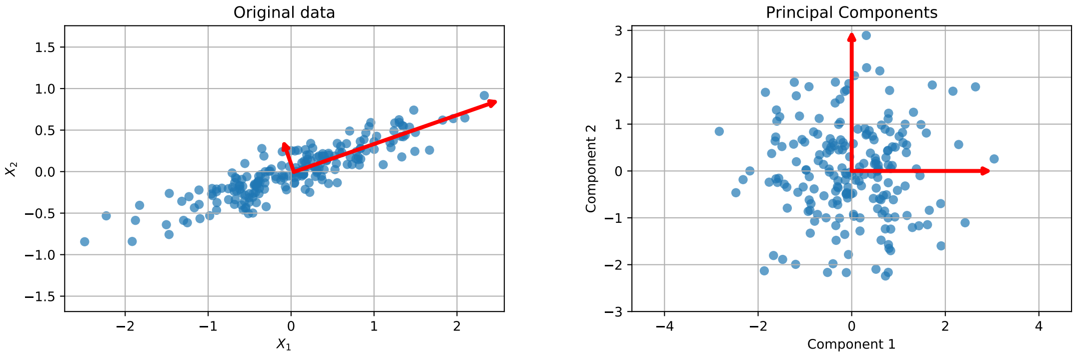


Figure 4.1: Principal component analysis in $\mathbb{R}^2$ for a multivariate Gaussian distribution.

where

$$SL_P(f) = \underset{i=1,\dots,n}{\text{mean}} \frac{1}{T}\int_0^T \mathbb{1}_{\{t\in[0,T];\ f(t)\le f_i(t)\}}(t)dt \tag{4.4}$$

and

$$IL_P(f) = \underset{i=1,\dots,n}{\text{mean}} \frac{1}{T}\int_0^T \mathbb{1}_{\{t\in[0,T];\ f(t)\ge f_i(t)\}}(t)dt, \tag{4.5}$$

being $\mathbb{1}_A(t)$ the indicator function of set A. SL and IL measure the proportion of time that f is below the samples or above the samples.

In order to improve cases where noise was dominant, we add artificial curves to the sample before computing the depth measure. In particular, we add the median function, i.e.

$$\text{median}(\{f_i\}_{i=1}^N) = \{\text{median} f(x_j)\}_{j=1}^d,$$

which is just the median at the grid points. This curve normally obtains the highest depth measurement as the median is resilient against the noise of the signals, but it can lie outside the distribution. Since our model separates the noise ($\varepsilon_t^{(i)}$) and the trend ($\mu_t^{(i)}$), we are still interested in this approach since we expect that the median resembles $\mu_t^{(i)}$ better.

To conclude, we mentioned before that k-Means algorithm can be strongly affected by a wrong choice of the initial centroids, compromising convergence of the algorithm. To deal with this issue, we can just run the algorithm several times with different seeds that lead to different initializations and choose always the one that scores better.

4.4 Principal Component Analysis

PCA is one of the most powerful and used techniques for dimensionality reduction. It is based on searching for the directions where the data has more variance, i.e. more information. Concreting, the v_k is the kth principal component of random vector X if and only if

$$v_k = \underset{\substack{\alpha\perp v_j;\ j<k \\ \|\alpha\|=1}}{\arg\max}\ \text{Var}(X\cdot\alpha). \tag{4.6}$$

Figure 4.1 shows an example of PCA directions. In the left-hand side, we see the original space that has a dominant direction that contains almost all the information of the sample and its orthogonal that contains the rest. In the right-hand side, the transformed space shows that data is now equally distributed in both directions.

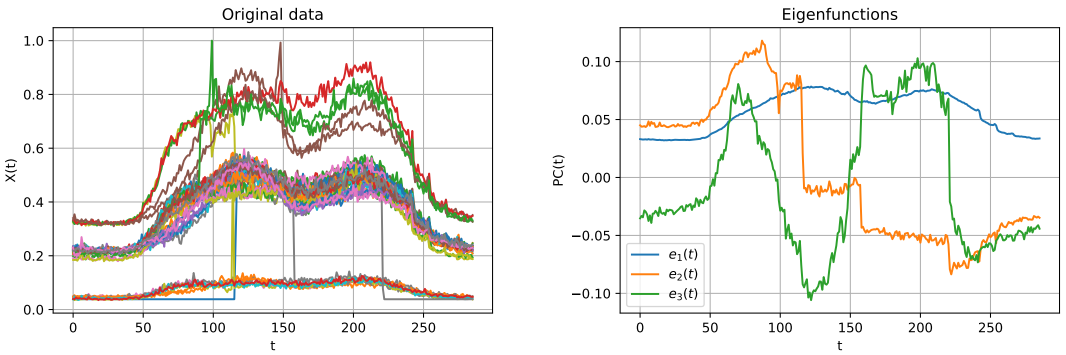


Figure 4.2: Original data and the estimated eigenfunctions whose eigenvalues are the largest ones in magnitude.

Nevertheless, this technique is commonly used with high-dimensional data. Now, let X be our data matrix of size $N \times d$ and let $\overline{X}$ be the centered version, $\overline{X} = X - \frac{1}{N}\sum_{i=1}^{N} X^{(i)}$. It is easy to that $\overline{X}'\overline{X}$ is proportional to the empirical covariance matrix.

We need to compute the eigenvalues $\{\lambda_k\}_{k=1}^{d}$ and eigenvectors $\{w_k\}_{k=1}^{d}$ of $\overline{X}'\overline{X}$. These eigenvalues, order by magnitude, represent the amount of variance explained by each direction, and the eigenvector is the direction itself.

To obtain the projected version, we just need to use the directions, i.e. multiply the data and the direction:

$$Y = \overline{X}[w_1; \dots; w_K] = [\overline{X}w_1; \dots; \overline{X}w_K]. \tag{4.7}$$

It is clear that if $K \geq d$, the projection is just a change of basis that makes the covariance matrix of X the identity. If $K < d$, then Y is a compression of the original data where the amount of information is normally measured in terms of the explained variance (i.e. the partial sum of the eigenvalues).

4.5 Functional Principal Component Analysis

Once PCA is clear, it is simple to explain the FPCA. In this case, X will not be a matrix or a random vector but a stochastic process, which means that samples will be functions. As before, we want to maximize the variation, but directions in this case should be understood as directions in $L^2([0,T])$, this means that $v_k(t)$ is a principal component if and only if

$$v_k(t) = \underset{\substack{\alpha\perp v_j;\, j<k \\ \|\alpha\|=1}}{\arg\max} \operatorname{Var}(\langle X, \alpha\rangle) = \underset{\substack{\alpha\perp v_j \\ j<k \\ \|\alpha\|=1}}{\arg\max} \operatorname{Var}\left(\int_0^T \alpha(t)X(t)dt\right), \tag{4.8}$$

where $\langle\cdot,\cdot\rangle$ denotes the scalar product and $\alpha \in L^2([0,T])$. Although approach is quite similar, computation is not as direct as before.

First, we call $\overline{X}$ the centered version of the process, $\overline{X} = X - \mathbb{E}[X]$ and $k(t,s)$ the covariance operator, $k(t,s) = \mathbb{E}\left[\overline{X}(t)\overline{X}(s)\right]$. Thanks to Mercer's theorem (Theorem 2.2.1), this operator can be decomposed as follows

$$k(t,s) = \sum_{j=1}^{\infty} \lambda_j e_j(t) e_j(s), \tag{4.9}$$

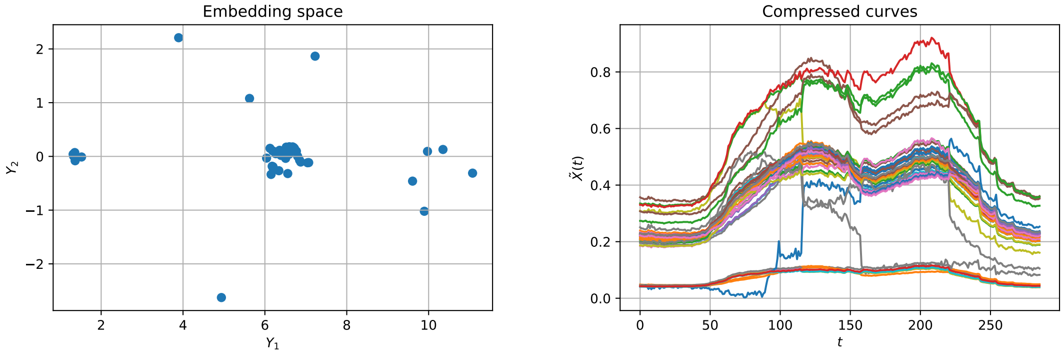


Figure 4.3: Results of FPCA. Left-hand side shows the embedding space of the coefficients and right-hand side the compressed version with three principal components

where $\{\lambda_j\}_{j=1}^{\infty}$ play the same role of the eigenvalues and $\{e_j\}_{j=1}^{\infty}$ are the eigenfunctions, which are akin to the eigenvectors but in the functional setting. Figure 4.2 shows some of the empirical data we use in this chapter and the three principal components whose eigenvalue has the largest magnitude.

Once we have this decomposition, the subsequent steps are analogous to the previous case. We obtain the projection by just using scalar products:

$$Y = [\langle\overline{X}, e_1\rangle, \ldots, \langle\overline{X}, e_K\rangle]. \tag{4.10}$$

In this case, the dimension of the data, unless unusual cases where almost all the eigenvalues are zero, is always infinity. In particular, Karhunen-Loeve theorem (Theorem 2.2.2) decomposes X as the sum

$$X = \sum_{i=1}^{\infty} Y_i e_i(t), \tag{4.11}$$

where Y_i is the ith component of the Y vector and $e_i(t)$ the eigenfunction. If we sort the eigenfunctions by the magnitude of the eigenvalue, then a partial sum of the previous series provides a compressed version of X, i.e.

$$\tilde{X} = \sum_{i=1}^{K} Y_i e_i(t). \tag{4.12}$$

Figure 4.3 displays Y, belonging to the embedding space $\mathbb{R}^2$ and also the compressed version $\tilde{X}$ that can be reconstructed from $\{Y\}_{i=1}^{3}$.

An useful way of understanding PCA and FPCA is also thinking of it as another minimization problem, in particular,

$$\min_{\{\alpha_i\}_{i=1}^{K};\{y_i\}_{i=1}^{K}} \mathbb{E}\left(\left\|X - \sum_{i=1}^{K} y_i \alpha_i\right\|^2\right), \tag{4.13}$$

where the norm $\|\cdot\|$ is either the l^2 norm of $\mathbb{R}^d$ (PCA) or the $L^2([0,T])$ norm (FPCA) and α_i are either the eigenvectors (PCA) or the eigenfunctions (FPCA).

Once we have computed Y, either with PCA or FPCA, the next step is the same, perform a cluster algorithm in Y. This approach based on projections has the advantage

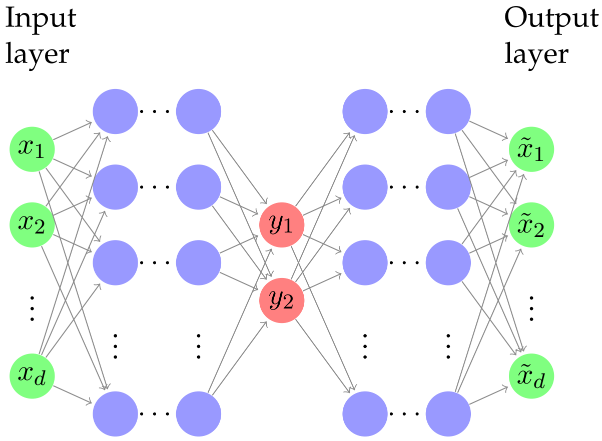


Figure 4.4: Architecture of a sequential autoencoder. Green nodes represent the original data X and its compressed version $\tilde{X}$, red nodes the embedding output $Y \in \mathbb{R}^2$ and blue nodes intermediate layers.

of using the projection for compression or anomaly detection, but the embedding can be sometimes misleading. In this particular case, it is quite intuitive since PCA and FPCA are linear transformations, but we will see next that non-linear techniques produce completely different embeddings.

