UNIVERSITÉ DE STRASBOURG

ÉCOLE DOCTORALE 269

Mathématiques, Sciences de l'Information et de l'Ingénieur

Laboratoire ICube, équipe CSTB

THÈSE présentée par :

Fabio GUIGOU

soutenue le : **18 juin 2019**

pour obtenir le grade de : **Docteur de l'université de Strasbourg**

Discipline/Spécialité : Informatique

The Artificial Immune Ecosystem: a scalable immune-inspired active classifier

An application to streaming time series analysis for network monitoring

THÈSE dirigée par :

M. COLLET Pierre	Professeur, Université de Strasbourg
M. PARREND Pierre	Professeur d'Enseignement Supérieur, ECAM Strasbourg-Europe, HDR

RAPPORTEURS :

M. PASTOR Dominique	Professeur, IMT Atlantique
M. MERELO Juan J.	Professeur, Universidad de Granada

AUTRES MEMBRES DU JURY :

Mme. LEGRAND Véronique	Professeur, Conservatoire national des arts et métiers

Contents

List of Figures

List of Tables

Acknowledgements

This project would have been impossible from the start without the tremendous effort and involvement of many amazing people. First and foremost, I thank my advisors, Pierre Parrend and Pierre Collet. You both helped me stay on track and gave me so much food for thought – way too much so for a single thesis!

My utmost gratitude goes especially to Pierre Parrend, who followed me most closely. Thanks for the encouragements and the priceless advice. Thanks for all the blunders you pointed out and the structure you gave me. Seeing this manuscript grow and eventually get your approval was a joy!

I would like to specially thank Véronique Legrand, Michel Tyburn and Thierry Coussy for the work they put in creating this opportunity for me. You made the collaboration between IPLine and the ICube lab possible and turned a nice idea into a concrete project.

I thank Professors J.J. Merelo and Dominique Pastor for accepting to be my rapporteurs for this thesis and coming all the way to Strasbourg for my defence. Thanks again to Professor Véronique Legrand for also being part of my jury.

Of course, I thank all my family for their constant support and interest, with a special thought for my grandfather, who never missed an opportunity to discuss my progress and read my publications.

Finally, a very special *thank you* to Sophie Menneson, who emotionally put me back together more times than I could count.

None of this would have been possible without *every single one* of you.

Chapter 1

Introduction

1.1 Cloud computing and monitoring

We live in the Cloud era. Our data and applications have left or will soon leave the ubiquitous Personal Computer of the eighties and nineties to migrate into the Cloud. A clear example is that, as of 2018, Microsoft Office 365, the on-line edition of the well-known office suite, was used by 56% of the companies[1], thus exceeding the use of desktop version. This new paradigm in application design – with SaaS[2] replacing desktop applications and Cloud storage replacing hard drives – also implies a new paradigm in network access and connectivity. People and corporations have become connected, 24 hours a day, and a disruption in Internet access can grind a business day to a halt in a matter of seconds. The success of the Cloud model is paralleled by the mobile computing revolution, which was made possible by the rise of Cloud-hosted services.

This revolution enjoyed a wide media coverage, highlighted with iconic statistics. In October 2016, for the first time, mobile platforms took over desktop computer as the first platform to access the Web[3]. Furthermore, it has been predicted in 2016 that by 2020, 67% of corporate IT will be running in the Cloud[4]. Reports show that this prediction was exceeded by 2018, with 77% of businesses running at least one of their applications in the Cloud[5]. This sketches a new computing landscape leaning heavily on mobile devices and Cloud-based platforms – even in the corporate world.

Though the notion of applications and data belonging "in the Cloud" still appears mysterious to some [Dillon et al., 2010], there is actually a precise definition of Cloud computing. According to NIST, "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (*e.g.* networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [Mell et al., 2009]. The same document also defined the three levels of Cloud computing:

[1] https://wire19.com/over-half-of-organizations-have-now-deployed-office-365/

[2] Software-as-a-Service, the new wave of feature-rich Web applications aimed at replacing traditional desktop tools such as word processors, CRM or accounting software...

[3] http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide

[4] https://www.forbes.com/sites/gilpress/2016/11/01/top-10-tech-predictions-for-2017-from-idc

[5] https://www.idg.com/tools-for-marketers/2018-cloud-computing-survey/

- IaaS, Infrastructure-as-a-Service: on-demand provisioning of basic computing and networking building blocks, *i.e.* network segments and virtual machines,
- PaaS, Platform-as-a-Service: on-demand provisioning of a runtime environment on top of which applications can be deployed, and
- SaaS, Software-as-a-Service: on-demand access to applications running in the Cloud, with remote processing and storage, generally through a Web browser.

In other words, IaaS provides users with virtual machines and networks, PaaS with application servers and SaaS with applications; all three are accessed *via* the network. These new ubiquitous access modes put a heavy load on the Internet infrastructure, which has now become more critical than ever. This new criticality makes it all the more important to monitor the network for early signs of failure, congestion, overload or attacks [Alhamazani et al., 2015, Mdini et al., 2017, Ward and Barker, 2014].

Monitoring is the process by which network operators and support teams can be informed in real time of the status and of any potential issue of the network, networking devices, servers and applications they operate. It is a crucial part of the operation of any computing infrastructure of non-trivial size. However, we will see that in spite of the importance of monitoring, especially after the Cloud revolution, solutions in use by hosting companies and Internet Service Providers (ISP) are still rudimentary, and the topic still raises a number of research challenges, partly addressed by a wide community. Before reviewing these challenges and outlining a proposed solution, let us first define monitoring more precisely.

Monitoring can be accomplished by capturing and analysing traffic [Zhou et al., 2018] or traffic metadata [Yuan et al., 2017] or by querying the networking devices for health metrics [Alhamazani et al., 2015, Gupta et al., 2016]. The former are more advanced, but the latter still the most widely used: a simple Google search for "network monitoring tools" typically returns results for Nagios (and its many forks), Cacti, PRTG or Zabbix. These are open source or commercial monitoring suites, all based on active monitoring of networking nodes. *Active* monitoring means polling, *i.e. actively* retrieving information, as opposed to *passive* monitoring where the information is sent to the monitoring nodes as SNMP traps, logs or raw network traffic. Monitoring of the *nodes* refers to the data being generated by the networking devices themselves, as opposed to traffic capture. These different "flavours" of monitoring serve different purposes and become relevant in different contexts: traffic capture is crucial for security analysis and intrusion detection [Gupta et al., 2016], while passive monitoring is well suited to server monitoring, as servers can run local monitoring agents [Anand, 2012]. Monitoring generic networking devices, however, remains driven by polling, *i.e.* active monitoring [Chowdhury et al., 2014].

For ISP and IaaS hosts, the most critical aspect is active monitoring at the nodes – generally referred to as "network monitoring" even though it reaches beyond the scope of the network –, which coincides with the current offering in terms of commercial software. Our personal experience in the ISP and hosting industry points to network monitoring, *i.e.* active monitoring, as the most challenging area for a number of reasons [Ward and Barker, 2014]:

- Most devices in an ISP network can only be monitored actively.
- While security applications have traditionally been heavy on the computational resources, network monitoring is expected to scalable on low-end hardware, *i.e.* to be extremely lightweight in terms of computational demand.

- Some commercial tools, especially in security, apply artificial intelligence to traffic patterns[6]. None, however, attempt to train models on the time series obtained by active monitoring – though the research community acknowledges the need for deeper analysis of these metrics.

This last point – applying high level analyses on monitoring data at scale – is the subject on which we will focus throughout this thesis.

1.2 Challenges of Cloud-scale monitoring

Recent papers cite a number of challenges in network monitoring [Alhamazani et al., 2015, Mdini et al., 2017, Ward and Barker, 2014, Celenk et al., 2008]:

- Use of machine learning on the monitoring data
- Predictive monitoring
- Anomaly detection
- Scalability of the analysis

In essence, the main difficulty associated with network monitoring is the intersection of a large volume of incoming data and the need to process it in real time. This makes it extremely difficult to perform any non-trivial analysis on the monitoring data. Typically, the usual monitoring solutions such as those we cited above only compare values to thresholds to determine whether a device is functioning properly or not. Predictive monitoring, *i.e.* forecasting [Alhamazani et al., 2015], anomaly detection [Mdini et al., 2017] and machine learning [Ward and Barker, 2014], which are viewed as challenges in the monitoring community, all require a significant computational power and dedicated infrastructure [Zhou et al., 2018, Gupta et al., 2016]. Given that the analyses have to be applied at the scale of hundreds of thousands to millions of metrics, even for a relatively small Cloud provider, with a sampling interval between one and five minutes[7], the total computational power available to process a single metric is extremely limited.

The first challenge, namely the use of machine learning, is linked to the scalability problem: lightweight machine learning models are rare, and the typical models for time series, like recurrent neural networks [Greff et al., 2017], are among the most expensive models in the whole field of machine learning. The same goes for anomaly detection and predictive monitoring: scalable algorithms, even when not based on machine learning, tend to be costly, especially when long dependencies exist, *e.g.* the baseline behaviour of the metric is defined over thousands of samples.

The scalability challenge means that a monitoring system capable of analysing millions of time series in real time is fundamentally incompatible with the recent trends in machine learning such as deep learning, which rely on large clusters of dedicated hardware for training and still incur a considerable runtime cost.

[6] See for example Darktrace, a behavioural intrusion detection system based on Bayesian traffic models (`https://www.darktrace.com`)

[7] Default settings for Nagios: one check every 5 minutes, switching to one per minute if a problem is detected.

1.3 Requirements for a monitoring system

Monitoring systems, besides any research problem, have a number of requirements to be considered functional. A system that does not match these requirements will be considered essentially useless. It is important to note that these requirements are not challenges *per se*; instead, they influence the potential solutions to the research problems found in monitoring. Only a solution matching these requirements can be considered valid.

The first requirement is real-time analysis [Elsen et al., 2015, Zhou et al., 2018, Lin et al., 2016]. This requirement is common in monitoring and not limited to *network* monitoring: it also holds, for instance, for security monitoring [Zhang and Paxson, 2000], *i.e.* detecting attacks against an organization. Real-time analysis implies *online* analysis [Wang et al., 2016b]: the samples must be processed as they are received, in a streaming fashion, and not at rest, when all the data are available [Gupta et al., 2016]. Online operation limits the algorithms that can be used on the streaming data, as not all time series analysis models can function this way [Zhou et al., 2018].

The second requirement is autonomy, *i.e.* the ability to operate with the smallest possible level of human control [Liotta et al., 2002]. It is quite obvious that a system that will monitor millions of time series should require as little tweaking as possible, and most importantly no specific tuning *for each time series.* A global sensitivity parameter may be relevant, but no more. This eliminates a number of models that indeed have many parameters that need to be carefully adjusted in order to obtain meaningful results [Keogh et al., 2004].

The third requirement is a low false positive rate [Dastjerdi et al., 2012, Zhang and Paxson, 2000]. False alerts overload network operators, leading to a lack of attention and a higher probability of missing an important alarm. The goal of “advanced” monitoring, *i.e.* monitoring beyond simple user-defined alerting thresholds, is to achieve a higher detection capability while maintaining a low false positive rate [Wang et al., 2016b].

These are functional requirements: they apply to any monitoring system, be it commercial software or academic research. However the challenges listed above (use of machine learning, anomaly detection, failure prediction and scalability) give rise to another set of requirements for the system we propose. While machine learning can only be considered a tool and not a requirement, we consider the other three as necessary parts of an advanced monitoring system:

- *Anomaly detection:* a metric suddenly changing behaviour can be a better indicator of an ongoing incident than a value exceeding a threshold, even if the metric remains within theoretically acceptable bounds [Ward and Barker, 2014].
- *Failure prediction:* warning the operators at the earliest possible signs of an anomaly, *i.e.* before any dangerous behaviour is detected, can prevent incidents before they have an impact [Celenk et al., 2008].
- *Scalability:* arguably also a functional requirement, scalability means that millions of time series should be analysed in real time, on distributed hardware if necessary; the throughput should be in the order of tens of thousands of data points per minute and per server [Gupta et al., 2016].

To summarize, the requirements for our proposed monitoring system are:

- Low CPU and memory footprint, or scalability,

- Real-time analysis,
- Anomaly detection,
- Failure prediction,
- Low configuration overhead, or autonomy,
- Low false alarm rate.

1.4 Monitoring in nature

The starting point of this thesis is that a real-time, distributed, scalable, self-regulated subsystem that monitors the overall system in which it exists, reacts to anomalies and known threats and constantly tries to minimize its monitoring latency while maintaining a low false positive rate already exists in nature. In fact, we argue that this description matches that of the immune system, as it exists in humans and many other animals.

Since the early days of computing, the idea of replicating natural processes to solve complex problems has generated dozens of successful models, which in turn have spawned hundreds of variations. The 1950s already saw the rise of the artificial perceptron [Rosenblatt, 1958] and artificial evolution [Minsky, 1958]; across the years, more natural phenomena were mimicked, simulated and used as metaheuristics: the flocking behaviour of birds [Reynolds, 1987], the path-finding abilities of ants [Dorigo and Gambardella, 1997], the information passing of foraging bees [Pham and Castellani, 2009]… Among these models, the artificial immune systems (AIS) were built around many key concepts and processes of the *biological* immune system put forward by *immunologists*, such as the immune network theory [Jerne, 1974] or the negative selection of self-reactive T cells in the thymus [Forrest et al., 1994].

As many other nature-inspired metaheuristics, AIS come in many flavours: any algorithm inspired by the antibody/antigen reaction, the maturation of lymphocytes, the clonal hypermutation of T- and B-cells, or any other immune process, can be labelled as an artificial immune system [Hart and Timmis, 2008]. This leads to a wide variety of models and applications: many unrelated algorithms and theories have been formulated under the umbrella term of artificial immune systems.

Although the AIS community never designed a single, unified metaheuristic, or even a consistent family of algorithms – in the way the neural network family encompasses shallow as well as deep networks, feedforward as well as recurrent, and many different models of neurons: linear, squashing, gated, spiking… –, the expected properties of these algorithms and of the artificial immune systems in general are very consistent; in fact, they closely follow the perceived properties of biological immune systems. In [Aickelin and Cayzer, 2008], they are summarized as follows:

- *Error tolerance:* a single fault should not have a long-term effect on the behaviour of the immune system.
- *Distribution:* just as the lymphocytes, through the lymphatic system, are distributed throughout the body, an artificial immune system should have the ability to be distributed across many computation nodes.
- *Adaptation:* changes in the “immunized” system should be learnt and followed by the immune system, rather than trigger a catastrophic auto-immune reaction.

- *Self-monitoring:* the immune system itself should be able to detect anomalies in its own behaviour and correct them.

In [Twycross and Aickelin, 2005], the authors express these properties as the ODISS framework, with ODISS standing for:

- *Openness:* the immune system and its immunized host interact in a state of dynamic equilibrium.
- *Diversity:* a vast repertoire of immune cells covers a wide range of possible antigens.
- *Interaction:* the immune system can be modelled as a network of interacting and signalling immune cells.
- *Structure:* the immune system is functionally decomposed into distinct subsystems with different roles.
- *Scale:* the immune system is composed of a large population of simple agents rather than a small population of complex ones.

In a later paper [Twycross and Aickelin, 2007], they describe a slightly different set of properties of the immune system:

- *Specificity:* an antibody targetting an antigen is only sensitive to *that* exact antigen.
- *Diversity:* a vast repertoire of immune cells covers a wide range of possible antigens.
- *Memory:* specific antigens, once encountered, leave a trace in the immune system and trigger a faster and more powerful response on secondary exposure.
- *Tolerance:* the immune system, while remaining capable of reacting against almost any antigen, does not react against the body itself.

In [Greensmith et al., 2010], Greensmith proposes the dendritic cells as a basis for artificial immune systems, citing the following properties:

- *Decentralization:* the immune system does not have a central decision or coordination point; instead, the various immune cells interact locally.
- *Robustness/error tolerance:* errors made by a single individual, *i.e.* a single immune cell, do not propagate through the system and have no real impact on its overall behaviour.

Of course, the core processes of the immune system such as anomaly detection, sometimes formulated as non-self detection, and memory, are generally not stated as properties: they are rather the use case for an AIS, and the natural function of the immune system. Thus, just as we formulated requirements for a monitoring system, we can propose requirements for an artificial immune system:

- *Decentralization/scalability:* the immune system, because it only relies on local interactions, can scale almost indefinitely. In computing terms, it scales horizontally, without single point of failure.

- *Anomaly detection:* unknown objects (vectors, antigens, patterns...) can trigger an immune reaction.
- *Memory:* objects that would not trigger an immune reaction can be made to cause one. This is the inverse of tolerance, and the principle behind vaccination: unknown pathogens causing damage and triggering a late reaction after first exposure are remembered and the secondary response is much more intense and rapid than the first.
- *Tolerance:* objects that would otherwise trigger an immune reaction can be tolerated, *i.e.* the immune reaction can be suppressed.
- *Adaptation:* when the monitored system (the body, a data stream...) changes, the immune system is able to silence the global auto-immune reaction that this change would trigger. Adaptation is similar to tolerance applied along the time axis.

A point often missed in the literature is that, to follow the natural immune system, these properties should be applied in real time: adaptation is meaningless if the whole history of the system/body is known in advance. The immune memory also evolves during the lifetime and cannot be reduced to a mere set of detectors generated at one point in time, *e.g.* during a hypothetical training phase.

The AIS community is unique in that it is extremely close to the immunologists: following the works of Jerne [Jerne, 1974], researchers like Varela and Bersini [Bersini and Varela, 1990, Bersini, 2002, Cutello et al., 2004] have explored both sides of the field, and their models have contributed to the development of artificial immune systems as well as the understanding of the biological immune systems themselves. The works of Greensmith [Greensmith and Aickelin, 2009] on the dendritic cell algorithm are as much a computational device as a model of a biological process. The danger theory put forward by Matzinger [Matzinger, 1994, 2002], an immunologist, has been studied and introduced in the AIS by Aickelin [Aickelin and Cayzer, 2008] only a few years after its inception.

Unfortunately, the AIS community tends to have a pessimistic view of the field, highlighting its inadequacy to real-world problems [Freitas and Timmis, 2003, 2007] and sometimes naïve view of immunity [Matzinger, 2002, Bersini, 2002], at odds with the current biomedical research. The vital questions for the AIS community are: can AIS actually implement the desired properties of immune systems? And can they scale to real-world problems as well as neural networks or evolutionary algorithms do?

A pragmatic view of the issue, introduced in two grounding papers by Freitas and Timmis [Freitas and Timmis, 2003, 2007], is that one should focus on the *properties* of the immune system instead of the *processes*, and follow a problem-oriented approach, with real-world constraints and domain-specific evaluations.

One important proof the AIS community has made over the years is that various facets of biological immunity, and therefore various properties, are best modelled by different algorithms [de Castro and Von Zuben, 2001, Forrest et al., 1997, De Castro and Von Zuben, 2002, Aickelin and Cayzer, 2008], which are discussed in detail in Section 4.1; the logical conclusion is that, in order to verify any set of immune properties, multiple algorithms must work together. This would turn the artificial immune system, from a family of metaheuristics, into a meta-algorithm integrating a number of these metaheuristics, with different AIS-like algorithms modelling, for instance, the innate immune system, the adaptive immune system and the interactions between the two [Greensmith and Aickelin, 2009].

Rather than discussing "meta-metaheuristics", we refer to this approach as the Artificial Immune Ecosystem (AIE), considering that a set of systems in interaction constitutes an *ecosystem*. This "ecosystemic" approach is developed in depth in Chapter 6. In essence, we show that, once we admit that various components – *i.e.* systems, or algorithms – should be used to model different properties of the immune system, the original AIS, as designed in the 1990s and 2000s, are inappropriate tools, and that a set of specialized algorithms is preferable, both in terms of performance – as measured either in CPU and memory cost or in accuracy – and of understanding of the overall ecosystem. Therefore the AIE becomes a framework in which task-specific algorithms can be plugged to provide the desired properties, with the AIE specifying the interactions and communications between these algorithms.

The central argument of this thesis is that the requirements of the artificial immune system are compatible with those of monitoring systems, and that a model compatible with both sets of requirements yields an efficient monitoring system, and at the same time a valid artificial immune system model. The test case for the AIE is the monitoring of a Cloud provider network, which challenges the requirements of artificial immune systems. It entails monitoring hundreds of thousands of time series *(scalability)* in real time in order to detect potential failures *(anomaly detection)*, screen them for false alarms *(tolerance)*, record the failure modes for earlier detection *(memory)* and alert the support team when a failure is detected. These roles map onto the immune properties the following way:

- *Anomaly detection:* unknown patterns trigger an alarm,
- *Memory:* known failure signals trigger an alarm,
- *Tolerance:* known rare patterns do not trigger an alarm,
- *Adaptation:* obsolete signals are purged to prevent false alarms.

These properties can be verified by a number of different algorithms or sets of algorithms. The AIE does not define which algorithm is used for which function, but how the models providing these properties interact. The choice of algorithms is domain-specific, depending on the data. In this thesis, it will be guided by the requirements for monitoring systems that are not directly expressed in immune systems:

- Low CPU and memory footprint,
- Real-time analysis,
- Low configuration overhead,
- Low false alarm rate.

Most importantly, let us remember the wise words of Andrew Watkins and Jon Timmis [Watkins and Timmis, 2004] about their Artificial Immune Recognition System, which are true for any nature-inspired model: *"Immune-**inspired** is the key word. We do not pretend to imply that AIRS directly models any immunological process, but rather AIRS employs some components that can metaphorically relate to some immunological components."* Neither do we claim the AIE to be a biologically plausible model. It is, however, plausible in the world of artificial immune systems, insofar as it is based on the same metaphors, as embodied by the properties defined above.

1.5 Contribution

1.5.1 Objectives

The striking resemblance between monitoring and immune systems, in terms of their goal and some of their properties – or *desirable* properties – is the starting point of this thesis. Our hypothesis is that adding the algorithmic blocks to provide the properties of immune systems to existing monitoring systems can address at least some of the challenges described by the community. We ask the following questions: can an immune system-like algorithmic design solve some of the problems of monitoring systems? In particular, can it reach the required level of accuracy, failure prediction and autonomy to compete with state-of-the-art approaches, while retaining a limited footprint in terms of computational power and memory?

These questions should be read in the context of the requirements expressed for monitoring systems:

- *Accuracy* is measured by the false positive rate and the new failure modes that can be detected with respect to a threshold-based system.
- *Failure prediction* is intended as reaction time *before* an alerting threshold is crossed, *i.e.* before the actual failure occurs.
- *Autonomy* is understood as the ability to function without human intervention, *e.g.* fine parameter tuning.

We show in Chapter 2 that these are the research challenges of the monitoring community. The statistics and machine learning communities bring models that may be suited to answer these challenges, yet remain to be benchmarked against them. The immune inspiration also brings a notion of *memory* that, to the best of our knowledge, can be modelled by many algorithms but has never been formalized as such. Likewise, the notion of *tolerance* has not been explored as such by the machine learning community.

Our objectives are to derive, adapt or design algorithms to model the immune properties described in the previous section. We expect the resulting artificial immune-like system to fulfil the requirements of monitoring systems better than traditional methods, mostly because of the extremely limited resources available per time series. This endeavour raises a number of questions that have not been solved in the literature:

- Can anomaly detection in time series be feasible at the scale of network monitoring and in real time with a small computational budget?
- Can an immune-like form of memory provide failure prediction, or at least, as in the biological immune system, make the reaction to already encountered pathogens, *i.e.* failure modes, faster?
- How can a system designed to *detect* anomalies and failures be made tolerant to rare patterns, the same way the immune system tolerates foreign elements in the body such as the intestinal flora?

The objective of the Artificial Immune Ecosystem is to answer these questions, and it will be challenged along these axes. Note that they reflect the key immune properties that were highlighted before; these properties will be the target of the benchmarks performed on the AIE.

1.5.2 Overview

This thesis brings three main improvements to the state of the art, which are summarized in Section 11.1 of the final chapter. They cover the following topics:

- *Monitoring process and architecture:* the Cloud-Monitor model adds anomaly detection and failure prediction to traditional monitoring systems at a computational overhead that does not hinder scalability.
- *Active learning:* the ALPAGA model uses the output of a binary classifier as a heuristic for active learning, *i.e.* training a model with a human in the loop, which allows for false positive filtering.
- *Anomaly detection:* the SCHEDA models approximate the euclidean anomaly score in periodic time series at an extremely low computational cost. We also show that they outperform the euclidean score in terms of accuracy.

SCHEDA makes Cloud-Monitor scalable, while ALPAGA filters its false positives based on expert feedback. The system thus constructed forms the Artificial Immune Ecosystem, or AIE, and is expected to exhibit the core immune properties of anomaly detection, memory and tolerance.

1.5.3 Outline

This thesis is organized in two parts. Part I reviews the current literature in order to find the right models to build an artificial immune ecosystem. It also identifies the perceived shortcomings of these models and the remaining challenges that will require new algorithms to be designed. This part is outlined as follows:

Chapter 2 reviews the field of network monitoring. It presents the various communities involved in monitoring and the research associated with each specific area: network security, virtual machines and service level monitoring. This review expands the challenges we already discussed, in particular advanced analytics at scale, and presents the current advances in solving these challenges. The literature shows a trend of integrating Big Data tools such as Apache Spark Streaming[8] into the new monitoring frameworks; however, these techniques and tools are not yet applied to the time series obtained by active monitoring. Instead, they are used on flow metadata, which are snapshots of the system at a given time, without context. Machine learning also remains largely unused.

Chapter 3 covers anomaly detection models for time series data. In particular, it focuses on streaming data, online algorithms and scalability. The literature is extremely rich and covers a wide variety of methods, but also shows a number of limitations. The challenges of scalable algorithms that can operate online, in real time, on the type of time series obtained by monitoring systems, are not solved by any current model. In particular, the long-term dependencies observed in monitoring time series defeat auto-regressive models and recurrent neural networks. From this study, we conclude that a new algorithm is required to handle the specific characteristics of the time series obtained in network monitoring, namely the long, multi-scale patterns they exhibit and the potential for anomalies to happen at any of these scales.

[8] `https://spark.apache.org/streaming`

Chapter 4 bridges the domains of monitoring and artificial immune systems. We argue that the requirements of failure prediction in monitoring and memory in immune systems have a large overlap: a failure in a system can be predicted either by detecting an anomaly in a metric of this system or by noticing that the considered metric follows a pattern that has, in the past, led to a failure; both approaches are known as *failure symptom monitoring* [Salfner et al., 2010]. The second definition, however, is extremely similar to the notion of memory in immune systems. In this chapter, we review models that may be applied to the problem of creating an immune repertoire, or a pattern classifier. In particular, the literature shows that instance-based classifiers, *i.e.* entirely data-driven classifiers that do not build models, are the best suited for such a task. The most efficient of them are nearest neighbours classifiers, which are more efficient and more precise than traditional artificial immune systems or rule-based systems such as Learning Classifier Systems (LCS).

Classifiers can not only be supervised or unsupervised, there also exist families of semi-supervised models that use unlabelled test samples as well as labelled ones to predict the categories of new testing samples. Given that the immune memory is the mechanism that enables both the fast secondary reaction of the immune system and its tolerance [Matzinger, 1994], the nearest neighbour classifier implementing the function of immune memory must be able to recognize a failure symptom (memory) as well as an erroneous reaction (tolerance). However, this last function requires erroneous reactions to be somehow detected as such. In the context of monitoring, we propose that the users, *i.e.* the network operators and IT support staff, could tag alerts as true or false alarms. The immune memory classifier would then receive these labels in real time and adjust its behaviour in consequence. This approach of classifier training is called *active learning* [Tong and Chang, 2001, Veeramachaneni et al., 2016]. Chapter 5 investigates the models of active learning and use scenarios. We find, however, that active learning lacks the real-time component in many evaluations, and instead is only compared to supervised learning in terms of the number of labelled samples necessary to reach a similar accuracy to traditional models.

Part II of this thesis describes the proposed AIE model. The AIE is an immune-inspired classifier that fulfils the requirements of monitoring systems. As such, it addresses some of the open problems highlighted in Part I, in particular the challenges of network monitoring. It also proposes new algorithms for active learning and time series anomaly detection, which are required in the model. These contributions as well as the overall AIE are evaluated in the context of Cloud network monitoring and against the requirements of immune systems and monitoring systems. This part is organized as follows:

Chapter 6 proposes a high-level view of the artificial immune ecosystem. It describes the individual subsystems that form the ecosystem itself and their interactions, as well as the properties of each of them. It also presents the immune metaphor on which the AIE is based; in particular, how the AIE models the innate and adaptive immune systems as three different processes that map to the functioning of a monitoring system:

- Innate immunity provides coarse-grained detection of a broad range of pathogens. It maps to the ability of monitoring systems, based on thresholds, *i.e.* domain knowledge, to detect severe, “obvious” malfunctions.
- Acquired immunity provides fine-grained detection of both known and unknown pathogens – though known pathogens are detected faster and trigger a stronger immune response. The detection of unknown pathogens maps to anomaly detection [Greensmith et al., 2005].