4.6 Autoencoders

About non-linear techniques, Autoencoderss (AEs) are a common technique based on neural networks that aims at building an embedding space with non-linear transformations. This is done by training a neural network where inputs and outputs are the same and placing a bottleneck, a layer with a number K of neurons, in the middle. Figure 4.4 shows the architecture of a sequential autoencoder, generally called just autoencoder.

Some remarks should be made before seeing its properties. The left-hand part of the network from X to Y will be called the encoder and usually referred as $f_\theta : \mathbb{R}^d \to \mathbb{R}^K$, where θ are the parameters of the network. On the other hand, the right-hand side from Y to $\tilde{X}$ is called the decoder, $f_\phi : \mathbb{R}^K \to \mathbb{R}^d$. Basically, the training of the network can be seen as a minimization procedure, i.e.

$$\theta, \phi = \arg\min_{w_1, w_2} \operatorname*{mean}_i \text{Loss}(X^{(i)}, f_{w_2}(f_{w_1}(X^{(i)}))), \tag{4.14}$$

where, as loss function, we will use the MSE, $\text{Loss}(X, Z) = \frac{1}{N}\sum_{j=1}^{d}(X_j - Z_j)^2 = \|X - Z\|_2$.

Seeing the AE as a minimization problem allows us to expose a relation between PCA and AE. Assume we have a neural network with only one hidden layer of size K and linear activation functions, so in this case, the encoder is

$$f_\theta(X) = Y = [X \cdot \theta_1, \ldots, X \cdot \theta_K] \tag{4.15}$$

and the decoder is

$$f_\phi(Y) = \tilde{X} = \sum_{i=1}^{K} Y_i \phi_i, \tag{4.16}$$

where θ_i and ϕ_i are vectors in $\mathbb{R}^d$. Focusing on the decoder, it is easy to see that expression of the MSE is similar to PCA,

$$\text{Loss}(X, f_\phi(Y)) = \left\| X - \sum_{i=1}^{K} Y_i \phi_i \right\|_2^2 \tag{4.17}$$

and the cost function is

$$\underset{i}{\text{mean}}\, \text{Loss}(X^{(i)}, f_\phi(Y)) \approx \mathbb{E}\left[\text{Loss}(X^{(i)}, f_\phi(Y))\right] = \mathbb{E}\left[\left\| X - \sum_{i=1}^{K} Y_i \phi_i \right\|_2^2\right]. \tag{4.18}$$

Therefore, as long as $\text{Im}(f_\theta) = \mathbb{R}^k$ (i.e. independent variables in data are at least K), we proved the equivalence between minimization problem (4.14) and

$$Y, \{\phi_i\}_{i=1}^d = \underset{Y \in \mathbb{R}^k, \phi_i \in \mathbb{R}^d}{\arg\min}\; \mathbb{E}\left[\left\| X - \sum_{i=1}^{K} Y_i \phi_i \right\|_2^2\right]. \tag{4.19}$$

So, the simplest possible AE is equivalent to PCA in terms of minimization problem. Bear in mind that PCA ensures Y is composed of variables that are not correlated, where in this case we have not this property and some further orthonormalization procedure should be conducted.

The next reasonable question is, is FPCA also equivalent to some AE? To answer this, we define a brand-new concept, functional neural network. We will call a functional neural network a neural network whose inputs are coefficients of an orthonormal functional basis. Consequently, an AE with functional inputs will be called a Functional AE (FAE). In some basis representation as Fourier, it is automatically achieved the orthonormality but in others, such as finite elements, it is not. For non-orthonormal representations, we recall Gramm-Schmidt orthonormalization procedure in equation (4.20).

$$v_i = \begin{cases} u_i & \text{if } i = 1 \\ u_i - \sum_{j=1}^{i} \frac{\langle u_i, v_j \rangle}{\|v_j\|^2} v_j & \text{otherwise} \end{cases} \tag{4.20}$$

The orthonormalization base allows us to state an important property: the minimization of the MSE of the coefficients is equivalent to minimize the $L^2([0, T])$ norm. This is due to Plancherel's theorem. It states that if we have an orthonormal basis $\{v_i\}_{i=1}^{\infty}$ and $X = \sum_{i=1}^{\infty} \alpha_i v_i$, then the norm can be expressed as:

$$\|X\|^2 = \sum_{i=1}^{\infty} \alpha_i^2, \tag{4.21}$$

being then a generalization of the Pythagorean theorem for more general spaces.

Analogously to PCA, we consider again a single hidden layer of size K. We will assume that the basis used is one so that data is represented with no loss. In some general case where X is some stochastic process, this is not possible, but given that data is sampled at some finite frequency, we cannot have infinite dimension empirically. Thus, it is easy to see that the decoder can be seen as $\tilde{f}_\phi : \mathbb{R}^2 \to L^2([0, T])$

$$\tilde{f}_\phi(Y) = \sum_{i=1}^{d} \langle \phi_i, Y \rangle b_i, \tag{4.22}$$

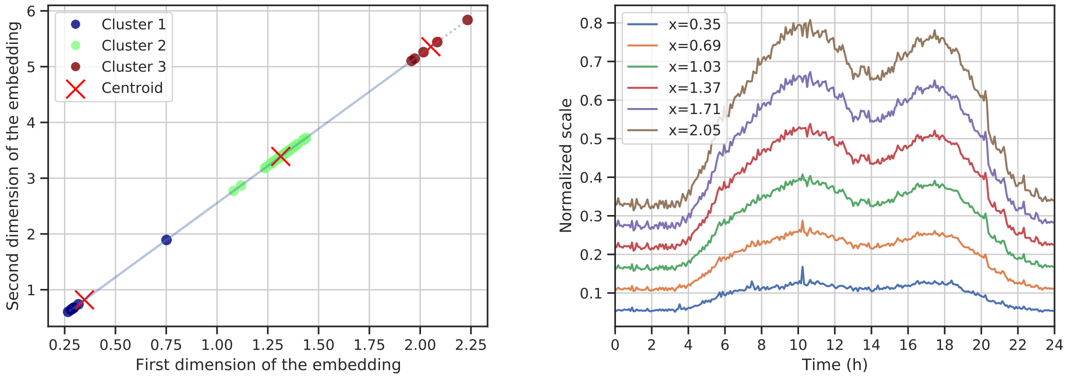


Figure 4.5: Autoencoder embedding. Curves 1, 4 and 6 in right-hand side of the figure are the centroids in the original space, indicated with a red cross in left-hand side.

where $\{b_i\}_{i=1}^{d}$ is the functional basis. As long as $d > K$, we can see also the minimization problem as

$$\begin{aligned}\min_{\phi} \mathbb{E}\left[\left\|X - \sum_{i=1}^{d} \langle \phi_i, Y\rangle b_i\right\|^2\right] &= \min_{\phi} \mathbb{E}\left[\left\|\sum_{i=1}^{d} (X_i b_i - \langle \phi_i, Y\rangle b_i)\right\|^2\right] \\ &= \min_{\lambda_i} \mathbb{E}\left[\left\|\sum_{i=1}^{d} (X_i - \lambda_i)\, b_i\right\|^2\right] \\ &= \min_{\lambda_i} \mathbb{E}\left[\sum_{i=1}^{d} (X_i - \lambda_i)^2\right],\end{aligned}$$

which as mentioning before is minimizing the MSE of the components.

Figure 4.5 shows the embedding space of an autoencoder. In this case, the non-linear decoder hides a geometrical property. As it can be seen in the embedding, there exists a linear relation, i.e. the true dimension of the embedding is 1. So, if we say that this linear relation is $y = ax + b$ with $x \in \mathbb{R}$ and the decoder is f_ϕ, we have that

$$\Phi(x) = f_\phi(ax + b), \tag{4.23}$$

defines a parametrization of a manifold in $L^2([0,T]$, displayed in right-hand side of Figure 4.5. This means that the dimension of the data is, in fact, lower than expected since the structure of the curves lives mostly in this 1-dimensional space.

4.7 Choosing the most appropriate *K*

A question that remains unanswered is which is the most suitable number of components. To answer this, we use the silhouette coefficient (SC). This multivariate technique is entirely valid for any metric space, in particular, for functional spaces such as $L^2([0,T])$. In particular, the silhouette coefficient is defined as

$$SC_K = \text{mean}_{1\le j\le N} s_K(j), \tag{4.24}$$

where

$$s_K(j) = \frac{b_K(j) - a_K(j)}{\max\{a_K(j), b_K(j)\}}, \tag{4.25}$$

$$a_K(j) = \frac{1}{|C_j|-1} \sum_{i \in C_j; i \neq j} d(i,j) \tag{4.26}$$

and

$$b_K(j) = \min_{i \neq j} \frac{1}{|C_i|} \sum_{l \in C_i} d(l,j), \tag{4.27}$$

where $d(\cdot,\cdot)$ is the distance, in our case, the L^2 distance, C_i stands for the cluster which the curve i^{th} belongs to, and $|C_i|$ the number of curves in the cluster.

Analyzing these formulas, we see that a_K is the mean distance of all elements in the same cluster, so it is a measure of the compactness of the cluster and the lower it is, the better the performance is. On the other hand, b_K is a minimum of the mean distance to all the functions that belong to other clusters, i.e. a measure of how far it is from other clusters and, thus, the greater it is, the better the clustering obtained is.

Since $s_K = b_K - a_k$ with some additional normalization factor, we expect that higher values of this metric implies that associated clustering has compact clusters and clusters are well-spaced. s_K is a local metric that it is compute per sample, so we take the mean as a final metric. As shown before, it is clear that we want to maximize the silhouette coefficient and therefore we choose the K that maximizes it

$$\hat{K} = \arg\min_{K \in \mathbb{N}} SC_K. \tag{4.28}$$

It is also remarkable that silhouette coefficient needs to be a metric that is not biased by the number of components, i.e. it is not systematically better if we increase the number of components and it can be used for this purpose [Kaufman and Rousseeuw, 1990].

4.8 Results

First of all, we have to remark that it is not possible to use traditional metrics such as accuracy or recall if we do not have the true labels. Only artificial scenarios will have then these metrics available. So, we will cover first the metrics that we can use to evaluate a unsupervised problem.

4.8.1 Metrics

We will use two metrics, the silhouette coefficient already covered and whose formulation is in (4.24) and the Davies–Bouldin Index (DBI) [Davies and Bouldin, 1979].

As well as silhouette coefficient, DBI works for any metric space since it is wholly based on distances. So, in particular, it works in $L^2([0,T])$.