> However, to the best of our knowledge, the fast detection of known pathogens has no equivalent in monitoring systems, though it can be compared to the signature database of an anti-virus or intrusion detection system.

Chapter 7 describes a simple AIE model that verifies *some* of the key immune properties, namely anomaly detection and memory. We call this first model Cloud-Monitor [Guigou et al., 2015]. It is evaluated in the context of a monitoring system. We show that the added ability of detecting potential failures based on novelty, *i.e.* new behaviour, and past failures, leads to faster detection times. Though this finding supports our initial hypothesis that immune-inspired models can be the basis of monitoring systems, we also find that, without any feedback loop, the false positive rate becomes so high as to threaten its usability. This is expected, as Cloud-Monitor does not model immune tolerance. It also confirms the importance of this property.

To filter these false alerts, a second block is added to the AIE. It wraps the immune detection system and allows false alerts to be detected and filtered. As such, it provides immune tolerance. Chapter 8 proposes ALPAGA (Actively Learnt Positive Alert GAteway) [Guigou et al., 2017b], a methodology based on the active learning paradigm to use feedback given by experts to distinguish between true and false alarms. ALPAGA is based on the notion that a monitoring alarm can be labelled by the network operators as corresponding to a real incident or a simple false positive, and that the associations of fragments of time series and labels can be used to train a classifier. Said classifier can then filter future alarms and discard those that are predicted to be false.

We benchmark several families of algorithms to determine the best candidates for the role of false alarm filters: neural networks, Bayesian filters, decision trees, nearest neighbour and support vector machines. Given that alarms are a rare phenomenon, with at most one or two occurring per month in typical time series, this classifier must be able to work with very small datasets. Experiments show that in these conditions, instance-based classifiers such as nearest neighbour classifiers perform best in terms of accuracy and computational load. Comparing the accuracy of a monitoring system generating alerts without filter and filtered by ALPAGA, we show that nearest neighbour models and decision trees perform better than the raw monitoring system, and support vector machines, neural networks and Bayesian classifiers perform worse or on par with the unfiltered alerting system. Interestingly enough, this shows that the algorithms best suited to provide immune tolerance are also those that best model immune memory. This fact opens the possibility to use a *single* memory for the secondary reaction and the immune tolerance. Not only does it make the model simpler, it also corresponds to the actual way the biological immune system works: by building a repertoire of memory T- and B-cells.

A crucial process in this monitoring immune system is anomaly detection. The time series obtained by monitoring have a number of properties that can be readily observed by simply plotting them and have been investigated in other fields, such as electrical power usage [Espinoza et al., 2005, Keogh et al., 2004]. Typically, the patterns present in these series are multi-scale, nested and non-linear [De Chaves et al., 2011], with strong periodic daily and weekly structures. As we show in Chapter 3, this type of long-term dependency defeats learning models such as recurrent neural networks and the multiple complex patterns makes random process models, *e.g.* auto-regressive moving average models, unusable.

Chapter 9 describes a set of heuristics that allow anomaly detection to be performed online, in real time, on monitoring time series. We show that these heuristics, which we call SCHEDA for Sampled Causal Heuristics for Euclidean Distance Approximation, are good predictors of the euclidean anomaly score, but can be computed in linear instead of quadratic time. We show that SCHEDA and the euclidean anomaly score, which is the naïve way to search for novelty or outliers – naïve in the sense that its computational cost makes it essentially useless for long time series [Keogh et al., 2005] – perform similarly on monitoring data. SCHEDA, however, is scalable to hundreds of thousands of series on a single server.

Thus Cloud-Monitor provides *memory* for secondary reaction, ALPAGA adds *tolerance* and SCHEDA allows for *scalability* and *anomaly detection*. All these algorithms work *online*, in *real time*. The last property, *autonomy*, is evaluated in the context of the full AIE.

Chapter 10 describes and evaluates the complete AIE integrating the Cloud-Monitor immune detection system, using SCHEDA for anomaly detection, and a nearest neighbour classifier shared between the failure database of Cloud-Monitor (immune memory) and the ALPAGA filter that eliminates false alerts (immune tolerance). The benchmarks are performed against a simple monitoring system, a monitoring system with an ALPAGA filter and Cloud-Monitor alone. The relevant metrics are execution time, false alarm rate and missed alarm rate, with various sensitivity tunings and a variable amount of feedback to train ALPAGA.

We show that the AIE performs significantly better than any combination of its subsystems and maintains a runtime cost compatible with the requirements of monitoring systems. By verifying the property of *autonomy*, it also fulfils all the requirements of artificial immune systems. The results of the experiments on the AIE validate the hypothesis that immune and monitoring systems are compatible, and that the former can solve some of the challenges of the latter. We further claim that by providing a strong use case for an artificial immune system and novel ideas from immunology that were not present in the monitoring community, the AIE makes AIS relevant while building a strong basis for future monitoring suites.

Chapter 11 concludes this work, summarizes its salient contributions and opens up perspectives on further improvements on the AIE model.

Part I

State of the art

In the first part of this thesis, we propose a literature review guided by the requirements for monitoring systems, as we defined them in Section 1.3:

- Low CPU and memory footprint,
- Real-time analysis,
- Anomaly detection,
- Failure prediction,
- Low configuration overhead,
- Low false alarm rate.

Our approach in this thesis is to use the metaphor of the immune system to design a model verifying these requirements. We propose to model failure prediction as a form of immune memory and to consider low false alarm rate as the role of immune tolerance. We showed in Section 1.4 that anomaly detection, memory and tolerance were core processes of immune systems, the remaining requirements that are specific to monitoring systems are low CPU and memory cost, real-time analysis and low configuration overhead. These properties will guide us through the literature to identify the strengths and weaknesses of the current models providing repectively:

- Anomaly detection in time series,
- Memory, and
- Tolerance.

The first chapter of this review is dedicated to the application domain of the AIE: network monitoring. It highlights the current challenges faced by the community, the approaches taken to solve these challenges and the recent trends in the field. Through this literature review, we try to answer the following questions:

- How can anomaly detection scale to millions of time series?
- Which architecture is required to support such scalability?
- How will the scale of the problem influence the design methodology of our model?

The second chapter covers anomaly detection, which is a large class of problems: point anomalies can be detected as points lying outside of a statistical distribution, and structural anomalies, also known as behavioural anomalies, can be generated by normal-looking activity occurring at an abnormal point in time, *e.g.* a regular pattern of activity detected during a week-end [Chandola et al., 2009]. During the last two decades, research has spawned numerous approaches to this problem, with different strengths and weaknesses and different trade-offs. The question we try to answer is: given our requirements and the challenge of scalability, what are the existing models suitable for anomaly detection in a monitoring system, or how should one be designed?

The third chapter considers immune memory. Immune memory models are typically instance-based, *i.e.* based on a set of vectors similar to the training set rather than a model built around it, with negative instances for negative selection [Forrest et al., 1994] and positive examples for clonal expansion [De Castro and Von Zuben, 2002]. Besides these artificial immune systems, we review the most closely related classifiers. The nearest-neighbour classifiers are also directly based on the training samples, without any real model built on top of it; they are the simplest algorithms that can be derived in this model-free paradigm. We also study the literature on condition-action systems, *i.e.* systems learning to perform specific actions based on detected patterns, which share common characteristics with artificial immune systems by matching a representation of an *environment* with a population and, on a positive match, triggering the *action* associated with the matched agent. These models are alternatives to the AIS for multi-agent modelling.

Finally, the fourth chapter focuses on tolerance, and the existing models that can provide it. In particular, we study active learning. Active learning is a way to train a classifier – *any* classifier that can work with small training sets – incrementally. It is generally implemented in one of three ways: by using internal metrics of the classifier, by training a committee of classifiers, or by using an external heuristic unrelated to the classifier. By incrementally training filters through time, it creates a meaningful interaction between the AIE and its users, *i.e.* an interaction that takes place in real time and allows the network operators and support staff to correct the behaviour of the system. The questions we try to answer are:

- Can active learning be used to filter false alarms from a monitoring system?
- If so, which models are the most suitable for time series and monitoring systems, *i.e.* taking into account the scalability challenge?
- Otherwise, how can an active learning model be designed to provide tolerance to an immune monitoring system?

Chapter 2

Domain overview: Network and service monitoring

While this constitutes quite a digression from the academic argument, we must first acknowledge that the phrase “Monitoring sucks” is quite popular in the sysadmin world – for valid reasons. System and network monitoring still relies on old technologies and protocols, mainly ICMP (as embodies by the ubiquitous `ping` command) and SNMP[1]. These protocols are not designed for efficient collection of advanced metrics and are severely limited in any non-trivial scenario. The large ecosystem of tools, from open-source collections of scripts to integrated enterprise products, have seldom scaled to the demand of large service providers in terms of performance, support for decentralized agents, independent operation of datacentres or data quality. A lot of configuration still has to be performed by hand; the monitoring data are stored in lossy formats (see *RRDtool* as a popular round-robin, flat-file database used in monitoring suites such as Cacti, Nagios and many of their forks, in the open-source world; a variant is used in PRTG[2] on the commercial side). While circular buffers implementing subsampling of historical data are great for viewing performance graphs during debug sessions, they are utterly useless if the data are to be used for pattern mining, anomaly detection or forecasting. These systems also fail to provide a centralized, API-enabled view, while at the same time ensuring that network outages between a central server and distributed probes does not compromise data collection. Mostly, probes are unable to operate on their own, and the only way to achieve partition tolerance is to deploy multiple instances in parallel, which implies giving up a unified view of the network. Lastly, behavioural analysis, in the form of anomaly detection or pattern extraction, is completely absent from the industrial monitoring world: a failure is detected when a script returns a value superior – or inferior – to a threshold, with no regard to context.

What led to this state of things, as far as we understand from the plethora of tools and projects, both free and commercial, is the following: the world split between companies *using* IT services, and monitoring *their* infrastructure, which may scale to a few hundred or even a few thousand devices, and companies *providing* IT services and monitoring not only their infrastructure, but also their customers’ – which brings the count somewhere between tens and hundreds of thousands of devices, and millions of metrics. The former have historically used open-source solutions, Nagios being the historic leader, whereas the latter have either

[1] Simple Network Management Protocol, designed in the late 1980s and early 1990s to provide a unified language for device management; almost exclusively used today to poll various metrics for monitoring.

[2] Pässler Router Traffic Grapher

developed their own solutions in-house or purchased the high-end commercial ones, which are typically unaffordable for small and medium businesses. However the increasing dependency of all types of business on their IT infrastructure has led to the emergence of small providers and the increase in the size of the IT service of larger companies, creating a new type of actor: small structures managing big networks. These actors are typically too small for large scale commercial solutions and too big for hand-configured, partially sysadmin-patched, centralized, poorly scalable free solutions. These new companies have new requirements that are yet to be addressed, leaving them with inadequate tools.

On the academic side, it can be argued that the topology of problems that must be addressed in network monitoring is often on the edge between research and engineering, and that the purely scientific topics such as time series analysis are researched in a multitude of domains, from health [Wei et al., 2005] to seismology [Zhu et al., 2018], and seldom get a chance to be considered in the realm of network monitoring, let alone integrated in monitoring solutions. Furthermore, the current trend towards deep and recurrent neural networks, which are heavyweight models, is not well-aligned with the requirements of network monitoring, in particular low computational cost and low configuration overhead [Greff et al., 2017].

The scientific literature has evolved in time, with contributions in the 1990s and 2000s focusing on monitoring as a new field, tackling problems such as polling methods, scheduling and distribution, and most research in the 2010s dealing with the new challenges brought forth by the explosion of the Cloud computing paradigm, both in terms of scale and security. In particular, the rise of SDN[3] [da Silva et al., 2016, de la Cruz et al., 2016] and the concern over distributed attacks [da Silva et al., 2016, Zaalouk et al., 2014] has opened new challenges, while the new Big Data tools have provided new ways of dealing with them [Macit et al., 2016].

This chapter is a review of this broad spectrum of the literature.

2.1 Network monitoring

When researching the literature in the domain of network monitoring, two different understandings of the term emerge: monitoring of the network itself, and of the devices using the network. Monitoring of the network itself entails packet capturing *via* technologies such as OpenFlow or port mirroring. Network security is often the pursued goal, with applications such as intrusion or DoS (Denial-of-Service attacks) detection. This branch of network monitoring keeps expanding as the technologies underpinning today's networks, *e.g.* SDN or SD-WAN[4].

Many research projects have been carried out in this field, such as OrchSec [Zaalouk et al., 2014], which uses OpenFlow as a data provider to detect DoS attacks in real time. Commercial products, while rare, also exist; an example is Darktrace[5], which builds Bayesian models of network traffic in a LAN to detect various sorts of threats propagating from machine to machine, such as viruses, that induce a change in traffic patterns.

[3] Software-Defined Networking, a way of abstracting the hardware and connectivity into elementary functions and flows that are easier to manage.

[4] Software-Defined Wide Area Network, an extension of SDN to inter-site and global networking.

[5] `https://www.darktrace.com`

2.1.1 Early research

The scientific contributions up to the mid-2000s were mostly aligned with the state of the industrial art in terms of technologies used and problems studied: polling used SNMP and ICMP and the problems were about scheduling and distribution. The problem of scalability and monitoring of large networks across different routing domains is addressed in [Hanemann et al., 2005] in the form of a decentralized, hierarchical architecture separating the user interface layer, service layer and measurement layer. The lowest layer, measurement, handles the collection of metrics; the service layer handles the transformation, aggregation and protection of said metrics, and the interface layer allows users to interact with the system. This model decouples collection, analysis and presentation of the metrics and allows a monitoring system to scale horizontally by distributing each function across multiple servers.

In [Brutlag, 2000], as early as 2000, a simple anomaly detection scheme is proposed and integrated in RRDtool, a popular flat-file database used to store time series data. The technique used is based on forecasting: the time series is decomposed using the Holt-Winters method [Brockwell and Davis, 2016] and the prediction error is measured. Whenever that error is out of the confidence interval of the model, an anomaly is detected. This model is of course not perfect: it requires tuning, lacks theoretical grounding and proper evaluation, but accurately reflects the concerns at that specific point in time.

In [Liotta et al., 2002], the problem of statically configured pollers and their inadequacy when dealing with large, dynamic networks is approached in a multi-agent paradigm. Instead of configuring monitoring agents on specific hosts, they propose a mobile agent that can move from host to host and discover the topology of the network. These agents are able to query network informations from the routers, clone themselves and move across the network; therefore, the deployment of new agents and the monitoring of new devices is automatic. While this approach to complex topology and scale is interesting, it assumes that any node on the network is able to become a poller, *i.e.* can run the monitoring agent, which has implications in terms of security and provisioning (the agent needs to be able to connect to a node, install its code there and ensure that all required software libraries are present).

In [Cranor et al., 2003], Gigascope, a streaming database is proposed for real-time query processing. It addresses the limitations of traditional databases or storage back-ends when dealing with high volumes of data that require complex processing by translating queries into optimized task-specific executables that read and write the monitoring data and query results as real-time streams. Such stream processors can be chained for complex tasks or factoring of common subtasks. A SQL-like query language is proposed as the base of the system. Note that today's Hadoop map-reduce [Shvachko et al., 2010] and Apache Spark frameworks share a number of common features with Gigascope, though they are more generic in nature. Hence the problems known today as Big Data were already part of the monitoring landscape in the early 2000s.

2.1.2 Monitoring software-defined networks

Recent trends in monitoring, as far as academic research is concerned, has shifted towards supporting new networking technologies, such as Network Function Virtualization (NFV) [Palmisano et al., 2017], and using monitoring to pilot the network, *e.g.* using traffic engineering [Yuan et al., 2017]. These new axes show a tendency to move away from the technical

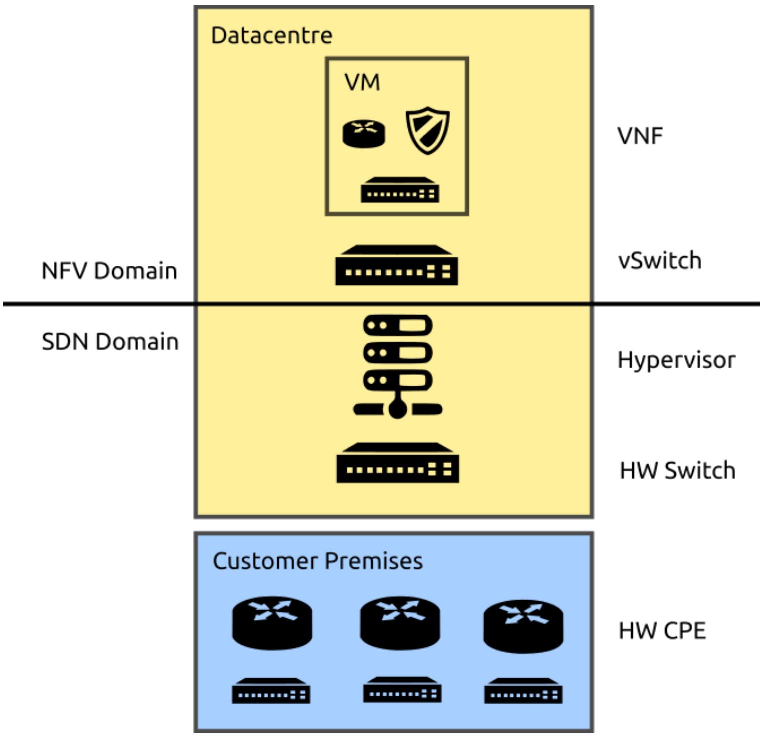


Figure 2.1 – Network Function Virtualization and Software-Defined Networking in ISP datacentres

implementation of monitoring towards a more ecosystemic view, where monitoring is just a subcomponent of a larger, mostly autonomous environment.

This new direction can be partially explained by the ongoing paradigm change in the networking world, with the leading edge of the industry moving away from the traditional proprietary boxes – dedicated routers, switches, firewalls... – and towards fully virtualized network "functions" implemented as virtual machines on x86 or ARM hardware. A function, in this case, is typically a well-defined subset of the functionalities provided by a dedicated appliance: BGP dynamic routing, VLAN tag swapping, flow control based on IP addresses and ports (*i.e.* basic firewalling), but also more advanced features, including some that are harder to implement in hardware: intrusion detection, anti-spam, DDoS protection, anti-virus, but also reporting, real-time dashboards and graphs generation... Figure 2.1 shows the domain of application of NFV and SDN in ISP networks.

NFV goes hand in hand with SDN and SD-WAN, both of which provide a device-independent way of specifying flows across the LAN or WAN, but also of auditing said flows and collect a number of statistics about them. In [Chowdhury et al., 2014], the authors propose PayLess, a monitoring solution based on OpenFlow, one of the main components of SDN, to poll performance metrics from switches and expose them through a REST[6] API. Applications of this framework include customer billing, intrusion detection, quality of service management and failure detection.

The introduction of SDN and OpenFlow for metrics collection has also led the community to design better approaches to traffic engineering: when anomalies are detected in a particular flow

[6] REpresentational State Transfer, HTTP-based API that uses hypertext semantics to operate on arbitrary objects

due to a failure of one of the switches in its path, new mechanisms are proposed to recompute a path and reroute the affected flows. In [de la Cruz et al., 2016], a failure detection module is proposed for the open source SDN controller OpenNetMon, which forces the instant rerouting of flows passing through a failing switch. Here failure detection is performed by individual flow sampling.

2.1.3 Advanced monitoring

While this integration work is necessary to monitor networks based on newer technologies, it does not solve the anomaly detection problem, and especially the issue of anomaly detection *at scale, i.e.* on every single metric polled from each device. Failure metrics in PayLess are only metrics polled from the devices, not results of data analysis.

One implementation of anomaly detection on monitoring metrics is CFEngine by Burgess *et al.*. In [Burgess, 2002, 2006], they propose approaches for adaptive computer systems that include time series anomaly detection. In particular, they model time as a cylinder, and univariate time series as two-dimensional, each point having a n coordinate modelling the period to which it belongs and a τ coordinate corresponding to its offset within a period. Thus time is expressed as:

$$t = nP + \tau \tag{2.1}$$

This representation allows a rolling estimate of the mean and standard deviation to be computed at constant τ, thus modelling the variation interval of each point within a period. Typically, a period is chosen to represent the equivalent of a week. These estimates are updated in such as way as to discount older samples and keep the model up-to-date with changes in the time series.

While this approach is robust and lightweight, it has to maintain a model (mean and standard deviation) of each time series over a full period. More importantly, it only exploits a *single* period in the time series. To deal with events happenning at shorter time scales, additional tests have to be used, such as the Leap-Detection Test (LDT) [Begnum and Burgess, 2007], which builds a *local* model of the time series and detects changes at the scale of about 10 time steps. Using two different models also means having two different, independent sources of alerting in a monitoring system.

One further step toward advances analyses is the NetQRE framework introduced in [Yuan et al., 2017]. NetQRE provides a high-level language to manipulate metrics and execute actions based on the results of these manipulations. While the intended use of this language is to specify policies that are to be executed in real-time, in particular with respect to quality of service and traffic engineering, it implements a generic business rule engine for real-time data streams of network events. From a scalability standpoint, it can monitor the traffic of a 10 Gbps line on a single server, which is typical of small datacentres scenarios. However, the fact that it operates on raw packets and has no notion of historical data beyond the concept of a stream – typically a TCP session, beginning with a SYN packet and ending with a FIN – scopes it outside our domain of application.

In [Zhou et al., 2018], the problem of high-volume, high-velocity analysis of streaming data is explored using the Apache Spark Streaming framework. Spark Streaming allows computations to be distributed across a cluster of nodes and carried out in real time. A cluster of 5 servers was able to handle 150,000 packets per second, or 500 Mbps of traffic. However, once again, the proposed architecture is only adapted to short-term analysis of TCP streams and does not attempt to implement anomaly detection. Instead, it is a proof-of-concept of streaming analysis on a state-of-the-art distributed analysis platform. A similar approach is taken in [Gupta et al., 2016] to detect DNS reflection attacks.

The problem of anomaly detection is tackled in [Mdini et al., 2017] for VoIP call logs. The proposed Watchmen Anomaly Detection algorithm uses the periodicity of the monitoring data to compute an anomaly score on the time series and raise alarms based on that score. This score is the euclidean distance between the time series and a model maintained daily. It is also adapted to low-resources environments and uses compressed representations of time series such as PCA and SAX, which are described in detail in Section 3.3.1.

One interesting aspect of the current research on monitoring is the distribution of the computational load associated with time series analysis. The authors of NetQRE [Yuan et al., 2017] as well as D-StreaMon [Palmisano et al., 2017] emphasize the distributed nature of their frameworks, both of which rely on SDN for metrics collection. Decentralization of NetFlow traffic analysis is also explored in [Elsen et al., 2015].

Along with network management technologies, monitoring techniques constantly improve thanks to the development of new computing paradigms. In [da Silva et al., 2016], ATLANTIC (Anomaly deTection and machine LeArNing Traffic classifICation for software-defined networks), a new combined traffic anomaly detection and mitigation scheme based on SDN, is proposed. It uses SDN to obtain a fine-grained view of the network at the flow level, *i.e.* to visualize the traffic between individual applications running on different machines, and to reconfigure the network on-the-fly when anomalies are detected. The typical use of ATLANTIC is to detect attacks, *e.g.* DDoS (Distributed Denial-of-Service) attacks, or spurious traffic such as port scans, and to automatically block then.

ATLANTIC uses a 2-step detection, with a *lightweight phase* detecting potentially anomalous flows and a *heavyweight phase* performing more advanced analysis to confirm the anomaly and trigger the corrective actions. In particular, the lightweight phase uses the entropy of the metrics, which is simple to compute but imprecise, to detect anomaly candidates, and the heavyweight phase applies an SVM on these candidates to accept or reject them. This is a typical case where a heuristic, here the entropy, is used to trigger an alert which is then filtered by a secondary machine learning process before any actual action is performed. Here the metrics used are traffic counters rather than performance data from the devices, and the anomalies are computed on the statistical distribution of these counters rather than the evolution of their actual value in time: it operates on histograms instead of time series.

An alternative to flow sampling is log analysis. In [Macit et al., 2016], the logs of all the networking devices in a large MPLS[7] network are collected and processed in real time to detect anomalies. Log lines are aggregated by device and type, generating a $logID$. The number of lines for this $logID$ is counted over a running window of 5 minutes; if it exceeds a threshold, *e.g.* three standard deviations above the mean, an alarm is emitted. The estimates of mean and variance are computed using an exponentially weighted moving average. The test implementation matches the scale of a real provider network and uses standard Big Data technologies such as Apache Kafka for persistent cacheing, Hazelcast for in-memory indexing and Apache Storm

[7] MultiProtocol Label Switching, standard protocol for carrier networks

for running analytics. The same Apache Storm framework is used in [Wang et al., 2016b] to combine multiple online anomaly detection methods for network flows:

- Statistical distribution of the number of concurrent flows,
- Statistical distribution of the traffic volume,
- Correlation between number of flows and traffic volume.

On the other end of the spectrum, small networks of low-power nodes, *i.e.* IoT (Internet of Things) networks, face different challenges, with resource constraints beings much more stringent than in regular core networks. In [Mayzaud et al., 2016], the low-power routing protocol RPL[8] is used as the only source of information for flow analysis in IoT networks. Interestingly, the approach bears many commonalities with more traditional scenarios: a parallel network of monitoring nodes passively receives the traffic and analyses the routing messages. This is made possible by the wireless nature of IoT networks: any node in range can receive the messages sent over the radio links. The routing path to which RPL converges allows the monitoring nodes to detect failing nodes when routes are recomputed.

Though SDN is not our direct target, the advanced analyses carried out on SDN flows can be replicated on time series acquired using simpler techniques. In particular, the idea of using different levels of analysis, with lightweight processing for screening and more advanced (and costly) models to confirm anomalies, as used in ATLANTIC [da Silva et al., 2016], is a promising way to achieve scalability.

2.2 Cloud monitoring

A different aspect of monitoring that appears in current research is Cloud monitoring, *i.e.* monitoring of virtual machines and applications hosted in the Cloud. Cloud monitoring is distinct from network monitoring for two main reasons:

- Virtual machine monitoring is a wildly richer domain than simple device monitoring: as a VM runs a general-purpose, multitasking operating system, it is possible to run a monitoring agent directly on it to capture more relevant metrics than cannot be accessed *via* simple SNMP.
- Each host can actually run analyses locally instead of relying on a central server. For instance, in [Lin et al., 2016], agents are used for automatic root cause analysis and use the available real-time CPU load and network usage information.
- Cloud monitoring is dynamic: new hosts can spawn at any time to respond to a change in load, without any human intervention [Alhamazani et al., 2015].

This branch of the monitoring research is also more concerned with architecture than tools and processes, because the latter are no longer limitations. Our interest here is to extract the common elements, both in terms of challenges and methods, between Cloud and network monitoring, in particular the monitoring processes, requirements and architecture.

[8] Routing Protocol for Low-power Lossy Networks, an IETF protocol designed specifically for IoT networks and IPv6

2.2.1 Processes of Cloud monitoring

In [Ward and Barker, 2014], the authors present a comprehensive overview of Cloud monitoring, both in terms of processes and challenges. In particular, they conceptualize monitoring as a 3-step process:

- Collection: metrics are polled using either SNMP or standard tools (*e.g.* `df` for disk space, `top` for CPU and RAM usage, globally and per process, `uptime` to detect reboots...),
- Analysis: the collected metrics undergo a number of post-processing steps, which entail, from the simplest to the most complex: graphing, thresholding (*i.e.* comparing values to pre-defined alerting thresholds and raising alarms if necessary) and behavioural analysis. The latter, because of the more complex algorithms involved, is also by far the heaviest on CPU and is seldom used in network or Cloud monitoring,
- Decision: based on the results of previous analyses, actions can be automatically taken, though the authors acknowledge the fact that almost no current monitoring system implements such a step. Automatic VM respawn is a typical response to a failure in the Cloud.

These processes are essentially the same for network monitoring.

2.2.2 Requirements for Cloud monitoring

The same paper also proposes a comprehensive list of requirements for Cloud monitoring:

- Scalability: the system should have no bottleneck or single point of failure.
- Fault tolerance: the system should be able to differentiate between normal termination of a VM and VM failure and should not be impacted by widespread failure of the VMs it monitors.
- Autonomy: deployment should be automatic and self-healing.
- Multiple granularities: collection of data should vary depending on the circumstances to simplify troubleshooting; in particular, the collected metrics and the sampling rate should vary as needed, with sampling rates increasing when incidents are detected and decreasing for values that only change slowly in time.
- Comprehensiveness: the system should be able to monitor any platform.
- Real-time analysis: the collection should induce as little latency as possible.