DBI is based on two concepts, the scatter of cluster C_i,

$$S_i = \operatorname*{mean}_{x \in C_i} d(x, \hat{\mu}_i), \tag{4.29}$$

and the separation of the clusters

$$M_{i,j} = d(\hat{\mu}_i, \hat{\mu}_j), \tag{4.30}$$

where $\hat{\mu}_i$ is the centroid of cluster C_i. The ratio between the two quantities above

$$R_{i,j} = \frac{S_i + S_j}{M_{i,j}}, \tag{4.31}$$

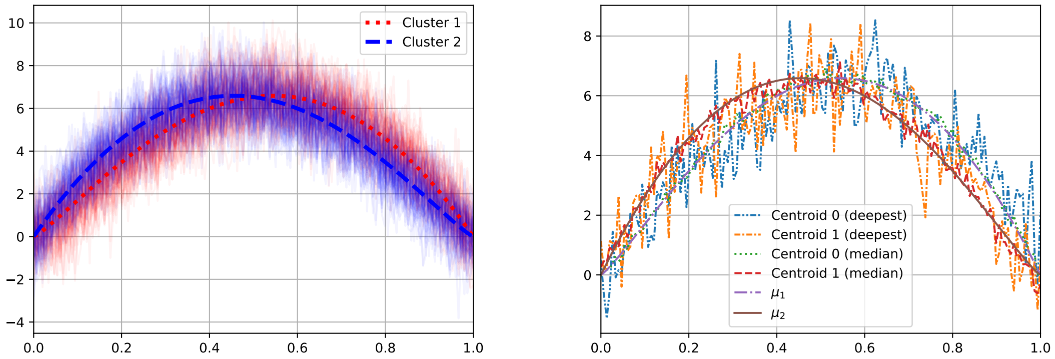


Figure 4.6: Centroids obtain with simulations

preserves some interesting properties such as being positive, being symmetric and improving in the following two situations: if two clusters are equally compact ($S_j = S_k$) but one is farther away from C_i than the other ($M_{i,j} \leq M_{i,k}$), then $R_{i,j} > R_{i,k}$ and the dual one, if two clusters are equally far from C_i ($M_{i,j} = M_{i,k}$), but one is more compact than the other ($S_j \leq S_k$), then $R_{i,j} < R_{i,k}$. To measure the performance of cluster C_i, the worst $R_{i,j}$ is chosen, i.e. we define

$$D_i = \max_{i \neq j} R_{i,j} \tag{4.32}$$

and the functional DBI as

$$DBI = \underset{i=\{1,\ldots,K\}}{\text{mean}} D_i. \tag{4.33}$$

Compared to silhouette coefficient, it relies heavily on the centroids to ensure the obtained clusters are tight and separate one from each other. Furthermore, it is computationally less expensive to compute, since it only iterates once over the whole set of time series. Nevertheless, the more clusters we have, the better this metric scores, so it is only comparable if we fix the same number of clusters for all the methods.

4.8.2 Simulations

For the simulations, we will reproduce the experiments in [Cuevas et al., 2007], where the authors introduced some synthetic situations to benchmark classification using depth measures. Bear in mind that this problem differs from theirs, since we are coping with unsupervised data.

We will cover two cases where the main difference is the number of elements in each class. In both cases, the model is assumed to be (4.2).

Case 1: balanced clusters

First, the mean functions are

$$\mu_t^{(1)} = 30(1-t)t^{1.2} \tag{4.34}$$

and

$$\mu_t^{(2)} = 30(1-t)^{1.2}t, \tag{4.35}$$

and the random noise functions are defined ε_t^1 and ε_t^2 as two independent Gaussian processes with zero mean and covariance $\text{Cov}(X(s), X(t)) = 0.2\text{exp}(-|s-t|/0.3)$. Also,

Table 4.1: Results of the experiments for case 1 and configurations of the parameters

	Case 1: balanced clusters			
Experiment	**Precision**	**Recall**	L^1 **distances**	L^2 **distances**
F. k-means N=60	0.93103	0.93103	0.80747, 0.85301	1.00959, 1.06130
with median	0.93548	1.00000	0.17028, 0.20957	0.21123, 0.26858
F. k-means N=120	0.98182	0.76056	0.83224, 0.74568	1.02664, 0.95606
with median	0.92727	1.00000	0.11272, 0.16516	0.14534, 0.20065
F. k-means N=360	0.85185	0.98773	0.78501, 0.76361	0.99911, 0.98696
with median	0.95238	0.97297	0.07627, 0.07682	0.09528, 0.09744
PCA N=60	0.96774	1.00000	0.13131, 0.16403	0.16821, 0.21379
PCA N=120	0.96923	0.92647	0.09033, 0.14816	0.11298, 0.17953
PCA N=360	0.95238	0.97826	0.05821, 0.06100	0.07280, 0.07664
FPCA N=60	0.96774	1.00000	0.15670, 0.23892	0.19767, 0.29530
FPCA N=120	0.96923	0.92647	0.11445, 0.15417	0.14161, 0.18603
FPCA N=360	0.95238	0.97826	0.06451, 0.05751	0.07843, 0.07613
AE N=60	0.63636	0.61765	0.42593, 0.38448	0.49871, 0.44875
AE N=120	0.93846	0.96825	0.16077, 0.23679	0.20234, 0.28080
AE N=360	0.94578	0.96319	0.09355, 0.12453	0.11849, 0.15043

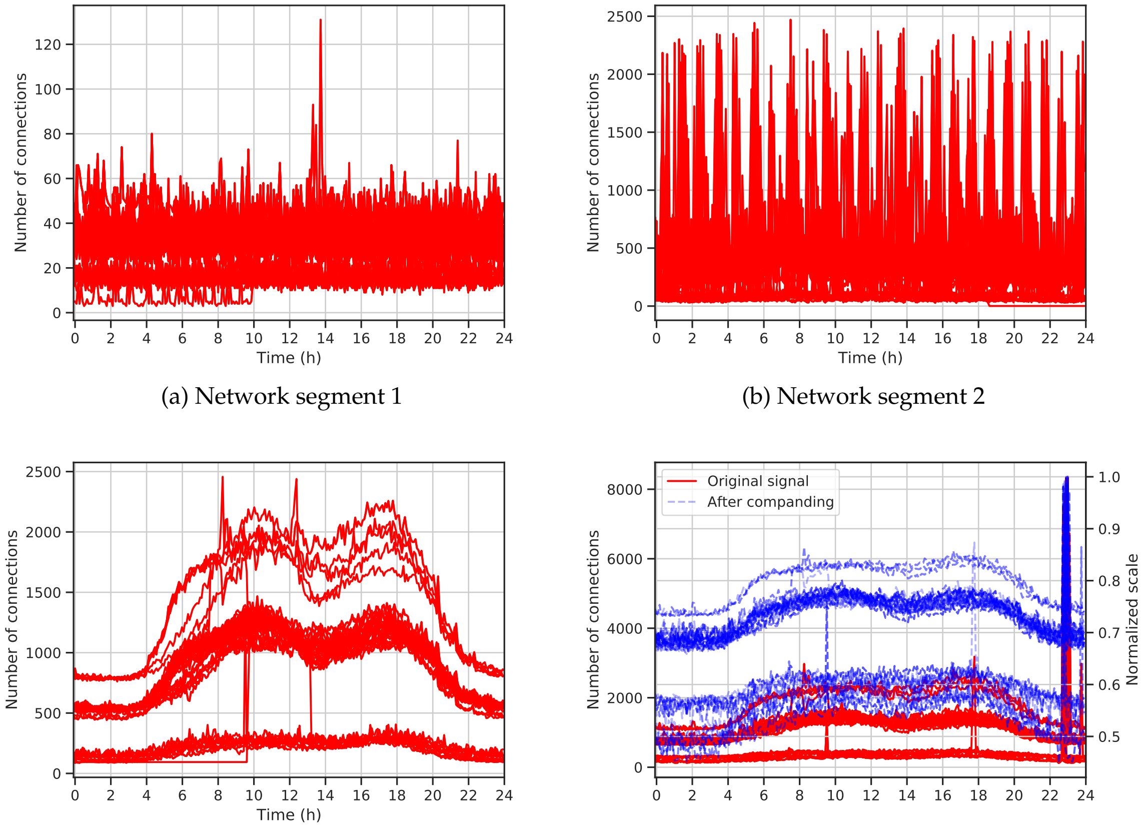
Table 4.2: Results of the experiments for case 2 and different configurations of the parameters

	Case 2: Imbalanced clusters			
Experiment	**Precision**	**Recall**	L^1 **distances**	L^2 **distances**
F. k-means N=60	0.87500	0.58824	0.80747, 0.75365	1.00959, 0.94715
with median	1.00000	0.74194	0.26790, 0.55455	0.32386, 0.64678
F. k-means N=120	0.95000	1.00000	0.78501, 0.74499	0.99911, 0.93045
with median	0.47000	0.90385	0.24270, 0.65084	0.29543, 0.74889
F. k-means N=360	0.98305	0.80556	0.78819, 0.76475	0.96087, 0.91427
with median	0.98305	0.87879	0.06515, 0.13854	0.08136, 0.17489
PCA N=60	1.00000	0.75000	0.25478, 0.56055	0.29804, 0.65403
PCA N=120	1.00000	0.95238	0.07703, 0.18788	0.09816, 0.23166
PCA N=360	0.96678	0.99658	0.05155, 0.09762	0.06422, 0.12366
FPCA N=60	0.53846	1.00000	0.23362, 0.53867	0.27395, 0.61573
FPCA N=120	1.00000	0.95238	0.07760, 0.27962	0.10012, 0.33064
FPCA N=360	0.96678	0.99658	0.08739, 0.12541	0.10425, 0.15362
AE N=60	0.50980	0.92857	0.27384, 0.64494	0.33641, 0.74088
AE N=120	0.56075	0.90909	0.24730, 0.68374	0.29386, 0.77499
AE N=360	0.96429	0.80597	0.15626, 0.24205	0.19043, 0.28721

$\theta_1 \sim \text{Bernoulli}(0.5)$ and $\theta_2 = 1 - \theta_1$, i.e. both clusters are balanced and the choice is random. This should be the ideal case as it happens in classification problems where classes are balanced.

Case 2: imbalanced clusters

This case is exactly the same as before but $\theta_1 \sim \text{Bernoulli}(0.85)$, this is, we assume a situation where clusters are not balanced. Although the original authors do not cover imbalanced experiments, these cases are closer to real-world data as we will see in next section.



(a) Network segment 1

(b) Network segment 2

(c) Network segment 3

(d) Network segment 4

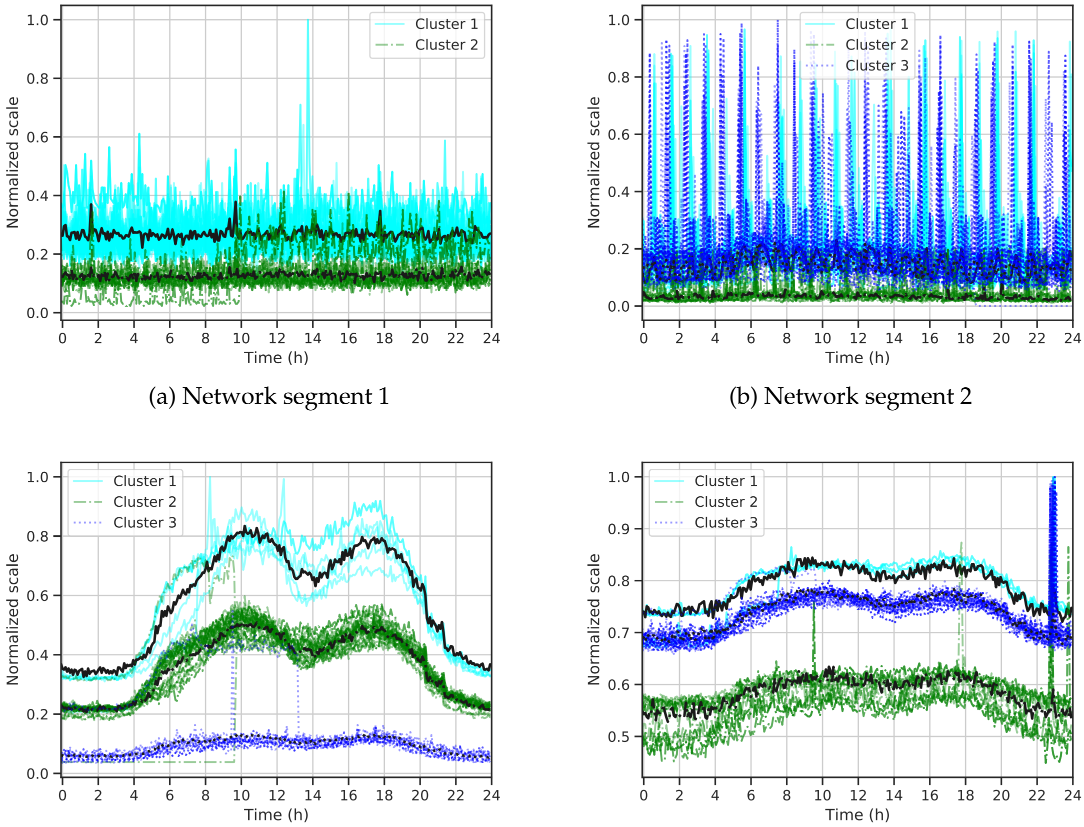
Figure 4.7: Network segments daily time series for three months

Figure 4.6 displays simulations corresponding to the case of imbalanced clusters. We see that the centroids are pretty similar, and many curves even overlap. Right-hand side of the figure shows the result of k-means algorithm. In this case, we see that using the median suppose an improvement since the centroids has no noise and all the samples are noisier.