Another requirement is Cloud sensitivity: the fact that a monitoring system should be aware of the costs associated with VM migration, the latency induced by data exchange across different providers; it is also implied that the monitoring servers should move to follow the VMs under monitoring. We excluded this particular requirement from the main list because it applies to Cloud *customers* only, while all others also apply to *providers*. Once again, what applies to Cloud monitoring holds true for network monitoring, and the same requirements apply.

2.2.3 Cloud monitoring architecture

In [De Chaves et al., 2011], [Alhamazani et al., 2015] and [Anand, 2012], the problem of the design of monitoring agents and monitoring architecture is tackled. While the authors of Cloud-Monitor [Anand, 2012] stick to the traditional centralized architecture, they insist on the importance of self-healing in monitors and the possibility to react to failure in other ways than alerting a human; they fail, however, to provide any real-world action that may be taken in any realistic failure scenario.

The authors of PCMONS (Private Cloud MONitoring System) [De Chaves et al., 2011] choose to offload the problem of collection and scheduling to a time-proven solution, Nagios, and to build an orchestration layer on top of it. This layer is designed to provide the required functionalities, in particular the automatic update of the monitoring system when new VMs are spawned and the injection of monitoring scripts into VMs.

The authors of [Alhamazani et al., 2015] propose a more abstract and thorough overview of different types of architectures that can be used when deploying monitoring systems. In particular, they compare centralized architectures, which, though simple to understand and to manage, lack scalability and can easily become single points of failure, with decentralized architectures. They propose three models of decentralized architectures:

- Structured peer-to-peer: a central index server keeping track of all the nodes is still required, but the nodes are otherwise independent,
- Unstructured peer-to-peer: the index is distributed throughout the nodes, eliminating the need for a central repository,
- Hybrid peer-to-peer: special nodes called “super-peers” can act as index servers for subsets of the network, creating an unstructured archipelago of structured islands.

We can correlate these different architectures with the notion of “intelligence” of the monitoring nodes. A “smart” node can fulfil a number of functions but requires more resources, while a “dumb” node would only collect metrics and forward them with little to no processing, thus requiring a lot less computational power. In the language of network monitoring:

- A centralized architecture means that a single supernode needs to concentrate the metrics collection for the whole network, the analysis and the reaction, which is simply impossible for large networks.
- A structured architecture means a central node coordinates many distributed probes but is still responsible for the analysis and reaction: all nodes but one are dumb. This solves the problem of network access, but not that of scalability: the single smart node still needs to carry out all the computations and, if it fails, takes down the entire system.
- An unstructured distributed architecture means that each node is able to collect, process and react, *i.e.* all nodes are smart and no set of node failure (short of the simultaneous failure of *all* nodes) can cause a complete breakdown of monitoring. This is also the most costly architecture, as each node needs the computational power to perform whatever analysis is necessary.
- A hybrid distributed architecture means that most nodes are dumb, but they report to a number of smart nodes, which allows the system to scale horizontally and maintain

a sufficient level of redundancy (*i.e.* the probability that the smart nodes will all fail at once can be made arbitrarily small) without allocating large amounts of resources to all the nodes. Note that the structured and unstructured architectures are only the two edge cases of hybrid architectures.

The most advantageous approach, in terms of simplicity and robustness, is the unstructured architecture. However, it implies that the data processing enabling the alerting must remain computationally inexpensive: large central nodes are not available for number crunching.

2.3 Service level monitoring

As applications and servers move away from local computer rooms into remote datacentres, two side effects appear: one the one hand, the IT team is no longer responsible for the uptime and usability of the Cloud-hosted services; the provider is; on the other hand, the path from the client (*e.g.* a desktop computer) to the application server is now much longer and involves WAN links instead of simple LAN connections. This in turn implies that a failure of the Internet access, be it optical fibre, xDSL or 4G, also makes the applications unreachable and can bring a business day to a brutal stop instantly.

To mitigate this risk, Cloud customers negotiate Service Level Agreements (SLAs) with their providers. SLAs are contracts that guarantee a number of quality of service clauses, *e.g.* minimal availability of the applications, maximal packet loss, latency and jitter on a link... They also include financial compensations in case these agreements are not met. This in turn forces the providers to set up SLA monitoring systems that track these quality metrics across time.

Note that the concept of SLA predates Cloud computing by a significant margin: in 2002, research was already concerned with SLA monitoring in the context of Web services – which are arguably the ancestors of SaaS[9]. In [Molina-Jimenez et al., 2004], the problem of allowing a third party to monitor the network on the behalf of the ISP[10] and the customer is addressed in the context of network QoS (Quality of Service). In [Sahai et al., 2002], a formal specification of SLA is proposed with the goal of automating SLA monitoring. The specification language can be compiled into instructions for a Service Level Monitoring engine, specifying which metrics are to be collected, how often, and where (client-side or server-side). While the authors only focus on the SLA of SOAP[11]-based Web services, the ability to specify SLAs using a formal language is extremely powerful and is still not implemented in today's monitoring systems.

2.4 Open challenges

Though network monitoring has been commonly deployed for over 20 years, there are still challenges when it comes to any non-trivial mode of operation, *i.e.* anything more complex than polling values and comparing them to a threshold.

A domain-specific issue, partly addressed in [Sahai et al., 2002] by a domain-specific language expressing SLA, but also in [Yuan et al., 2017] with policy specifications, is that of describing the expected behaviour of the monitored systems. A formal language for describing SLAs is a good

[9] Software-as-a-Service, one of the prominent Cloud paradigms

[10] Internet Service Provider

[11] Simple Object Access Protocol, a Web service paradigm mostly superseded by REST since the 2010s

starting point, but it is not sufficient to capture generic domain knowledge, which is the domain of expert systems. Domain-specific languages designed to model SLAs as well as critical values for specific metrics and pattern rules (*e.g.* the consumed bandwidth for a given link should not remain constant for more than 10 minutes, otherwise it indicates a link saturation) are still an open problem.

The hardest problem in monitoring is that of behavioural analysis and anomaly detection based on time series collected using various protocols, as first described in [Thottan and Ji, 2003] and studied since then using distributed computing frameworks [da Silva et al., 2016, Lin et al., 2016, Macit et al., 2016]. While most of the research relies on anomaly detection to detect intrusions [Garcia-Teodoro et al., 2009, Bhuyan et al., 2014], our goal is to use the same type of techniques for failure detection based on performance metrics, and to avoid relying on a central processing cluster (*e.g.* an Apache Spark cluster) to detect the anomalies.

The challenges we will address in this thesis are the following:

- Using behavioural analysis and predictive monitoring in real time, while
- Keeping the computational cost of such detection acceptably low, and
- Avoid the high false positive rates that are common in monitoring.

Chapter 3

Detection: Anomalies in time series

In this chapter, we present a literature review on a central topic of this thesis: anomaly detection in time series. Anomaly detection, along with memory and tolerance, is one of the key immune processes the Artificial Immune Ecosystem (AIE) should implement, along with tolerance and memory. The simplest forms of anomaly detection are distance-based methods, where any time series subsequence with a high enough distance to its closest subsequence is considered an anomaly; the most complex methods employ recurrent neural networks to learn precise models of the time series and can require an extremely large computational power to run [Chandola et al., 2009].

Let us first define what a time series is. In its most generic form, a time series T of size N is an ordered collection of N samples of the form $T_i = (t, x)$, x being the actual value and t the timestamp at which it was sampled. Alternatively, we can define $T = (\tau, X)$, with $\tau = t_1, t_2, ..., t_N$ the series of timestamps and $X = x_1, x_2, ..., x_N$ the associated data [Cranor et al., 2003].

It is generally assumed that the sampling is regular, *i.e.* :

$$\forall i \in 1, 2, ..., N-1, \tau_{i+1} - \tau_i = \Delta_t \text{ is a fixed, known quantity.} \tag{3.1}$$

In this case, we usually use the notation $T = T_1, T_2, ..., T_N$ where T_i is the actual sampled value, and discard the timestamps. As monitoring uses regular polling and, even in the case of slight irregularities, time series databases natively resample the data to correct for it[1], this approach will be used throughout this thesis.

This review is particularly concerned with the following aspects of the existing algorithms:

- *Low cost:* the algorithm should be able to process new data fast enough to scale to network monitoring problems (tens of thousands of data points per minute per server),
- *Long-term dependencies:* the algorithm should be able to detect long patterns (*i.e.* multiple thousands of time steps),
- *Real time:* the algorithm should process a new data point immediately, without buffering,
- *Configuration-free:* the algorithm should not require an expert to tune it.

[1] RRDtool, often used in legacy monitoring applications, was the first time series database to systematically introduce resampling; more modern databases such as InfluxDB (`https://www.influxdata.com/time-series-platform/influxdb`) use it too.

3.1 Challenges of time series

One of the main reasons why time series analysis is seemingly difficult and requires special models and methods, while generic classifiers perform well across very different problem spaces, is that time series are a very special data type. While they appear quite natural to the human mind, from a mathematical and statistical perspective, they deviate from a number of fundamental assumptions. This, in turn, makes many basic tasks of machine learning or statistical inference much harder than they usually are. Let us highlight some of the difficulties when working with time series [Chandola et al., 2009]:

- If the entire time series is considered as a vector, then its dimensionality is extremely high and its dimensions are highly correlated. This is a problem for machine learning algorithms that expect input vectors consisting of a few independent dimensions such as support vector machines or naïve Bayesian classifiers.
- If subsequences of the time series are considered instead, then they still show a high *internal* correlation, but also a high correlation with each other, depending on how they are extracted. This typically defeats any type of clustering algorithm applied directly on subsequences extracted by a sliding window [Keogh and Lin, 2005].
- As subsequences are generally not aligned in any meaningful way, any particular dimension changes meaning over time: the i^{th} point of a subsequence is nothing more than a point at a particular index in an arbitrarily sliced time series. Many classification and regression – *e.g.* user profiling [Raghuram et al., 2016] – tasks are carried out on "static" features, where the i^{th} dimension of the input vector can be the height of an image, or the number of white pixels, or *any* feature – but, most importantly, remains the *same* feature in all the dataset.

Furthermore, the way time series data are acquired can induce additional difficulties:

- As the data are sampled from a running process, the entire time series is not available by the time an analysis is performed on it. This is the difference between *online* algorithms and *offline*, or data-at-rest, algorithms. We focus here on *online* algorithms [Salfner et al., 2010].
- Generally, the most interesting part of a time series, at least for online analysis, is the most recent. However, by definition, the last few points of a time series have a left hand side neighbourhood but not a right hand side one, *i.e.* their past is known but not their future. This, in turn, either constrains the design of algorithms or introduces lag: when a new point is added to the time series, the analysis can run on a point at some distance in the past. This means that any conclusion obtained on a point is related to an event that occurred some time before. This phenomenon occurs in algorithms that use a *lead window* [Wei et al., 2005], *i.e.* compare what happened *before* a given point in time and what happened *after*. The lead window represents the samples on the right hand side, *i.e.* in the future, required to perform the analysis on the considered data point.

Note that not all these constraints or difficulties are relevant in all contexts. For network monitoring, however, online operation is naturally a requirement, and lag a problem. Simply put, it is more useful to detect a fault as it happens than the day – or even the week – after.

Other factors often come into play in time series analysis and relate to the long-term versus short-term behaviour of the series. Short-term behaviour is generally what is being analysed, especially in anomaly detection. It is the scale at which the series can display interesting features. By contrast, long-term phenomena include seasonality (repeating patterns at low frequency) and concept drift (slow shifts in the mean of the series). These are generally considered as noise and are removed in some way when they interfere with the short-term analysis.

In [Chandola et al., 2009], the authors describe in detail the various contexts in which anomaly detection can be performed and the various techniques deployed to achieve it. They identify, among others, the following challenges associated with anomaly detection:

- Normal behaviour is hard to define. In particular, creating a baseline profile that summarizes all possible normal patterns in the series is very difficult.
- The boundary between normal and abnormal behaviour is often ill-defined.
- The normal behaviour often changes over time, so that the baseline needs to be redefined.
- Labelled data are rare.
- Anomalies can easily be "washed out" in noise and confused with it, which yields a high rate of false positives and false negatives.
- The very concept of an anomaly changes across application domains, which makes anomaly detection techniques non-trivial to port from one domain to another.

These difficulties and challenges are added to the basic problems associated with time series as a data type.

If we summarize these issues, we find that time series anomaly detection in the context of network supervision leads us to deal with:

- Incomplete information (unavailable future data)
- Ill-defined anomalies, often confused with noise, leading to high false positive rate in the monitoring context (see Section 2.4)
- Changing baseline of the system
- Lack of labelled data

From this point, two radically different approaches emerge. On the one hand, statistical methods will try to detect and remove long-term effects, build a model of the short-term variations and match data with statistical distributions. Broadly speaking, these methods deterministically and procedurally analyse the series as a stochastic process and, in the case of anomaly detection, mark outliers that are not generated by this process.

Statistical methods may operate on the raw time series data or translate it into an intermediary representation. Auto-regressive and distance-based methods typically use the raw samples and will be reviewed in Section 3.2. Many other algorithms have been built around symbolic approaches such as SAX (Symbolic Aggregate approXimation) [Lin et al., 2003] that convert a real-values time series into a text stream. We review them in Section 3.3.

On the other hand, learning systems consider a time series as a sensory input from an environment that can be changing, respond to different laws at different times, and display various patterns at different scales. Machine learning methods have also been applied to time series anomaly detection, each with its own set of strengths and weaknesses; recurrent neural networks are among the most well-known [Greff et al., 2017], as we will see in Section 3.4.

These methods differ from statistical approaches in that they take a holistic approach to the problem at hand. Where a statistical method may require a preprocessing step to remove long-term components, then a filter to remove noise, then an auto-regressive model, and eventually a prediction error computation, machine learning frameworks rely on generic internal mechanisms to work out the features of the environment in which they evolve and the appropriate reactions. They are also stochastic: the initial parameters of the model are chosen at random and not hand-picked, then converge towards an optimum. Typically, neural networks have their weights randomly initialized [Elman, 1990].

3.2 Statistics on raw data

Many anomaly detection methods work on the raw time series data. They fall into two categories: distance-based methods compare analysis windows, or subsequences, and generate at each point a value deduced from these comparisons. Auto-regressive methods, on the other hand, model the random process that generated the observed values and use this model to predict the next sample; the prediction error can be used for anomaly detection.