Results of all methods can be found in Table 4.1 and 4.2. As we see, simple methods as k-means, PCA and FPCA scores better than FAE. This is due to the fact that neural networks usually need many samples or proper hyperparameter tuning. As commented before, the k-means algorithm performs better adding the median. In terms of classification metrics, it does not impact the final result, it only affects the obtained centroids as it can be seen in terms of the L^1 and L^2 distance.

4.8.3 Real-world data: data center monitoring

From the network under study, we have selected four network segments that provide different services, which represent some of the most interesting cases that arise when facing network monitoring in real-world environments. Each time series is a full day of the metric aggregated in intervals of five minutes, and days are not necessarily consecutive. In this case, we recall that one of the objective is to classify similar trends together, so we want to see if this is the case and it can really improve the monitoring systems. Once this is done, they can be fed into systems such as [Vega et al., 2018], providing a more consistent approach instead of guessing a weekly periodicity of the time series.



(a) Network segment 1

(b) Network segment 2

(c) Network segment 3

(d) Network segment 4

Figure 4.8: Clusters for each network segment. Best centroids are represented as wider black lines with the same line style.

1. *Network segment 1, survivable branch appliance*: it contains several devices responsible of the uninterrupted Voice over IP (VoIP) communications of different branches of the company. Figure 4.7a shows the time series of the number of connections among the day. Several trends can already be guessed but, in fact, it seems to have a stable behavior between 15 to 75 connections. Figure 4.8a shows the two detected clusters: one of high activity (Cluster 1) and another one of low activity (Cluster 2). This simple classification also shows that Cluster 1 is more likely to have extreme behaviors whereas cluster 2 is more stable but it also jumps to levels similar to the cluster 1 at the end of the day. Other state-of-the-art systems that would incorrectly consider that, when Cluster 2 intersects the centroid of Cluster 1, the level of dispersion would be the one of Cluster 1, while we are able to properly detect the real trend.

2. *Network segment 2, regional DHCP and DNS servers*: it is composed of several servers that act as DHCP or DNS of the company for a whole geographical region. Figure 4.7b displays the time series of the number of connections among the day. Although it seems to have a very bursty behavior, it is periodic and has to do with the synchronization with the rest of regional servers. Figure 4.8b depicts three clusters. Cluster 1 and Cluster 2 are work days and Cluster 3 is weekends. We observed a limitation of our method: since we consider daily trends, any trend that is not periodical each 24 hours or a multiple of 24 hours can lead to

Table 4.3: Results of the clustering methods for network segments 1 and 2 for several sizes of dataset.

	Network Segment 1		Network Segment 2	
Methods	**SC**	**DBI**	**SC**	**DBI**
F. k-means (1 month)	0.61714	0.47776	0.42202	0.80097
with median	0.61714	0.41333	0.49735	0.76369
F. k-means (2 months)	0.59032	0.65805	0.32844	1.05486
with median	0.59032	0.57102	0.37922	0.87207
F. k-means (3 months)	0.57107	0.66552	0.29950	1.22141
with median	0.57107	0.59909	0.34325	1.24314
PCA (1 month)	0.61714	0.54452	0.49735	1.00426
PCA (2 months)	0.59032	0.59857	0.37978	1.31938
PCA (3 months)	0.57107	0.61461	0.36283	1.54504
FPCA (1 month)	0.61714	0.56901	0.45510	1.72075
FPCA (2 months)	0.59032	0.61359	0.37978	2.03991
FPCA (3 months)	0.57107	0.59858	0.34250	1.82842
AE (1 month)	0.61714	0.81439	0.47984	4.58737
AE (2 months)	0.59032	0.58760	0.34726	2.30413
AE (3 months)	0.55785	0.60325	0.33632	2.23477

* SC: More is better. DBI: Less is better.

Table 4.4: Results of the clustering methods for network segments 3 and 4 for several sizes of dataset.

	Network Segment 3		Network Segment 4	
Methods	**SC**	**DBI**	**SC**	**DBI**
F. k-means (1 month)	0.89050	0.11431	0.82868	0.20817
with median	0.85614	0.17242	0.82868	0.22088
F. k-means (2 months)	0.84572	0.20396	0.83508	0.25474
with median	0.84572	0.18104	0.83508	0.25324
F. k-means (3 months)	0.86293	0.21974	0.78822	0.30459
with median	0.86293	0.21802	0.77675	0.29689
PCA (1 month)	0.89050	0.21448	0.85415	3.76542
PCA (2 months)	0.84572	0.22227	0.83508	3.89442
PCA (3 months)	0.86293	0.22415	0.81444	3.94855
FPCA (1 month)	0.89050	0.23799	0.85415	0.26204
FPCA (2 months)	0.84572	0.28235	0.83508	0.29437
FPCA (3 months)	0.86293	0.26692	0.81444	0.32805
AE (1 month)	0.89050	1.85030	0.77582	6.93602
AE (2 months)	0.84487	0.30296	0.75568	2.62730
AE (3 months)	0.86293	0.23725	0.74551	5.33807

multiple clusters. Nevertheless, clusters are still usable for daily baselines and it only impacts the estimation errors since cluster sizes are smaller.

3. *Network segment 3, payment gateway 1*: it is composed of only one server that acts as gateway for the payment mechanism, providing not only connectivity to bank companies but security mechanisms to protect the data of the transactions. Figure 4.7c depicts the time series of the number of connections among the day. Clearly, it exposes three different trends that are stronger in the peak hours of

activity of the day. Figure 4.8c exposes the three observable clusters. This case proves the usefulness of this approach. One of the motivations was to help baselines systems by grouping similar trends. In this case, operators were unable to estimate when a day belong to Cluster 1, since it has some random and user-dependent behavior. Our system provide a baseline for these days preventing false alarms and improving monitoring capabilities.

4. *Network segment 4, payment gateway 2*: similar to the previous one, it is composed of only one server that acts as gateway for the payment mechanism. In Figure 4.7d, the time series of the number of connections among the day can be observed. Although it is related to a payment system as before, we see a rather different case. Some similarities are still present such as the three trends and the peak hours of the morning and the afternoon, but the peak activity at the last hour of the day is the most remarkable event that makes it difficult to differentiate the three existing trends. Because of this, we employ the aforementioned companding procedure using the μ-law with $\mu = 1023$, which makes the apparent clusters more spaced apart. Figure 4.8d shows the resulting clusters and conclusions are similar to previous case since these systems have similar purpose.

For all the network segments described above, we have fitted the methods described in previous sections. Results in terms of **SC** and **DBI** are shown in Table 4.3 and Table 4.4. As a reminder, **SC** does not take into account centroids, so it is common to have ties if two methods formed the same clusters. Thus, we must look at the **DBI** to distinguish which algorithm scores better.

K-means algorithm is usually better than projection-based algorithms, which is reasonable since projections reduce the amount of information, whereas direct approach can work with the whole signal. Difficult cases as network segments 2 and 4 are particularly hard for PCA or AE, which can be due to the loss of information of the projection and the small number of samples for training. It is remarkable that in case of network segment 4, FPCA scores better than PCA, given that it is able to differentiate better the two clusters that were very close.

4.9 Conclusion

In this chapter, we have successfully created a model that represents different functional modes or daily trends. This classification provides a deep insight of the series behavior without depending on expert information which, in some cases, is obvious (e.g. workings days and holidays behave differently) but in many other cases it is not. Thus, providing new mechanisms that automatically detects these different trends help to provide the best model.

Besides, we manage to show the relationships among the different projection techniques, which leads to the development of new methodologies where we are able to use FPCA or PCA as lower bounds in terms of performance to properly dimension more complex and capable neural networks.

Once we have a model that groups similar time series, we can rely on simple methods to provide monitoring and baselines for every network of a datacenter, without the necessity of operators who configure them manually. Thus, it is a more scalable and economical approach that simplifies the tasks of network operators and analysts.

5 Variable size network registers: DNS registers

5.1 Introduction

In this chapter, we introduce another type of network register. We will extract information of the web browsing behavior using just the DNS records. In contrast to previous chapters and as we will cover next, data are sequences of strings. Consequently, special techniques will be covered to deal with this particular case.

We cover three approaches: the first two ones, TF-IDF and doc2vec, are based on projections, i.e. they rely on an intermediate representation in a numeric space; and the last one is a direct approach based on sequence modeling neural networks. The three methods are compared to see the performance and the impact of the DNS cache.

5.2 State of the art

Deep Packet Inspection (DPI) allows network operators to be able to have a deeper insight than the one with just flow aggregates. In many cases, DPI techniques are not employed due to the additional overhead that introduces in the performance of the monitoring systems, but it has several use cases when it is necessary. The main example is security: firewalls, Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) perform DPI to analyze payloads of sensible protocols as HTTP, HTTPS, DNS or SMB to detect possible security breaches.

In the last few years, IDSs have been introducing brand new techniques based on neural networks, in particular, in CNN and in RNN. We have observed a change in the perspective, some of previous work in DPI just consider byte frequency information to classify a packet as anomalous, as PAYL [Wang and Stolfo, 2004]. Besides, many of the previous papers that consider payload rely on feature extraction to feed a ML classifier [Lin et al., 2015].

Nowadays, neural networks are becoming increasingly popular and so they are systems that used them to build intrusion detection algorithms. Authors in [Liu et al., 2019] introduced Payload CNN (PL-CNN) and Payload RNN (PL-RNN) which are just neural networks that are fed with the preprocessed payload of each packet. Indeed, preprocessing techniques applied by the authors are word embeddings word2vec [Mikolov et al., 2013] that are explained in next sections. The main difference of our approach is that we consider further work of the same authors and use doc2vec [Le and Mikolov, 2014] instead of just feeding word2vec vectors to a RNN and a CNN.

Far from security, there are other many applications for DPI. In particular, we will look at the main topic of chapter 5, which is traffic identification and classification: know what the user is actually doing. Authors in [Morichetta and Mellia, 2019] proposed a classification of traffic based on iterative modification of the density-based spatial clustering of applications with noise (DBSCAN) algorithm. Compared to our objective, there are two differences: first, their approach is unsupervised, so labels are not needed for training and second, their categories are coarse-grain, i.e. in many cases they do not resemble web pages as we want but general concepts as video streaming.

A seminal work of this project is [García-Dorado et al., 2018]. In this paper, authors propose DNSPRINTS, a system based on DNS weighted footprints that build an exhaustive algorithm that emulates the behavior of the DNS cache. Encryption of the traffic is more and more popular, and this makes sniffing protocols such as HTTP totally unfeasible, which makes DNS sniffing a promising alternative. Nevertheless, even for DNS, encryption is gaining popularity and DNSSec [Rose et al., 2005] or DNS over HTTPS (DoH) [Hoffman and McManus, 2018, Mozilla Foundation, 2020] are the most common ways of addressing this. Fortunately, similar data can be obtained from TLS's Server Name Indication (SNI) field. The only issue is that we need information about TTL for training this system, whereas this alternative cannot obtain any similar concept. Thus, in this project, we want to train new methods just using the sequence of DNS queries so that it can also be trained with, for instance, SNI data. This make our system not only an improvement from current DNSPRINTS but also future-proof to incoming changes in network operations.

5.3 Description of the problem

The objective of our model is to identify the domain the browser is searching given the sequence of DNS queries it has performed. So, let D be our vocabulary, i.e. the set of all domains. Also, all the possible sequences may have many more elements outside D, all these words, that are basically any string that is returned in the DNS queries, will be called $\mathbb{S}$. The first issue is the dimension of D. We are normally solving classification problem where there are few classes, but, in this case, the number of classes is potentially infinite. Furthermore, the different subdomains we use as predictors are also potentially infinite and they come in sequence with no fixed length. Here is an example of what our model f should do

$$\{\text{encrypted-tbn0.gstatic.com, twitter.com, twitter.com}, \dots\} \xrightarrow{f} \text{twitter.com}$$

In order to specify properly the problem, we will call the sequence of subdomains S_d, where $S_d = \{s_i\}_{i=1}^{N(d)}$ and the domain is $d \in D$. Although these sequence S_d may seem constant, we will see cases where there is some random behavior, mainly two cases: first, the domains are not queried because they are in the cache of the browsers and, second, some parts of the name of the domains are random.

It may be thought to be an easy problem, since the main domain d is likely part of S_d, but the problem is that, for other domain d', it may also happen that $d \in S_{d'}$, in our example, that Twitter appears in other pages. Indeed, Twitter is always referenced whenever a button of "Sign in with Twitter" is placed, so it is more common than expected and makes the problem way more difficult.

5.4 Classical approach: term frequency and inverse document frequency

In this very first approach, we want to think of this as a recommendation algorithm based on k-Nearest Neighbors (k-NN) or a MLP classifier. As a training set, we use, at least, a sample sequence of each domain that we want to identify. For each document we want to classify, we assign the class or domain of the nearest neighbor (if $k = 1$, if we have more samples, we can use $k = 3$ or $k = 5$) or the one given by the MLP classifier.

The k-NN algorithm needs a metric space, in this case, the Term Frequency - Inverse Document Frequency (TF-IDF) methodology provides one. First, we define TF as

$$\text{TF}(s, S_d) = \begin{cases} 1 + \log_2 \text{freq}(s, S_d) & \text{if freq}(s, S_d) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

and IDF as

$$\text{IDF}(s) = \log \frac{|D|+1}{|D_s|+0.5}, \tag{5.2}$$

where $D_s = \{d \in D : s \in S_d\}$, i.e. the domains whose sequences contain s. Then, for each $d \in D$, we define v_d as the vector

$$v_d = [\text{TF}(s, S_d) \cdot \text{IDF}(s) \text{ For } s \in \mathbb{S}] \tag{5.3}$$

As we see, v_d is a vector with infinite dimension. This would be problematic, but, the number of non-zero terms is finite due to the form of $TF(s, S_d)$. In order to measure the similarity between two domains, we use the cosine distance defined as simply the cosine of the vectors v_d,

$$\begin{aligned} \text{sim}(d, d') &= \cos(v_d, v_{d'}) \\ &= \frac{v_d \cdot v_{d'}}{|v_d||v_{d'}|} \\ &= \frac{\sum_{s\in\mathbb{S}} \text{TF}(s, S_d)\text{TF}(s, S_{d'})\text{IDF}^2(s)}{\sqrt{\sum_{s\in\mathbb{S}} \text{TF}(s, S_d)^2\text{IDF}^2(s)}\sqrt{\sum_{s\in\mathbb{S}} \text{TF}(s, S_{d'})^2\text{IDF}^2(s)}} \end{aligned} \tag{5.4}$$

As we see, the sum of the scalar product and the norm does only involve the non-zero terms, so, in fact, the dimension of $\mathbb{S}$ does not impact the algorithm, but the performance depends on length of the sequence. To properly use the TF-IDF representation, a sparse matrix is usually employed where only non-zero terms are stored. These sparse vectors can be fed into other supervised methods such as a MLP classifier.

5.5 Modern approaches: neural networks

Sequence modeling both in time series and text processing is one of the areas where neural networks excels. In this case, we want to follow a similar approach to neural networks that are able to classify texts or paragraphs (in our case, S_d) into categories (in our case, the domain d). Two approaches will be cover: first one is based on the construction of an embedding based on context and second one is a direct approach that exposes an end-to-end neural network model working.

Prior to the models, we want to highlight that neural networks do not work directly with strings as TF-IDF, they rely on building first a vocabulary. This vocabulary is a mapping of each word to a number. Due to dimensionality of the data, normally the vocabulary is capped and words that are less frequent are considered as OOV (Out of Vocabulary) tokens. Also, other tokens are usually added as start of the sequence, end of the sequence or padding token. These last tokens solve the problem of variable length of the sequence, since the neural network does only work with inputs of fixed dimension.

5.5.1 Generic embeddings: Word2Vec and Doc2Vec

First, in order to understand Word2Vec, it is necessary to understand the two techniques used to perform the algorithm: Continuous Bag of Words (CBoW) architecture and Skip-Gram. Both architectures rely on the same concept: an artificial target variable to train the neural network.

Continuous Bag of Words (CBoW)

In this first case, we build sequences where we delete an element, for instance, in sequence $S = s_1, \ldots, s_N$, we call S^i to the sequence without s_i and the idea is to train a neural network so that $f_\phi(S^i) = s_i$, using some classification loss functions such as the logarithm of the cross entropy. Once the neural network is trained, we only need to specify the embedding, for these purpose, we use the same approach as the AE, use a hidden layer of size K. Normally, since these networks can easily have millions of parameters ($\approx |\mathbb{S}| \times$Sequence length $\times K$), it is recommended to keep the architecture as simple as possible and usually it is just a hidden layer and an output layer with a soft-max activation function. Figure 5.1 shows the architecture of the CBoW.

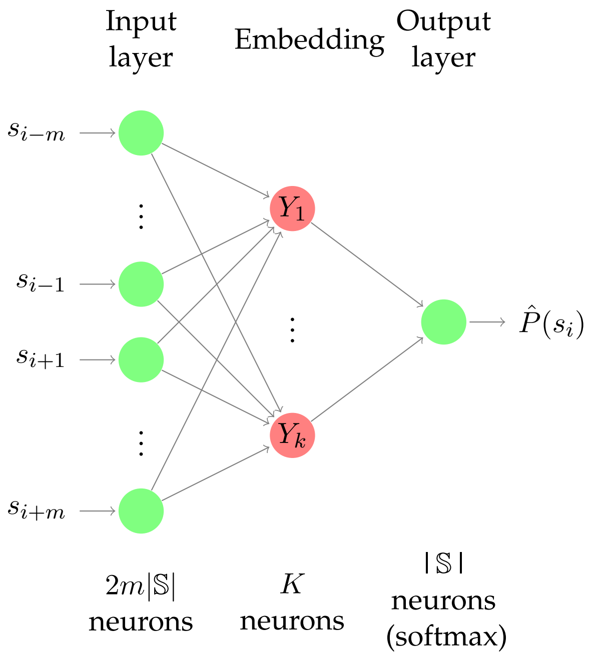


Figure 5.1: Continuous bag of words architecture

Skip-Gram

As before, we build an artificial target to predict. In this case, the approach is completely the opposite, just with the information of one word s_i, we try to guess the context S^i, i.e. $f_\phi(s_i) = S^i$. In terms of parameters, these problem looks heavier and, in fact, it is known to be slower in terms of convergence than CBoW but it also results in better representations. As before, a single hidden layer is usually considered to avoid an excessive number of parameters. Figure 5.2 shows the architecture of the Skip-Gram network.

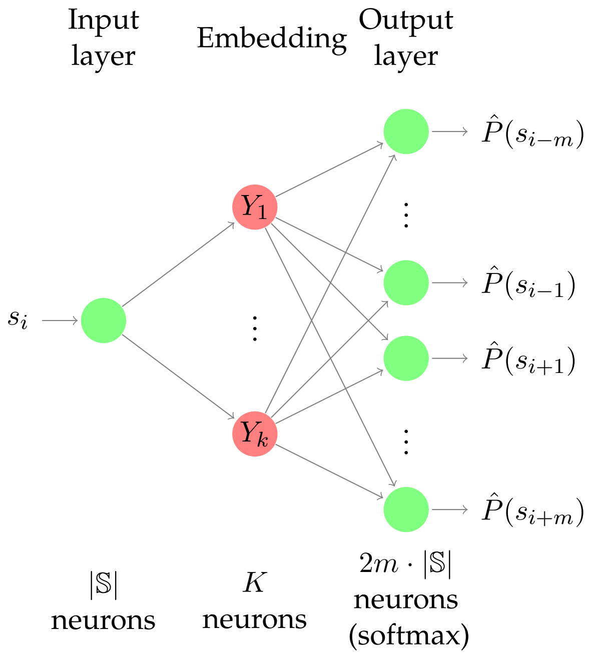


Figure 5.2: Skip-Gram architecture

Once the embedding is trained and ready, for each word, we have a number of $\mathbb{R}^k$. Given this, now, the problem is just a classification problem in $\mathbb{R}^k$ with many classes. As long as we have enough samples to train a classifier, we will be solving the problem. In order to compare the obtained result with incoming architectures, we use another neural network (a MLP) as classifier.

Once word2vec methods are clear, it is easier to follow the doc2vec approximation. As well as before, there are two possible approximations Distributed Memory Model of Paragraph Vectors (PV-DM) and Distributed Bag of Words Model of Paragraph Vectors (PV-DBoW).

First case, PV-DM is an extension of CBoW. For each word, we compute the representation using neural network in Figure 5.3. Once all vectors are computed for every word based on some tags, the word itself and the context, the resulting embedding is concatenated or averaged in a final vector which represents the whole paragraph. Tags act as a way of adding information of the domain to the network, so tags weight matrix encodes an embedding of the tags.

Second, PV-DBoW follows an analogous procedure than skip-gram, but instead of creating the context from the center word, it is performed with the tags. Figure 5.4 explains this architecture. Once network is trained, likelihood of the observed words can be computed to see from new samples the estimated probabilities of belonging to a class.

Implementation of all methods can be found in Python library for topic modeling gensim [Rehurek and Sojka, 2010].

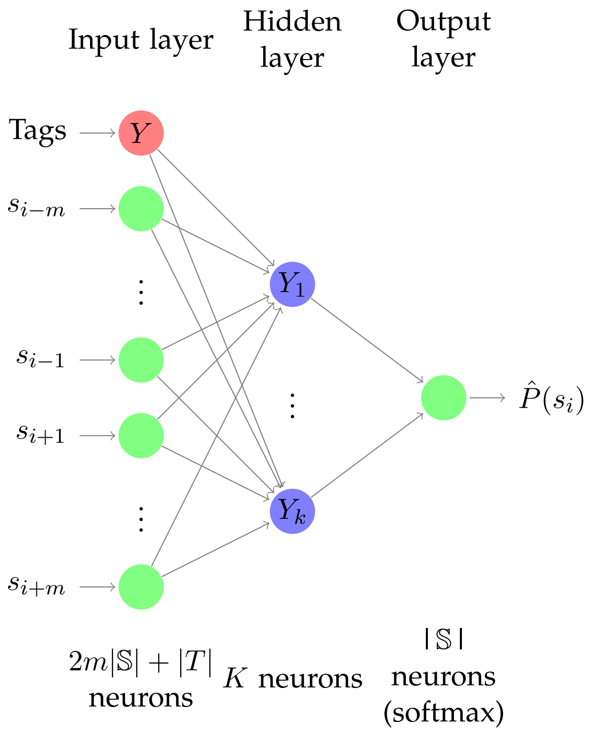


Figure 5.3: Fundamental unit for the distributed memory model for paragraph vector (PV-DM).

5.5.2 Custom neural networks: vectorizing layer and direct approach with recurrent neural networks

Although the usual approach to text processing is unsupervised, we have categories available so we can create directly a classifier. Thus, the objective now is to create a direct model based on neural networks. The architecture is shown in Figure 5.5.

In this case, in contrast to previous ones, we use an LSTM layer for the input, i.e. the input is treated as a sequence. If this is not done, the input would be treated as a vector. This means that having the whole sequence of subdomains but the first one would lead to a totally different vector, whereas with LSTM cells, neural network can learn patterns in the sequence.