3.2.1 Distance-based methods

A simple definition of anomaly can be derived based on the euclidean distance [Keogh et al., 2005]: a subsequence of a series whose distance to its most similar subsequence is high can be considered an anomaly. “High” in this context should be understood as “above a certain threshold”. Then we can define an anomaly score A for a time series T of size N, computed for subsequences of length n thus:

$$A_i = min_{j=1}^{N-n} \sqrt{\sum_{k=1}^{n} (T_{i+k} - T_{j+k})^2} \tag{3.2}$$

The number of euclidean distances necessary to fully compute A is $O(N^2)$, which means that the computational cost of such a task becomes rapidly high for long time series. In [Keogh et al., 2005], the authors propose HotSAX, a heuristic to speed up the discovery of the ***most*** anomalous subsequence in T, ***i.e.*** $max_{i=0}^{N-n} A_i$. This method is described in detail in paragraph 3.3.6.

Here, the size of the observation window can be crucial: a short window is highly sensitive to noise, and a long window can fail to detect subtle anomalies. In [Chuah and Fu, 2007], an adaptive window is proposed for ECG signals, which typically contain very similar spike-like signals, some of which, if anomalous, can help diagnose cardio-vascular diseases. In this paper, the window is scaled to capture exactly one heartbeat, and all windows are scaled to the same length before distances are computed. While effective, this method depends on a number of characteristics of the signal: the presence of clearly identifiable spikes over a low enough noise floor, and the absence of any other parasitic signal or pattern.

In [Rakthanmanon et al., 2012], pattern mining is performed under the Dynamic Time Warping (DTW) distance, an extension of the euclidean distance that allows comparison of vectors of different lengths or misaligned. The DTW “warps” both signals to align mismatching

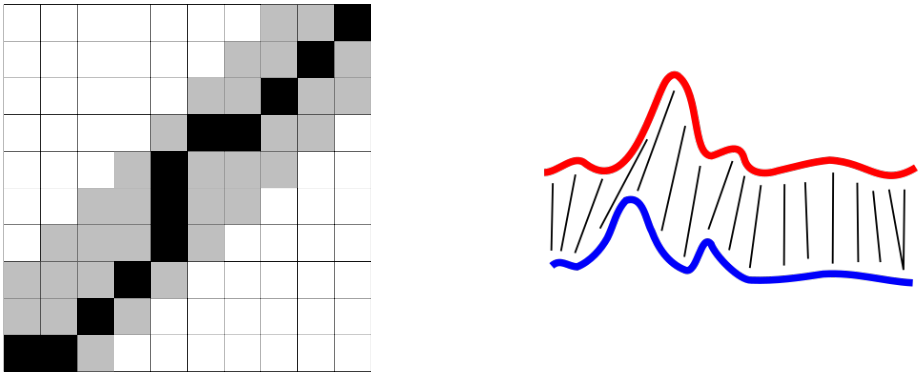


Figure 3.1 – Left: DTW matrix, with optimal warp path in black and maximal deviation in gray. Right: DTW matching between two sequences with warping path marked as black dashes.

parts. Figure 3.1 summarizes the notion of "warping path". For two time series T_1 and T_2, each point of T_1 corresponds to one or more points of T_2, and conversely. The euclidean distance is equivalent to a perfectly diagonal warping path: each sample of T_1 corresponds to exactly one sample of T_2. To limit the computational cost, the warping process in the DTW is limited in its deviation from the diagonal, so that a single point of T_1 can match at most a certain number of points of T_2.

In [Yeh et al., 2016], the notion of *join profile* is described: the matrix of all distances between each possible pair of subsequences in the series. This matrix allows the nearest neighbour of each subsequence to be queried rapidly by a simple array search. Building the matrix still has a quadratic complexity, both in time and space.

In [Lee et al., 2014], a Kullback-Leibler distance is proposed, based on the Kullback-Leibler divergence or relative entropy. This distance is used to cluster time series based on the statistical distribution of forecast errors.

The Kullback-Leibler divergence between two discrete statistical distributions P and Q is defined as:

$$KLD(P||Q) = \sum_{i} P(i) log \frac{P(i)}{Q(i)} \tag{3.3}$$

It estimates the amount of information necessary to encode a sample of P given the knowledge of Q. Note that this divergence is asymmetrical: $KLD(P||Q) \neq KLD(Q||P)$. To overcome this asymmetry, the authors define a Kullback-Leibler distance:

$$KLD_{avg}(P, Q) = \frac{1}{2}(KLD(P||Q) + KLD(Q||P)) \tag{3.4}$$

The KLD distance has some advantages over the euclidean distance; in particular, the KLD distance between two distributions with the same standard deviation but a different mean remains low. This makes it an efficient tool for comparing time series that may undergo linear or affine scaling (*i.e.* $T_1 = \alpha T_2 + \beta$) or time series of different lengths. This is demonstrated in [Barz et al., 2018], where the KLD is used to compare the distribution of each subsection to that of the rest of the time series, *i.e.* to *one single distribution*, which has a linear complexity. The euclidean anomaly score, in contrast, requires each subsequence to be compared to *every other subsequence*, which incurs a quadratic complexity. While the KLD distance is less computationally demanding, it also requires long subsequences, as the distribution of values, at least in the monitoring context, differs significantly between an hour-long, a day-long and a month-long window. Finally, the KLD is an entropy measure: as such, it is concerned with the *values* contained in a subsequence, but not the *order* in which they appear, thus discarding half the information in the time series (the timestamps, as defined in the chapter's introduction).

With regards to the criteria presented in this chapter's introduction, we can evaluate the family of distance-based methods:

- *Low cost:* not the strong point of distance-based methods, which tend to be computationally expensive.
- *Long-term dependencies:* distance-based methods are model-free, thus do not learn dependencies at all; this is therefore not a limitation.
- *Real time:* the euclidean and KLD nearest neighbour search operate in real time.
- *Configuration-free:* distance-based anomaly detection generates, for each subsequence, an anomaly score that needs to be thresholded, thus a detection threshold and a window size are required.

3.2.2 Auto-regressive methods

Approaches based on statistical modelling are the earliest developed in the time series analysis toolkit: they were already in use in the 1940s [Orcutt and Irwin, 1948]. Auto-Regressive Moving Average (ARMA) models use the previous values (auto-regressive) and prediction errors (moving average) to predict the next values of a random process. Along with their derivatives (*e.g.* ARIMA introducing an Integration term, SARIMA dealing with Seasonality, SARIMAX modelling eXogeneous variables), they have been used as early as the 1950s [Durbin, 1960] to derive models of time series, in particular for forecasting. The ARMA model is summarized in Equation 3.5. The time series X_t is described as a combination of a constant term c, a linear combination of its past p values (auto-regressive term), a linear combination of its past q error values (moving average term) and an error term.

$$X_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \tag{3.5}$$

Here p and q are the user-set parameters of the model; they must be carefully tuned based on the autocorrelation of the series and the correlogram of the error [Commandeur and Koopman, 2007]. The coefficients ϕ_i and θ_i are determined during the training phase of the model.

Auto-regressive methods are popular for time series forecasting; however, they have strong requirement for stationarity. Concept drift (aperiodic slow variation of the average of a time series) and seasonality (periodic changes in the time series, *i.e.* very low-frequency components) have to be removed before the ARMA model is estimated. A widely used method is the Holt-Winters decomposition [Brockwell and Davis, 2016], which separates a time series into a trend, seasonality and remainder, where the remainder is expected to be close to stationarity, and therefore eligible to ARMA fitting and forecasting. An alternative is to fit the trend and seasonal components into the auto-regressive model, yielding models such as SARIMA (Seasonal Auto-Regressive Integrated Moving Average), where integration removes the linear trend.

An example of the application of auto-regressive modelling to anomaly detection in networks is available in [Celenk et al., 2008], where an ARMA-based model for network traffic prediction is proposed. The goal of the research is to predict anomalies before they happen based on a signal decomposition between normal and abnormal traffic. Metrics are collected using network monitoring software, then filtered to remove noise and modelled as an ARMA process. The forecast error is then used to detect an anomaly. While the model proposed in [Celenk et al., 2008] is shown to work on short-term anomalies, it is unclear how well it captures regularities and long-term patterns in the input signal, and whether it is able to pinpoint structural anomalies as well as short term ones.

The limitation of auto-regressive methods lies in the model itself: the type of decomposition it uses is adapted to short-term correlations in a signal, not long-term dependencies and regularities. In order for ARMA and its derivatives to perform as intended, the signal must first be filtered to remove long-term autocorrelation, or low-frequency components. Seasonal models, *e.g.* SARIMA, have the built-in assumption that the period is exactly known in advance and does not change; they do not include the possibility of *multiple* patterns occurring at different scales, *e.g.* daily and weekly.

Considering our earlier criteria, ARMA and its derivatives can be thus be qualified:

- *Low cost:* these models are extremely lightweight, as long as the lags are only a few time steps.
- *Long-term dependencies:* while extensions exist to deal with a *single* periodic pattern, auto-regressive models struggle with long-term correlations.
- *Real time:* auto-regressive models can predict a value at each time step and compare it to the real acquired data, in real time.
- *Configuration-free:* the forecasting error is a form of anomaly score, thus requires a threshold, and the order of the auto-regressive process must be specified.

3.3 Symbolic methods

Alternative methods for time series analysis were developed in the early 2000s by the data mining community for different tasks:

- Time series indexing: finding instances of a particular pattern in constant time,
- Similarity search: finding all subsequences closer to one subsequence than a given threshold,
- Clustering: grouping similar subsequences together,
- Rule mining: determining that one pattern is always followed by another pattern.

These methods are often based on alternative representations of time series: instead of a long vector of real numbers, the series is quantized into a smaller vector of symbols from a discrete alphabet. This lowers the computational cost required to run algorithms on the series with respect to the original representation. Such representations also allow new classes of algorithms to be used, in particular algorithms designed to operate on strings such as compression algorithms [Senin and Malinchik, 2013] and similarity-preserving hashes [Wei et al., 2005]. These algorithms are generally lightweight. However, they can also reduce the precision of the analysis and introduce additional lag.

One interesting point to note here is that, at their fundamental level, many time series analyses are almost equivalent. Let us consider clustering, motif detection, anomaly detection, rule mining and forecasting: it is not obvious that they are, in fact, five different problems altogether. How different is a motif from a cluster? A rule is generally just a motif split in half [Shokoohi-Yekta et al., 2015]. An anomaly is, by certain definitions, a sample that differs significantly from its forecast value [Celenk et al., 2008]. By others, it is a subsequence that does not match any motif [Radhakrishna et al., 2016]. Thus an algorithm designed for motif extraction can, for instance, be reused with little tweaking for anomaly detection. A typical case is the Sequitur grammar generation of a SAX-encoded time series, which we will review in paragraph 3.3.6. In [Senin et al., 2014], it is used for data visualization; in [Senin et al., 2015], it is used as a basis for an anomaly detection algorithm and, in [Wang et al., 2016a], for pattern mining. This is why, in this section, we go further than simply anomaly detection algorithms and extend our review to other time series classification tasks.

In the following, we review specifically anomaly detection algorithms based on symbolic representations of time series. We begin by reviewing the most prominent representations, then we show how they can be applied to relevant problems:

- Indexing: As part of our effort to implement the immune memory and tolerance properties in the AIE, a query-by-content strategy may allow for fast comparison of time series signatures.
- Clustering: Famously, Eamonn Keogh discovered in 2005 that "clustering of time series subsequences is meaningless" [Keogh and Lin, 2005]: the algorithms used to deal with time series, at that time, were designed to operate on *feature* vectors, and unable to bridge the gap between the two input types. However, while clustering of (unaligned) subsequences is meaningless, clustering of *full* time series may allow a signatures or a rules database to be shared within a cluster of time series. We study this further in Chapter 8.
- Motif and rule mining: While motif study is not directly related to our research, both techniques can be applied to anomaly detection.
- Anomaly detection: This is the most important immune property we research here.

3.3.1 Symbolic representations

Since its inception in 2003 [Lin et al., 2003], the Symbolic Aggregate approXimation (SAX) has become the *de facto* standard for symbolic representations of time series. SAX builds a compact representation of a time series by sliding a window along the time series and slicing this window into short segments that are represented by a single symbol drawn from a small alphabet. Thus each subsequence is represented by a single word. This process directly stems from the need that appeared in the early 2000s for faster similarity search with respect to the euclidean distance [Keogh et al., 2001]. It is equivalent to a heavy subsampling without low-pass filtering followed by a vector quantization. Typically, a sliding window is decimated, *i.e.* subsampled without prior filtering, by a factor of 10, and a SAX word is about 10 symbols in length; hence the usual *window size* is about 100 points, and *symbol size* about 10. The quantization generally uses an alphabet only 3 or 4 symbols, *i.e.* with a *cardinality* of 3 or 4. Symbol size, cardinality and

Table 3.1 – Minimal distances between SAX symbols

	a	**b**	**c**	**d**
a	0	0	0.67	1.34
b	0	0	0	0.67
c	0.67	0	0	0
d	1.34	0.67	0	0

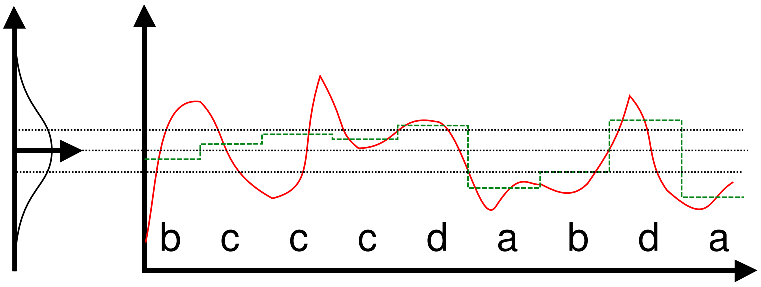


Figure 3.2 – SAX encoding of a time series (red) into the word 'bcccdabda'. The piecewise approximation is shown in green and the normal distribution quartiles in black.

window size are the only parameters for SAX encoding. These settings are consistently used across the literature [Lin et al., 2003, Wei et al., 2005, Senin et al., 2015]. Before quantization, each window is normalized to a mean of zero and a standard deviation of one. This process is illustrated in Figure 3.2. Algorithm 1 describes the process of encoding a single subsequence into a SAX *word*; Algorithm 2 shows how a time series is converted into its SAX representation.