5.6 Data acquisition and preprocessing

One issue here is how to get data to train the neural networks. For that purpose, authors of [García-Dorado et al., 2018] built a system that is composed of a capture engine only listening to DNS packets and a web browser that access the objective domain.

This system, hereinafter called "robot", is used to query the information for domains include in the top 100 and 2500 of Alexa. The robot provides a register that includes much more information than the performed DNS queries, e.g. the Time-To-Live or the server. Nevertheless, these extra attributes will not be used to feed the neural network, since we want to improve the system of [García-Dorado et al., 2018] so that it does not requires all that information.

Once we have acquired our dataset, we need to adapt it to be fed into a neural network. The first step is to build a vocabulary. The vocabulary models $\mathbb{S}$ by adding two extra tokens or words and eliminating the least used ones. So, the vocabulary will be

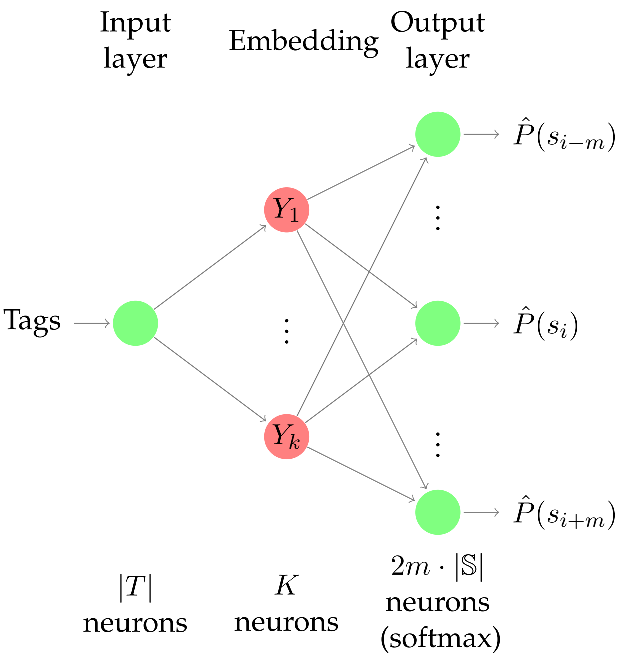


Figure 5.4: Distributed bag of words model for paragraph vector (PV-DBoW).

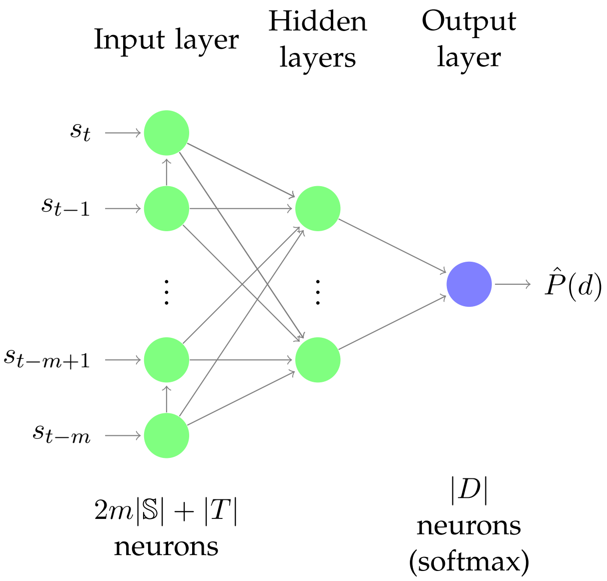


Figure 5.5: Direct approach to text-classification

called $\hat{\mathbb{S}} \subset \mathbb{S} \cup \{OOV, BLK\}$, where OOV and BLK stand for Out of vocabulary token and Blank token respectively. The first one is used whenever a element in $\mathbb{S}$ is not in $\hat{\mathbb{S}}$ and the second one is used when a sequence is shorter than the maximum allowed by the neural network.

Once this is done, we map each word to a number. This can be done through a one-hot encoding (so each word is mapped to a number in $\{0,1\}^{|\hat{\mathbb{S}}|}$) or by a simple hashing (so each word is mapped to a number in $\{0,\ldots,|\hat{\mathbb{S}}|\}$). Always the first approach is preferred, since distances between words in the second space are not representative whereas, in the first case, all the words are equally spaced.

Although it is clear that the first option is better, we have to think that this means that if vocabulary size is around 100 000, we are working with an input space of dimension $100000m$ where m is the length of the sequence. This makes everything so expensive to compute that we have to stick to the second option in many cases.

Keras have an Embedding layer which helps to deal with the second case. This layer uses a linear operator and a scaler to map the input space to fixed dimension real vector space, for instance, if we want to map the sequence of integers to a sequence of real vectors. This provides a mixed solution that it is usually used so that the dimension of the input layer is not too big and the topology of the words (that now are vectors in $\mathbb{R}^k$) is more coherent

5.7 Results

In this section, we will evaluate the different methods explained before TF-IDF, doc2vec and the direct approach (RNN). For all these methods, we will evaluate the performance with several datasets that will model the impact of DNS caching. We recall that each time a system queries a DNS domain, the result is returned with a Time-To-Live (TTL) field. This means that as long as TTL has not expired, the device will not ask for the same domain while accessing it.

To obtain the datasets, we used the aforementioned robot to query the top 100 and top 2500 of worldwide most visited domains according to Alexa [Amazon Web Services, 2020]. This is done for two web browsers: Chromium, the open-source alternative of Google Chrome, and Mozilla Firefox.

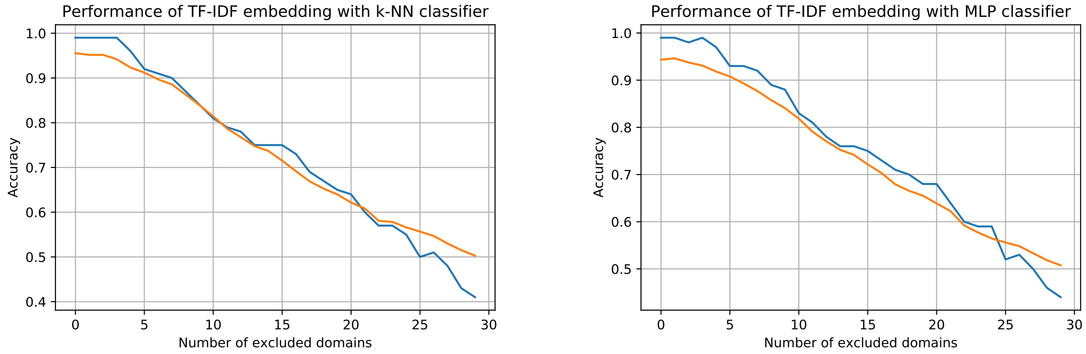
In real traffic, we have observed that a set of just four domains (Google, Facebook, Apple and Microsoft) can accumulate more than 60% of the global traffic in terms of number of accesses. In fact, these tops follow a Pareto's Law, as it happens in salary distribution.

5.7.1 Results for TF-IDF

First, we test the TF-IDF embedding. TF-IDF embedding retains a lot of information from the texts, in fact the frequencies of the words, but we expect that high-dimensional data may arise when coping with huge datasets. For that purpose, sparse matrices are used to avoid computational issues. Nevertheless, dimensionality of the data can also impact the convergence of the algorithm, so we do not have high expectations in this method for huge datasets.

Figure 5.6 shows the results in terms of the accuracy for the dataset obtained for the Top 100 of Alexa. The dataset is composed of 15000 samples of the top 100 domains in terms of visits. The train subset, in this case, it is just composed of one sample per class whereas the test is the rest of the dataset. Although this split seems very aggressive, bear in mind that this representation can get highly dimensional and it can be affected by noise (random subdomains). As we mentioned before, the objective of the experiments is to see the impact of the DNS caching in the results. We observed than results are not affected when 4 or 5 domains are in the cache. However, from that point on, the results are affected by an approximate ratio of 10% per 5 excluded domains.

For the case of the top 2500, we expect worse results since dimensions are way higher. In this case, the train set and test set are just classic random split with 70% of the sample for training and 30% for test. Figure 5.7 displays the results for this case. In this example, we see that the behavior is almost a straight descending line. MLP classifier scores better than k-NN due to its complexity, but it cannot show results higher than 85% of accuracy.

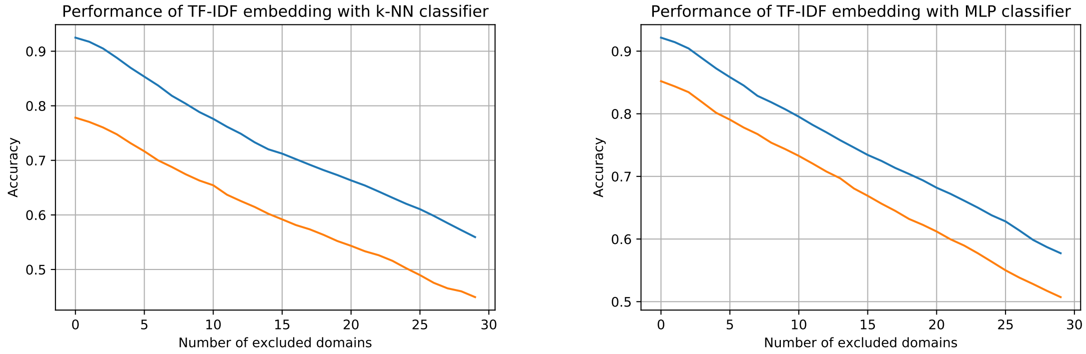


(a) k-NN classifier.

(b) MLP classifier.

Figure 5.6: Results for the top 100 of Alexa for TF-IDF embedding. Blue line represents training data set and orange line test dataset.

It can be observed also that in both classifiers we have overfitted the training data so the performance for the test subset is always lower.



(a) k-NN classifier.

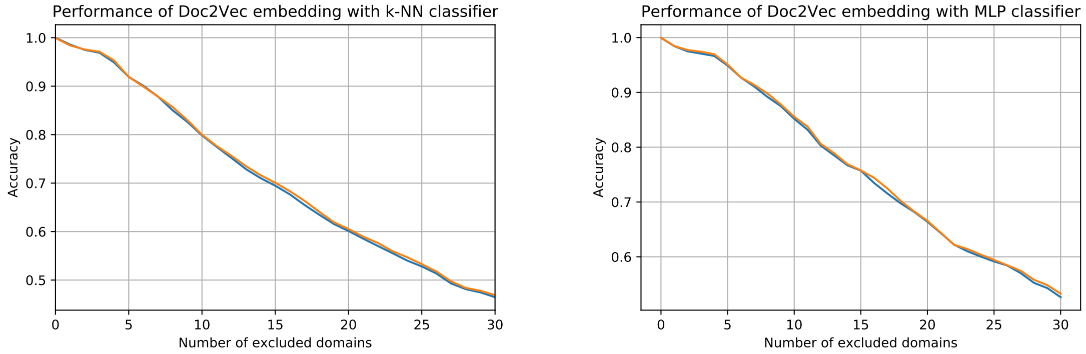
(b) MLP classifier.

Figure 5.7: Results for the top 2500 of Alexa for TF-IDF embedding. Blue line represents training data set and orange line test dataset.

5.7.2 Results for Doc2Vec

In this case, we split the dataset into a train set composed of 70% of the sample and a test set composed of 30% of the sample. Then we trained the embedding using both PV-DM and PV-DBoW algorithms with different sets of parameters. Once embeddings are trained, we use k-NN and a MLP classifier to solve the classification problem now in some real-valued space.