Besides a compact representation of time series, SAX also provides a distance metric that can be computed between two SAX-encoded time series. This *lower-bound distance* represents the minimal euclidean distance between any two time series that would have generated the two considered SAX sequences. It is defined as:

$$MINDIST(\hat{T_1}, \hat{T_2}) = \sqrt{\frac{n}{w}}\sqrt{\sum_{i=1}^{w}(dist(\hat{T_{1,i}}, \hat{T_{2,i}}))} \tag{3.6}$$

where n is the symbol size and w the window size. $dist(\hat{T_{1,i}}, \hat{T_{2,i}})$ represents the minimal distance between a single symbol of series T_1 and a single symbol of series T_2. The $dist$ function is a lookup table computed from the Gaussian quartile function; an example is given in Table 3.1 for an alphabet of size $\alpha = 4$, *i.e.* with 4 different possible symbols, or 16 different distances. The *minimal* distance between two SAX words, then, is simply the sum of the pairwise distances between their symbols, as given by the lookup table.

While other representations exist, benchmarks give them no clear advantage over SAX in the general case [Sant'Anna and Wickström, 2011], and these representations are conceptually and computationally more complex than SAX. Two prominent representations are the Aligned Cluster Analysis (ACA) [Zhou et al., 2008] and Persist [Mörchen et al., 2005]. Contrary to SAX, both are application-specific.

ACA is a dual temporal segmentation and distance minimization model for time series that was introduced for human motion sequence analysis. The one-dimensional time series is

Algorithm 1 SAX encoding of a single word

```
procedure SAXEncode
input:
    $S = [s_1, s_2, ..., s_N]$                ▷ real-valued time series subsequence
    $\alpha = int$                            ▷ cardinality (typical values: 3 or 4)
    $n = int$                                 ▷ symbol size (typical values: 8 to 10)
output:
    $s = string$                              ▷ SAX representation
algo:
    ▷ GQF is the Gaussian quartile function
    $breakpoints = GQF(1/(\alpha-1), 2/(\alpha-1), ..., (\alpha-2)/(\alpha-1))$
    ▷ (for $\alpha = 3$, $breakpoints = [-0.43, 0.43]$; for $\alpha = 4$, $breakpoints = [-0.67, 0, 0.67]$)
    $S = (S - mean(S))/std(S)$
    $S_{aggr} = [sum(s_1, s_2, ..., s_n)/n, sum(s_{n+1}, s_{n+2}, ..., s_{2n})/n, ...]$
    $s = quantize(S_{aggr}, breakpoints)$
```

Algorithm 2 SAX encoding of a full series [Lin et al., 2003]

```
procedure SAX
input:
    $S = [s_1, s_2, ..., s_N]$                ▷ real-valued time series
    $\alpha = int$                            ▷ cardinality
    $n = int$                                 ▷ symbol size
    $w = int$                                 ▷ sliding window size (feature size, usually user-specified)
output:
    $s' = [s'_1, s'_2, ..., s'_{N-w}]$        ▷ list of SAX words
algo:
    $s' = []$
    ▷ sliding window extraction
    $windows = [[s_1, s_2, ..., s_w], [s_2, s_3, ..., s_{w+1}], ..., [s_{N-w}, s_{N-w+1}, ..., s_N]]$
    for $window \in windows$ do
        $s' << SAXEncode(window, \alpha, n)$
```

segmented into slices in such a way that these slices form compact clusters under the Dynamic Time Warping distance (see Section 3.2.1).

Persist, on the other hand, builds a representation in which long runs of the same symbol are as frequent as possible; the breakpoints between symbols are computed to maximize symbol persistence, without subsampling in the time domain. This representation is part of a project to analyse muscle activation patterns.

3.3.2 Indexing and similarity search

One very interesting feature of symbolic representations is the possibility to build indexes of time series for query-by-content, *i.e.* to retrieve subsequences similar to a particular example. While a real-numbered representation would never yield perfect equalities between subsequences, a string-like representation can produce the same output for close enough inputs and, as such, serve as a similarity-preserving hash. SAX-based time series indexing was introduced with iSAX [Shieh and Keogh, 2008, Camerra et al., 2010, 2014] and has been used in many applications such as entomology, DNA mining, image querying [Camerra et al., 2010, 2014] or diagnosis of anomalous electrocardiograms [Shieh and Keogh, 2008].

An iSAX index means that a subsequence can be retrieved from a database in an very short time: depending on the implementation, a hash would require $O(1)$ operations and a prefix tree $O(w)$ for a SAX word size of w. In any case, the lookup time does not depend on the size of the database or the length of the series; only on the size of the SAX representation of a subsequence. Typically, a SAX word contains around 10 symbols [Lin et al., 2003], which makes any lookup operation very cheap.

Since the original iSAX paper, many improvements have been suggested to increase the efficiency of the index, allowing billions of time series to be indexed and queried for tasks such as similarity search, motif discovery or anomaly detection [Camerra et al., 2014]. These improvements mostly use database techniques, *e.g.* index building and update strategies, to reduce disk seeks, memory usage and unnecessary computations.

Indexing, in the context of massive time series monitoring, is of the highest relevance because it makes pattern querying possible, and thus allows “signatures” to be used to characterize the behaviour of time series [Keogh et al., 2001]. For instance, a failure mode of a device can be recorded as the time series subsequence measured just before the incident, encoded into a SAX word and put into an index structure that will be checked whenever a new measure is received. Just as a signature in an anti-virus or intrusion detection system, the signature of a failure mode can be derived and checked against in real time.

From a SAX – or any symbolic – representation, higher level operations can be derived. In [Wei et al., 2005], the frequency of groups of symbols (*e.g.* the number of occurrences of the substrings “aaa”, “aab”, ...) is used to build a fixed-size profile of a time series. Changes in this profile are then used to detect anomalies. However, a fixed-size representation of a time series of arbitrary length that summarizes its characteristics can be more generally regarded as a similarity-preserving hash, which is an important tool in many applications such as neighbour search in high dimensionality data [Lin et al., 2010, Masci et al., 2013], file fragments retrieval [Breitinger and Baier, 2012] and, generally, generating lightweight similarity scores for use in large datasets [Shen et al., 2018].

Similarity search can have multiple applications depending on the context:

- In a single time series, evolution of the hash of subsequences is a way of detecting novelty, or anomalies [Wei et al., 2005].
- Across different time series, the hashes of the series can be used to measure similarity between the series and a distance metric for clustering or anomaly detection [Wei et al., 2005].

Fast similarity search can readily be used for efficient nearest neighbour retrieval, *e.g.* for a k-NN classifier [Malinowski et al., 2013].

3.3.3 Clustering

Time series clustering can entail two different processes: clustering of a set of entire, short time series, or clustering of subsequences of a single, long time series [Chandola et al., 2009]. Keogh warns us in [Keogh and Lin, 2005] about naïvely clustering subsequences extracted by a sliding window, which is by *definition* unable to extract any information. Indeed, a model built from sliding windows ends up only averaging noise over time, as the salient features of the time series are successively learnt at all possible offsets of the window.

Clustering of sets of time series has been carried out successfully using variants of the traditional clustering algorithms such as k-means combined with signal processing methods such as the Fourier transform or wavelet decomposition [Lin et al., 2004], with high accuracy and performance. SAX-based hierarchical clustering is also demonstrated to perform well [Lin et al., 2003].

In the case of multiple time series clustering, the most important factor in the final clustering quality is the distance metric. SAX allows for very simple (*i.e.* without multiplications), yet accurate, distance computations, which in turn yield lower CPU costs and comparable, or slightly better, clustering quality as the raw data [Lin et al., 2003]. This is provided by the lower-bound distance discussed in Section 3.3.1.

The most difficult problem, however, is clustering of full time series – not subsequences –, in part because it is ill-posed and is actually equivalent to motif discovery [Patel et al., 2002].

3.3.4 Motif discovery

Motif discovery in time series is a prominent tool in data mining. It is the opposite of anomaly detection: a search for often reoccurring subsequences, *i.e. normality* detection. Motif discovery is another way to view the problem of time series subsequences clustering; it overcomes the "meaninglessness" problem discovered by Keogh [Keogh and Lin, 2005] by operating on the raw time series, not on subsequences extracted by a sliding window.

Essentially, motifs are frequently occurring patterns, and motif discovery refers to the automatic, unsupervised extraction of such patterns. This is different from query-by-content or similarity search, the search of patterns close to a user-submitted query under a certain distance metric, as described in Section 3.3.2. The need for fast, scalable mining algorithms has already been stated [Rakthanmanon et al., 2012], where the authors claim that "similarity search is *the* bottleneck for virtually all time series data mining algorithms".

In the following paragraphs, we present some motif discovery algorithms based on symbolic representations of the time series, which are thus supposedly computationally efficient [Lin et al., 2007]. Note that the extracted motifs can also be used to efficiently classify time series into clusters [Wang et al., 2016a] by applying the clustering algorithms on the motifs only instead of the full time series.

Motif tracking algorithm The Motif Tracking algorithm [Wilson et al., 2008], shown in Algorithm 3, is an attempt to bring the Artificial Immune Systems (AIS) class of algorithms that operates almost only on strings into the field of time series processing. More specifically, it is an adaptation of the clonal expansion algorithm (CLONALG [De Castro and Von Zuben, 2002]). The SAX representation is used to produce the antibodies/antigens. It uses a subsequence length equal to the word size, therefore each subsequence is represented by a single symbol. Trackers are used as memory cells; they contain the string representation of a subsequence and a match count. Contrary to CLONALG though, Motif Tracking is deterministic: the initial population is not randomly generated, and the alphabet size is small enough that all possible mutated sequences are enumerated.

Algorithm 3 The motif tracking algorithm

```
procedure MotifTracking
input:
    $T = [t_1, t_2, ..., t_N]$                    ▷ first differential of time series
    $\alpha = int$                                ▷ alphabet size
    $s = 1$                                       ▷ subsequence length
    $r = float$                                   ▷ matching threshold (Euclidean distance)
output:
    $M = (m_1, m_2, ..., m_N)$                    ▷ set of motifs
algo:
    ▷ Trackers are SAX alphabet symbols with associated scores and indexes of occurrences
    $trackers = ([a, 0, ], [b, 0, ], [c, 0, ], ...)$
    $symbols = [u_1, u_2, ..., u_{N-s}] = SAX(T, \alpha, s, s)$
    $trackers\_change = 0$
    $l = 1$                                       ▷ motif length
    repeat
        for $i = 1$ to $N - s - l$ do
            for all $tracker, score, indexes \in trackers$ do
                if $[u_i, u_{i+1}, ..., u_{i+l}] = tracker$ then
                    if $\forall j \in indexes, dist([t_i, t_{i+1}, ..., t_{i+s}], [t_j, t_{j+1}, ..., t_{j+s}])/s < r$ then
                        $score = score + 1$
                        $indexes = indexes \cup i$
        for all $tracker, score \in trackers$ do
            if $score < 2$ then
                Remove $tracker$ from $trackers$
            else
                for all $symbol \in alphabet$ do
                    Add $[tracker + symbol, 0]$ to $trackers$
        $s+ = 1$
    until $trackers\_change = 0$
    $M = trackers$
```

This work is the first, to the best of our knowledge, to bring together AIS and symbolic time series representations. However, it needs to perform an unspecified number of passes on the data before converging (*i.e.* when $trackers\_change = 0$). In case long patterns exist, this raises a scalability issue, as it requires as many passes on the data as the length of the longest motif (in the SAX representation).

Approximate Motif Mining The Approximate Motif Mining algorithm (AMM) presented in [Ferreira et al., 2006], as described in Algorithm 4, proposes a number of innovations: it uses aggregative clustering to detect motifs and Pearson's r correlation coefficient, ***i.e.*** the measure of statistical significance, instead of euclidean distance to match potential motif occurrences. Matches are encoded as 2-clusters that are later merged and extended. Aggregative clustering is a way of building clusters by incrementally merging similar samples instead of defining inter-cluster boundaries.

Algorithm 4 Approximate Motif Mining

```
procedure AggregativeClusteringMotif
input:
    $T = [t_1, t_2, ..., t_N]$                                  ▷ time series
    $R_{min} = float$                                          ▷ similarity threshold
    $\alpha, word, window$                                     ▷ SAX parameters
output:
    $M = (m_1, m_2, ..., m_N)$                                 ▷ set of motifs
algo:
    $D = SAX(T, \alpha, word, window)$
    $clusters = \{\}$
    for $i = 1$ to $N$ do
        for $j = i + window$ to $N - s$ do
            if $|r(D_i, D_j)| \geq R_{min}$ then
                $clusters = clusters \cup < i, j >$
    repeat
        $updates = 0$
        for all $X = [x_1, x_2, ..., x_{|X|}] \in clusters$ do
            for all $Y = [y_1, y_2, ..., y_{|Y|}] \in clusters \setminus X$ do
                if $\forall x \in X \forall y \in Y |r(D_x, D_y)| \geq R_{min}$ then
                    $clusters = clusters \setminus X \setminus Y \cup < x_1, x_2, ..., y_1, y_2, ... >$
                    $updates+ = 1$
    until $updates = 0$
```

The original algorithm proposes a way to extend the motif instances; however, our use of SAX sliding windows makes this step inconvenient and mostly useless: the motif length is an initial parameter. In this case, it is also possible to use SAX's lower-bound distance or Hamming distance as a similarity measure to compute $|r(D_i, D_j)|$ in the cluster aggregation step.

3.3.5 Rule mining

Rule mining is an extension of motif discovery where a motif can be broken into two distinct parts: the ***antecedent*** and the ***consequent***. The rule then states that whenever the antecedent occurrs, it is followed by the consequent [Shokoohi-Yekta et al., 2015]. This can be particularly useful for failure detection if a rule can be determined such that when a particular sequence of values is measured (antecedent), then the next sequence (consequent) will be one that corresponds to a known failure mode [Salvador and Chan, 2005].

The algorithm proposed in [Shokoohi-Yekta et al., 2015] is based on the Minimal Description Length (MDL): a motif is split into an antecedent and a consequent in such way that the total number of bits to describe all the occurrences of the motif (*i.e.* the combined length of all the antecedent and corrected consequents) is minimized. This is in response to the obvious problem that arises when splitting a motif into antecedent and consequent: finding the optimal split point. The MDL can be formalized as follows [Shokoohi-Yekta et al., 2015]:

Let D be a set of data. Then D can be encoded in by finite alphabet of symbols such as the binary alphabet. The required number of symbols is $L(D)$: the description length of D. However, a naïve encoding is rarely optimal. If D is a text stream, the ASCII encoding using 8 bits per character does not yield the most compact description of D. Instead, a set of hypotheses $\mathcal{H}$ can be made about D, such as "whenever the string S_1 occurs, it is followed by S_2". These hypotheses also require a certain number of symbols to be described, thus each hypothesis H of $\mathcal{H}$ has a description length $L(H)$. Using $\mathcal{H}$, the data D can now be encoded in a more compact way by only coding the prediction errors of $\mathcal{H}$. The description length of D is now:

$$L(D) = L(\mathcal{H}) + L(D|\mathcal{H}) \tag{3.7}$$

If the binary alphabet is used, then the symbols are bits, and we can compute the bit save of $\mathcal{H}$ as:

$$bitsave(\mathcal{H}) = -L(\mathcal{H}) + \sum_{H \in \mathcal{H}} \Big(L(D) - L(D|H) \Big) \tag{3.8}$$

The algorithm itself generates rules of the form "$S_A \to S_C$", where S_A is the antecedent string and S_C is the consequent string, optimizing the overall bit save.