As before, this is done for both top 100 and top 2500 of Alexa. Figure 5.8 shows the results for the top 100 of Alexa. We observed that both MLP and k-NN classifiers score similarly and, again, the decay of the amount of information when eliding subdomains of the sequence is linear and more or less with a similar slope to TF-IDF. As a positive advantage, there is no overfitting in this case. About the hyperparameters, we found out that a hyperparameter was critical: whether to concatenate or to average the representations of the words. When doing concatenation, the embedding retains information of the order of the sequence. However, retaining this information cause the algorithm to overfit and not to generalize really well. On the other hand, averaging

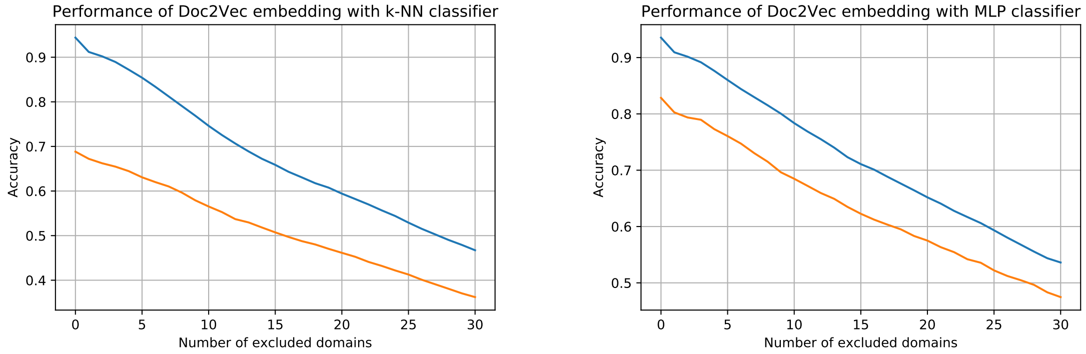


(a) k-NN classifier.

(b) MLP classifier.

Figure 5.8: Results for the top 100 of Alexa for doc2vec embedding. Blue line represents training data set and orange line test data set.

provides a way of representing the words and their contexts partially ignoring the order of the sequence.



(a) k-NN classifier.

(b) MLP classifier.

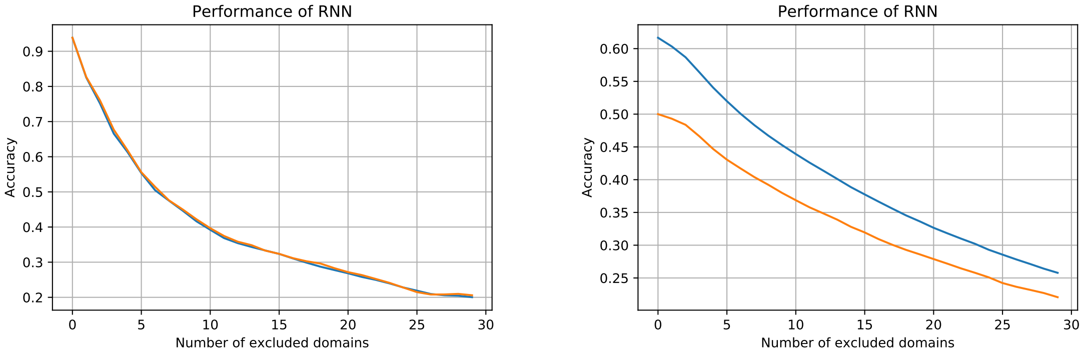
Figure 5.9: Results for the top 2500 of Alexa for doc2vec embedding. Blue line represents training data set and orange line test data set.

For the top 2500, Figure 5.9 represents the score for k-NN and MLP classifiers. Now, overffiting is clear and the difference between k-NN and MLP also is more obvious, being MLP better than k-NN. It is similar also to TF-IDF with no highlighted differences.

5.7.3 Results for RNN

Following the previous case, for both top 100 and top 2500 datasets, we performed a train-test split with 70% of the sample for training and 30% for test. In the first case, we found out that performance is similar to other methods as we can see in Figure 5.10 and the difference between train and test is negligible (which means there is no overfitting issues). However, the decay of the accuracy as function of the number of missing subdomains is quite steep and not linear in this case. This can be due to the fact that RNN takes into account order of the domains whereas TF-IDF and doc2vec with averaging do not, which makes the algorithm more sensible to missing elements of the sequence.

In the second case, the achieved accuracy is way worse than TF-IDF. In this case, the decay is not so steep but since accuracy is below 50%, it does not make sense to consider



(a) Top 100.

(b) Top 2500.

Figure 5.10: Results for the RNN for the top 100 and top 2500 of Alexa. Blue line represents training data set and orange line test data set.

the results. Again, neural network methods fail to properly converge when the number of classes is too high.

5.8 Conclusion

We have observed several text modeling techniques applied to DNS data. As in many situations, we did not find out a perfect technique able to be absolutely used in every situation. While TF-IDF seems the weakest (prior results), it is really useful for situations where there is a large number of domains and, thus, neural networks may not converge making doc2vec and RNN worthless. However, TF-IDF comes with an overfitting issue that doc2vec and RNN do not. In general, doc2vec is better than RNN, due to the fact that order of the sequence in this case provides no information. We leave as future work to improve RNN performance through data augmentation with permutations, since it looks promising if we are able to cope with the aforementioned problems.

Conclusion 6

6.1 Summary

In this work, we have covered three different topics: time-unaware models, time-aware models and models for text-based registers. In all three cases, we follow a similar approach testing different alternatives to choose the best one for each case, aiming at the generalization of the results. Besides, we support all the evidences with real-world data to test the algorithms in an environment as close as possible to a real deployment.

First main part of this document explored the application of different probabilistic model to network KPI modeling where there is no dominant time-dependent trend. We observed that, although state of the art employed many models for the same or almost the same purpose, there was no perfect model able to cope with every situation that we observed in a scenario such as multi-point RTT measurements. Thus, we found that criteria that use both goodness of fit and complexity help choosing the right model for the right situation. Also, when the considered time window gets longer, we found out that no model was able to properly fit the situation, so we employed projections to reduce the variance of the data. This approximation also allowed us to follow a more scalable deployment based on flow aggregates instead of storing all flows, what it is becoming unfeasible in some real cases.

Second, we moved to cases where a trend was dominant, i.e. the distribution changes as a function of the time. We considered daily curves of network KPIs, each one as a sample, and use FDA techniques to provide a model. This model has a clear advantage over the previous one, it is multi-modal, i.e. we assume that there are many possible daily behaviors. In contrast to classical methods where base lines were built with assumptions such as all Mondays have the same behavior, here we proposed a technique to automatically separate the sample into clusters and classify each curve to a cluster.

Last, we covered how to process text registers. These text registers can come from a very different source such as application logs, DPI or even social networks or chat bots. In this case, we use the information from DPI of DNS packets to characterize the web browsing behavior. We evaluated different methods: some of them classical such as TF-IDF and some of them based on neural networks. We paid attention to the loss of information when we remove elements from the sequence of subdomains emulating the DNS cache working.

6.2 Contributions

In all cases, we have studied the advantages and disadvantages of many models to determine the best use case scenario. As we have anticipated in chapter 1, the novelty of this work can be seen in terms of its main contributions.

For the first part, we have analyzed the application of model selection techniques to choose the best model for a stable KPI. Since the trade-off bias-variance was central in all this discussion, we have studied how projections can help to reduce the bias. In all this, we observed that the window size has an important effect in event detection, being then a critical parameter to be assessed in many systems. All these contributions that appear in this work are already published and can be found in [Perdices et al., 2019].

Next, we have studied the state of the art of unsupervised classifiers for functional data, daily curves of network KPIs. Using metrics for clustering methods extended to the functional setting, we have assessed which model is the best and see advantages or disadvantages. Besides, we showed that neural networks are sensible to the inputs, so using a functional basis as input of the neural network can help the neural network to consider the topology and structure of the space. This work was sent for publication to a journal and reviewers have considered it for a second round of peer review.

To conclude, we have also studied text processing methods that include text embeddings such TF-IDF or doc2vec and sequence models such as RNN. All the models were studied and compared in terms of their performance and the impact DNS caching policy. We have observed that many of them look promising, but performance suffers when the number of domains is too high leading not only to worse results but also overfitting. An extended version of this work is also intended to be published soon.

In all cases, an important factor in research is also the applicability of the results to real-word situations. In this work, we have applied the results and techniques to real-world situations. In the first case, we have shown the results of assessing the performance of the network equipment, which was done for both an international level telecommunication company and a national logistic company. The second part also displayed results obtained inside a Spanish energy company. About the DNS analysis, we are preparing a prototype alongside previous systems to be tested and compared in situation when not only accuracy but also prediction speed or parallelization can be key.

6.3 Future work

As future work, we leave some topics open that have yet to be explored and that can unveil the full capabilities of all the proposed models and techniques.

For the first part, we plan to extend data gathering modules to improve interoperability with SDN and virtualized elements. Additionally, and as stated above, we are starting to study the compatibility of adPRISMA with packet sampling techniques to alleviate computational burdens. Also, we point to the exploration of RTT decomposition as predictor of network overloads and failures.

For the time-aware modeling, we want to extend the system to support multivariate time-series as well as introduce other factors in the comparison as the time required for training or for test. Besides, we are working on also the use of this cluster and centroids to build improved regressors such as a RNN that play the same role as the base line.

As we foreseen before, we can also add to the text processing techniques new improvements such as data augmentation to cope with the effect of order or the missing elements of the sequence so that we are able to improve the performance of the RNN. Furthermore, we want to test new systems to collect data that do not only rely on DNS but also on TLS so that we can work even if DNS is not captured or it is encrypted, using the Server Name Indication and Server Certificate of the TLS handshake as a substitute.

Bibliography

[Aitken et al., 2013] Aitken, P., Claise, B., and Trammell, B. (2013). Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011.

[Akaike, 1974] Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.

[Amazon Web Services, 2020] Amazon Web Services (2020). Alexa - Top sites.

[Arthur and Vassilvitskii, 2007] Arthur, D. and Vassilvitskii, S. (2007). K-Means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA. Society for Industrial and Applied Mathematics.

[Baras et al., 1997] Baras, J. S., Ball, M., Gupta, S., Viswanathan, P., and Shah, P. (1997). Automated network fault management. In *Proceedings - IEEE Military Communications Conference MILCOM*, volume 3, pages 1244–1250. IEEE.

[Bari et al., 2013] Bari, M. F., Boutaba, R., Esteves, R., Granville, L. Z., Podlesny, M., Rabbani, M. G., Zhang, Q., and Zhani, M. F. (2013). Data center network virtualization: A survey. *IEEE Communications Surveys Tutorials*, 15(2):909–928.

[Ben Slimen et al., 2017] Ben Slimen, Y., Allio, S., and Jacques, J. (2017). Anomaly prevision in radio access networks using functional data analysis. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–6.

[Ben Slimen et al., 2018] Ben Slimen, Y., Allio, S., and Jacques, J. (2018). Model-based co-clustering for functional data. *Neurocomputing*, 291:97–108.

[Bickel and Frühwirth, 2006] Bickel, D. R. and Frühwirth, R. (2006). On a fast, robust estimator of the mode: Comparisons to other robust estimators with applications. *Computational Statistics & Data Analysis*, 50(12):3500 – 3530.

[Boutaba et al., 2018] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1).

[Broomhead and Lowe, 1988] Broomhead, D. S. and Lowe, D. (1988). *Radial basis functions, multi-variable functional interpolation and adaptive networks*. Royals Signals & Radar Establishment, Malvern, Worcs.

[Carisimo et al., 2017] Carisimo, E., Grynberg, S. P., and Alvarez-Hamelin, J. (2017). Influence of traffic in the stochastic behavior of latency. In *TMA PhD school*.

[Chen et al., 2016] Chen, Z., Wen, J., and Geng, Y. (2016). Predicting future traffic using Hidden Markov Models. In *Proceedings - International Conference on Network Protocols, ICNP*, volume 2016-December. IEEE Computer Society.

[Claise, 2004] Claise, B. (2004). Cisco Systems NetFlow Services Export Version 9. RFC 3954.

[Claise et al., 2013] Claise, B., Trammell, B., and Aitken, P. (2013). Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information RFC 7011.

[Coles et al., 2001] Coles, S., Bawa, J., Trenner, L., and Dorazio, P. (2001). *An introduction to statistical modeling of extreme values*, volume 208. Springer.

[Cuevas et al., 2007] Cuevas, A., Febrero, M., and Fraiman, R. (2007). Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics*, 22(3):481–496.

[Davies and Bouldin, 1979] Davies, D. L. and Bouldin, D. W. (1979). A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227.

[García-Dorado et al., 2018] García-Dorado, J. L., Ramos, J., Rodríguez, M., and Aracil, J. (2018). DNS weighted footprints for web browsing analytics. *Journal of Network and Computer Applications*, 111:35–48.

[Gijbels and Nagy, 2017] Gijbels, I. and Nagy, S. (2017). On a general definition of depth for functional data. *Statistical Science*, 32(4):630–639.

[Gupta et al., 2016] Gupta, A., Birkner, R., Canini, M., Feamster, N., Mac-Stoker, C., and Willinger, W. (2016). Network monitoring as a streaming analytics problem. In *Proc. 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pages 106–112, New York, NY, USA. ACM.

[Hartigan and Wong, 1979] Hartigan, J. A. and Wong, M. A. (1979). A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108.

[Hernández and Phillips, 2006] Hernández, J. A. and Phillips, I. W. (2006). Weibull mixture model to characterise end-to-end Internet delay at coarse time-scales. *IEE Proceedings - Communications*, 153:295–304(9).

[Hoffman and McManus, 2018] Hoffman, P. E. and McManus, P. (2018). DNS Queries over HTTPS (DoH). RFC 8484.

[Hohn et al., 2004] Hohn, N., Veitch, D., Papagiannaki, K., and Diot, C. (2004). Bridging Router Performance and Queuing Theory. *SIGMETRICS Perform. Eval. Rev.*, 32(1):355–366.

[Holger et al., 2016] Holger, K., Sevil, D., Manuel, P., Alex, G., Michael, B., Aurora, R., Josep, M., Shuaib, S. M., vanRossem Steven, Wouter, T., and George, X. (2016). DevOps for network function virtualisation: an architectural approach. *Transactions on Emerging Telecommunications Technologies*, 27(9):1206–1215.

[Jalalpour et al., 2019] Jalalpour, E., Ghaznavi, M., Boutaba, R., and Ahmed, T. (2019). Tmas: A traffic monitoring analytics system leveraging machine learning. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 408–414.

[Julián-Moreno et al., 2017] Julián-Moreno, G., López de Vergara, J. E., González, I., de Pedro, L., Royuela-del-Val, J., and Simmross-Wattenberg, F. (2017). Fast parallel α-stable distribution function evaluation and parameter estimation using OpenCL in GPGPUs. *Statistics and Computing*, 27(5):1365–1382.

[Kaufman and Rousseeuw, 1990] Kaufman, L. and Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, USA.

[Kilpi and Norros, 2002] Kilpi, J. and Norros, I. (2002). Testing the gaussian approximation of aggregate traffic. In *Proc. 2nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, pages 49–61, New York, NY, USA. ACM.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, {ICLR} 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[Kiran et al., 2020] Kiran, M., Wang, C., Papadimitriou, G., Mandal, A., and Deelman, E. (2020). Detecting anomalous packets in network transfers: investigations using PCA, autoencoder and isolation forest in TCP. *Machine Learning*, pages 1–17.

[Kirschstein et al., 2016] Kirschstein, T., Liebscher, S., Porzio, G., and Ragozini, G. (2016). Minimum volume peeling: A robust nonparametric estimator of the multivariate mode. *Computational Statistics & Data Analysis*, 93:456 – 468.

[Le and Mikolov, 2014] Le, Q. and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1188–1196, Bejing, China. PMLR.

[Leira et al., 2019] Leira, R., Julián-Moreno, G., González, I., Gómez-Arribas, F. J., and López de Vergara, J. E. (2019). Performance assessment of 40 Gbit/s off-the-shelf network cards for virtual network probes in 5G networks. *Computer Networks*, 152:133–143.

[Liebeherr et al., 2012] Liebeherr, J., Burchard, A., and Ciucu, F. (2012). Delay bounds in communication networks with heavy-tailed and self-similar traffic. *IEEE Transactions on Information Theory*, 58(2):1010–1024.

[Lin et al., 2015] Lin, W. C., Ke, S. W., and Tsai, C. F. (2015). CANN: An intrusion detection system based on combining cluster centers and nearest neighbors. *Knowledge-Based Systems*, 78(1):13–21.

[Liu et al., 2019] Liu, H., Lang, B., Liu, M., and Yan, H. (2019). CNN and RNN based payload classification methods for attack detection. *Knowledge-Based Systems*, 163:332–341.

[López-Pintado and Romo, 2009] López-Pintado, S. and Romo, J. (2009). On the concept of depth for functional data. *Journal of the American Statistical Association*, 104(486):718–734.

[López-Pintado and Romo, 2011] López-Pintado, S. and Romo, J. (2011). A half-region depth for functional data. *Comput. Stat. Data Anal.*, 55(4):1679–1695.

[Mandel, 1984] Mandel, J. (1984). *The Statistical Analysis of Experimental Data*. Dover Publications.

[Mata et al., 2012] Mata, F., García-Dorado, J. L., and Aracil, J. (2012). Detection of traffic changes in large-scale backbone networks: The case of the Spanish academic network. *Computer Networks*, 56(2):686 – 702.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.

[Moradi et al., 2017] Moradi, F., Flinta, C., Johnsson, A., and Meirosu, C. (2017). ConMon: An automated container based network performance monitoring system. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 54–62.

[Morichetta and Mellia, 2019] Morichetta, A. and Mellia, M. (2019). LENTA: Longitudinal Exploration for Network Traffic Analysis from Passive Data. *IEEE Transactions on Network and Service Management*.

[Mouchet et al., 2020] Mouchet, M., Vaton, S., Chonavel, T., Aben, E., and Hertog, J. D. (2020). Large-Scale Characterization and Segmentation of Internet Path Delays With Infinite HMMs. *IEEE Access*, 8:16771–16784.

[Mozilla Foundation, 2020] Mozilla Foundation (2020). Firefox continues push to bring DNS over HTTPS by default for US users.

[Muelas et al., 2015] Muelas, D., Gordo, M., García-Dorado, J. L., and López de Vergara, J. E. (2015). Dictyogram: A statistical approach for the definition and visualization of network flow categories. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 219–227.

[Muelas et al., 2017] Muelas, D., López de Vergara, J. E., Berrendero, J. R., Ramos, J., and Aracil, J. (2017). Facing Network Management Challenges with Functional Data Analysis: Techniques & Opportunities. *Mobile Networks and Applications*, 22(6):1124–1136.

[Nguyen et al., 2019] Nguyen, Q. P., Lim, K. W., Divakaran, D. M., Low, K. H., and Chan, M. C. (2019). GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection. In *2019 IEEE Conference on Communications and Network Security, CNS 2019*, pages 91–99. Institute of Electrical and Electronics Engineers Inc.

[Nieto-Reyes and Battey, 2016] Nieto-Reyes, A. and Battey, H. (2016). A topologically valid definition of depth for functional data. *Statistical Science*, 31(1):61–79.

[Papagiannaki et al., 2003] Papagiannaki, K., Moon, S., Fraleigh, C., Thiran, P., and Diot, C. (2003). Measurement and analysis of single-hop delay on an IP backbone network. *IEEE Journal on Selected Areas in Communications*, 21(6):908–921.

[Perdices, 2018] Perdices, D. (2018). Aplicación de técnicas estadísticas a registros de red para el análisis de tráfico y explotación de datos. Bachelor Thesis.

[Perdices et al., 2018] Perdices, D., Muelas, D., Pedro, L. D., and López de Vergara, J. E. (2018). Network Performance Monitoring with Flexible Models of Multi-Point Passive Measurements. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 1–9.

[Perdices et al., 2019] Perdices, D., Muelas, D., Prieto, I., de Pedro, L., and López de Vergara, J. E. (2019). On the Modeling of Multi-Point RTT Passive Measurements for Network Delay Monitoring. *IEEE Transactions on Network and Service Management*, 16(3):1157–1169.

[Rehurek and Sojka, 2010] Rehurek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

[Rose et al., 2005] Rose, S., Larson, M., Massey, D., Austein, R., and Arends, R. (2005). DNS Security Introduction and Requirements. RFC 4033.

[Rossi and Conan-Guez, 2005] Rossi, F. and Conan-Guez, B. (2005). Functional multi-layer perceptron: a non-linear tool for functional data analysis. *Neural Networks*, 18(1):45 – 60.

[Rossi et al., 2005] Rossi, F., Delannay, N., Conan-Guez, B., and Verleysen, M. (2005). Representation of functional data in neural networks. *Neurocomputing*, 64:183 – 210. Trends in Neurocomputing: 12th European Symposium on Artificial Neural Networks 2004.

[Royuela-del-Val et al., 2017] Royuela-del-Val, J., Simmross-Wattenberg, F., and Alberola-López, C. (2017). libstable: Fast, Parallel, and High-Precision Computation of α-Stable Distributions in R, C/C++, and MATLAB. *Journal of Statistical Software*, 78(i01).

[Samani and Stadler, 2018] Samani, F. S. and Stadler, R. (2018). Predicting Distributions of Service Metrics using Neural Networks. In *14th International Conference on Network and Service Management, CNSM 2018 and Workshops, 1st International Workshop on High-Precision Networks Operations and Control, HiPNet 2018 and 1st Workshop on Segment Routing and Service Function Chaining, SR+SFC 2*, pages 45–53.

[Schwarz, 1978] Schwarz, G. (1978). Estimating the dimension of a model. *Ann. Statist.*, 6(2):461–464.

[Simmross-Wattenberg et al., 2011] Simmross-Wattenberg, F., Asensio-Pérez, J. I., Casaseca-de-la-Higuera, P., Martín-Fernández, M., Dimitriadis, I. A., and Alberola-López, C. (2011). Anomaly detection in network traffic based on statistical inference and alpha-stable modeling. *IEEE Transactions on Dependable and Secure Computing*, 8(4):494–509.

[Sonchack et al., 2018] Sonchack, J., Aviv, A. J., Keller, E., and Smith, J. M. (2018). Turboflow: Information rich flow record generation on commodity switches. In *Proc. 13th EuroSys Conference*, EuroSys '18, pages 11:1–11:16, New York, NY, USA. ACM.

[Tadikamalla, 1980] Tadikamalla, P. R. (1980). A Look at the Burr and Related Distributions. *International Statistical Review / Revue Internationale de Statistique*, 48(3):337–344.

[Trevisan et al., 2017] Trevisan, M., Finamore, A., Mellia, M., Munafo, M., and Rossi, D. (2017). Traffic Analysis with Off-the-Shelf Hardware: Challenges and Lessons Learned. *IEEE Communications Magazine*, 55(3):163–169.

[Vega et al., 2018] Vega, C., Aracil, J., and Magana, E. (2018). KISS Methodologies for Network Management and Anomaly Detection. In *2018 26th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2018*, pages 181–186. Institute of Electrical and Electronics Engineers Inc.

[Wang and Stolfo, 2004] Wang, K. and Stolfo, S. J. (2004). Anomalous payload-based network intrusion detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3224:203–222.

[Wellner et al., 1977] Wellner, J. A. et al. (1977). A Glivenko-Cantelli theorem and strong laws of large numbers for functions of order statistics. *The Annals of Statistics*, 5(3):473–480.

[Xu and Wang, 2008] Xu, K. and Wang, F. (2008). Cooperative monitoring for internet data centers. In *2008 IEEE International Performance, Computing and Communications Conference*, pages 111–118.

[Zuo and Serfling, 2000] Zuo, Y. and Serfling, R. (2000). General notions of statistical depth function. *Annals of statistics*, 28(2):461–482.