One limitation of the proposed method is that it does not verify that the antecedent is always followed by the consequent, nor that the consequent is preceeded by the antecedent. Rule mining, as used in [Shokoohi-Yekta et al., 2015], is mostly a motif-encoding scheme, while [Salvador and Chan, 2005] effectively proposes a time series segmentation method, with rules determining transitions from a segment to another.

3.3.6 Anomaly detection

The simple, discrete representation provided by SAX allows for high performance anomaly detection based on algorithms designed to work on textual data. The most prominent domain from which data mining algorithms can be reused is bioinformatics: genomic and proteomic sequences bear a high resemblance to time series data – but with a finite, small alphabet instead of decimal numbers. Since about 2005, a number of bioinformatics algorithms and methods have been used on SAX-encoded time series: the Chaos Game representation [Wei et al., 2005] was designed to give a simple, visual outline of an entire genome in a small picture; applied on SAX-encoded data, it is used for anomaly detection based on the difference between two of these pictures. The Sequitur grammar induction algorithm [Nevill-Manning and Witten, 1997] was originally designed for motif extraction in genomic and proteomic sequences; when used on the SAX representation of a time series, it allows one to perform motif mining and anomaly detection [Wei et al., 2005].

HotSAX HotSAX [Keogh et al., 2005] is not an anomaly detection algorithm *per se*. Instead, it is a heuristic based on the SAX representation of a time series to accelerate the brute-force algorithm. The "obvious" way to detect an anomaly is to search for the subsequence with the highest distance to any other subsequence, ***i.e.*** finding $argmax_i(min_j(dist(x_i, x_j))), |i-j| > n$ over the set x of all sliding windows of n points extracted from the time series t. This search involves a quadratic number of distance computations and therefore is not suitable for more than a few hundred points. Note that self-matches are excluded: the nearest neighbor of a subsequence is only searched among other subsequences having no point in common with it. The naïve, unoptimized implementation of the most anomalous subsequence search (MASS) under the euclidean distance is shown in Algorithm 5.

Algorithm 5 MASS, brute force implementation

```
procedure Brute force discovery
input:
    $T = [t_1, t_2, ..., t_N]$                                  ▷ time series
    $n = int$                                                   ▷ window size
output:
    $dist = float$          ▷ max Euclidean distance between any 2 subsequences
    $loc = int$                   ▷ location of most anomalous subsequence
algo:
    $dist = 0$
    $loc = null$
    for $p = 1$ to $N$ do[0]
        $nndist = inf$
        for $q = 1$ to $N - n + 1$ do
            if $|p - q| \leq n$ then
                next
            if $dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}]) < nndist$ then
                $nndist = dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}])$
        if $nndist > dist$ then
            $dist = nndist$
            $loc = p$
```

However, changing the order in which distances are computed, while not changing the end result, may speed up the search by multiple orders of magnitude (while retaining a complexity of $O(N^2)$) by allowing early terminations of the loops. Ideally, the loops should unroll so that:

- The most anomalous subsequences are processed first,
- For each subsequence, the distances to the nearest neighbours are computed first.

This can be achieved by using heuristics to order the subsequences, as shown in Algorithm 6. The differences between the two versions are highlighted in red.

Using the SAX representation, the outer and inner heuristics index the time series by its symbolic approximation (see Section 3.3.2). The key assumptions of HotSAX are that rare SAX words correspond to rarely occurring subsequences, and that two similar sequences will have a similar SAX representation. Therefore, the outer heuristic first searches the most anomalous subsequence among thoses represented by the rarest SAX word, and the inner heuristic searches any subsequence's neighbours among other subsequences represented by the same word as shown in Algorithm 7.

Algorithm 6 MASS, HotSAX heuristics

```
procedure HotSAX
input:
    $T = [t_1, t_2, ..., t_N]$                                   ▷ time series
    $n = int$                                                    ▷ window size
output:
    $dist = float$            ▷ max Euclidean distance between any 2 subsequences
    $loc = int$                       ▷ location of most anomalous subsequence
algo:
    $dist = 0$
    $loc = null$
    for $p = 1$ to $N$ order by OuterHeuristic do
        $nndist = inf$
        for $q = 1$ to $N - n + 1$ order by InnerHeuristic do
            if $|p - q| \leq n$ then
                next
            if $dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}]) < nndist$ then
                $nndist = dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}])$
            if $dist([t_p, t_{p+1}, ..., t_{p+n-1}], [t_q, t_{q+1}, ..., t_{q+n-1}]) < dist$ then
                break                                  ▷ Early loop termination
if $nndist > dist$ then
    $dist = nndist$
    $loc = p$
```

Let us summarize the characteristics of HotSAX:

- *Low cost:* while less expensive than the naïve euclidean nearest neighbour search, HotSAX remains a heavyweight method.
- *Long-term dependencies:* just as other distance-based methods, HotSAX is not influenced by the time lag between nearest neighbours.
- *Real time:* HotSAX operates on data at rest, the opposite of real time.
- *Configuration-free:* a window size has to be specified, along with the SAX parameters; however the latter generally use the same set of default values [Lin et al., 2007], thus only the window size matters.

SAX/Sequitur The Sequitur algorithm proposed by Nevill-Manning in 1997 [Nevill-Manning and Witten, 1997] is a dictionary-based string compression algorithm designed to study protein structures. It uses the concepts of symbol, rule and digram to build a compact representation of its input data. In Sequitur vocabulary, a symbol is either an input token (*e.g.* a single character, or byte) or a token representing a rule; a rule is a symbol standing for a digram; and a digram is a pair of symbols. Therefore, in a string “abc”, “a”, “b” and “c” are symbols and “ab” and “bc” are digrams. A rule will have the form “A = ab”, meaning that a string “Ac” can be read as “abc”. With such a rule, “A” and “c” are symbols and “Ac” is a digram that can be itself part of another rule.

Algorithm 7 HotSAX heuristics for anomaly detection

```
procedure OuterHeuristic
input:
    $T = [t_1, t_2, ..., t_N]$                                  ▷ time series
    $n = int$                                                   ▷ window size
output:
    $P = [p_1, p_2, ..., p_N]$                                  ▷ ordered indexes
    $H = \{word => [indexes]\}$                  ▷ indexed SAX representation of $T$
algo:
    $words = SAX(T, n)$
    $H = hash\{word => [$indexes of each occurrence of this word$]\}$
    $min = min(hash.values)$                             ▷ (min = almost always 1)
    $P = [$indexes of words occurring only min time$] + [$other indexes, shuffled$]$
procedure InnerHeuristic
input:
    $T = [t_1, t_2, ..., t_N]$                                  ▷ time series
    $H = \{word => [indexes]\}$                  ▷ indexed SAX representation of $T$
    $p = int$                                        ▷ index of current subsequence
output:
    $Q = [q_1, q_2, ..., q_N]$                                  ▷ ordered indexes
algo:
    $Q = H\{words[p]\} + [$other indexes, shuffled$]$
```

Since Sequitur builds a compact, context-free generative grammar for any sequence, it has been used for various analytical tasks, such as program trace analysis [Walkinshaw et al., 2010], query of compressed XML databases [Lin et al., 2005] or structure inference in DNA and musical pieces [Earl and Ladner, 2003]. The algorithm has a complexity of $O(N)$, which makes it suitable even for long strings of text [Nevill-Manning and Witten, 1997].

Sequitur transforms any input sequence into a compact representation where two essentials constraints are met: no digram appears more than once in the output sequence (digram uniqueness), and no rule is used (either in the output sequence or in other rules) less than twice (rule utility). It is argued in [Senin et al., 2015] that the number of rules used to represent a given point in a Sequitur-compressed time series, being proportional to its compressibility, is a good approximation of the series' Kolmogorov complexity, or algorithmic complexity (*i.e.* the size of the smallest program capable of generating the series) at that point. The assumption is that an anomaly is very likely to correspond to a rising complexity, or a lowering compressibility.

Sequitur builds a grammar tree in which the depth of a leaf is directly related to its frequency in the original string. Higher level, non-terminal nodes in the tree correspond to frequent patterns in the data. Conversely, shallow branches denote rare patterns, which is of interest in anomaly detection. With the analytic decompression routine that follows, the rule density, *i.e.* the depth in the grammar tree, can be associated to any point in the series from the Sequitur representation as shown in Algorithm 8.

The anomaly detection algorithm proposed in [Senin et al., 2015] (Algorithm 9) applies Sequitur on the SAX words representing subsequences (*i.e.* a basic symbol could be 'aabacd'). It

Algorithm 8 Analytic unwrapping procedure

```
unwrap(token, depth, rules) =
 if token not in rules then (token, depth)
 else [unwrap(rules{token}[1], depth + 1, rules),
  unwrap(rules{token}[2], depth + 1, rules)]

unwrap(blob, rules) =
 unwrap(token, 0, rules) for each token in blob
```

uses the compressibility of the SAX representation of a time series, computed by compressing it and then unwrapping the compressed grammar tree, to approximate its Kolmogorov complexity. The rule depth of each symbol gives a "normality" score: a high rule depth corresponds to a highly compressible section of the time series, which is unlikely to contain anomalies, while a low rule depth indicates a section containing rare patterns that may be anomalies.

Algorithm 9 Anomaly detection using the Sequitur compression algorithm

```
procedure SequiturAnomaly
input:
    $T = [t_1, t_2, ..., t_N]$                                   ▷ time series
    $n = int$                                                    ▷ window size
output:
    ▷ Sequitur rule density @ each point of the series (inverse anomaly score)
    $density = [d_1, d_2, ..., d_N]$
algo:
    $words = SAX(T, n)$
    ▷ SAX words [e.g. 'abbcab'] are used as elementary alphabet for Sequitur
    $tokens, rules = Sequitur(words)$
    tokens = list of Sequitur symbols
    rules = hash {symbol → $[symbol|letter, symbol|letter]$}
    $unwrapped = []$
    for all $token$ in $tokens$ do
        $unwrapped << unwrap(token, 0, rules)$      ▷ $unwrapped$ is [(SAX word, depth)]
    for $i = 1$ to $N$ do
        $density = sum(unwrapped[max(i - n, 0), max(i - n + 1, 0), ..., i])$
```

The SAX/Sequitur anomaly detection algorithm fits our evaluation criteria as follows:

- *Low cost:* Sequitur has a linear complexity and SAX greatly reduces the size of the data on which it operates. This points to a low computational cost.
- *Long-term dependencies:* similar patterns occuring widely apart are still compressed the same way, thus tracking remote motifs should not be a limitation.
- *Real time:* the compression algorithm itself, as described in the original paper [Nevill-Manning and Witten, 1997], introduces lag because it requires the *full* pattern to be read

before it can be fully compressed. This is confirmed by the demonstrations shown in the SAX/Sequitur paper [Senin et al., 2015].

- *Configuration-free:* only the SAX parameters and the alerting threshold have to be specified.

SAX/Chaos Game The Chaos Game representation [Wei et al., 2005, Barnsley] is a way of generating a bitmap from a DNA fragment, *i.e.* a string generated by an alphabet of four symbols (A, C, T, G). It recursively splits the two-dimensional space into pixels representing the number of occurrences of specific strings, adding one character at each level (*e.g.* at level one, it generates a 4-pixel image with the total count of A, C, G, and T bases. At level 2, the image is 4-pixel wide, with pixel corresponding to length-2 strings such as AA, AC, AT, ..., TG, TT). The proposed algorithm uses the SAX representation of a time series, with an alphabet size of 4, to build such bitmaps [Kumar et al., 2005], and then compare the bitmap of a detection window (after the currently analysed point) and a lag window (before the point). Typically the lag window is 2 or 3 times longer than the detection window, therefore the bitmap must first be scaled.

Besides the graphical aspect of the bitmap generation, this algorithm is a simple histogram comparison: the frequency distribution of all possible N symbols strings is computed to the left and right of each point in the time series and the two distributions are compared to yield an anomaly score. Since the number of bins in the histograms grows exponentially with the analysis level, the SAX sliding window will typically be short. The authors use a level 3 analysis, *i.e.* 64-bin histograms.

The anomaly score at any given point t_i of the time series is given by $d_i = \sum_{j=1}^{N}(H_{i-lead,i} - H_{i,i+lag})^2$, where $H_{i,j}$ is the histogram computed between points t_i and t_j, as shown in Algorithm 10. Qualitative results presented in [Wei et al., 2005] show that even subtle structural anomalies in heartbeats can be detected in electrocardiogram time series.

Algorithm 10 Anomaly detection using the Chaos Game sequence representation

```
procedure ChaosGameAnomaly
input:
    $T = [t_1, t_2, ..., t_N]$    ▷ time series
    $n = int$    ▷ symbol size
    $w = int$    ▷ window size
    $D = int$    ▷ detection window length (in number of SAX windows)
    $L = int$    ▷ lag window length (in number of SAX windows)
output:
    $score = [s_1, s_2, ..., s_N]$    ▷ anomaly score
algo:
    words = SAX(T, 4, n, w)
    for $i = 1 + L$ to $N - D$ do
        $det\_map = ChaosGame(words[i, i + n, ..., i + n * (D - 1)])/D$
        $lag\_map = ChaosGame(words[i - L * n, i - (L - 1) * n, ..., i - n])/L$
        $score_i = EuclideanDistance(det\_map, lag\_map)$
```

The SAX/Chaos Game anomaly detection algorithm fits our evaluation criteria as follows:

- *Low cost:* the algorithm has a linear complexity and uses lookup tables and counters to compute distances. As with SAX/Sequitur, it is computationally inexpensive.

- *Long-term dependencies:* the algorithm can only track past patterns with its lag window, and its lead window needs to be large enough to accomodate a full occurrence of one.
- *Real time:* new data points are not processed until a full lead window has passed, which introduces a considerable lag.
- *Configuration-free:* the SAX parameters, lead and lag window sizes and a detection threshold need to be specified. However, let us remember that SAX parameters have sound defaults and little influence on the final result [Lin et al., 2003, Wei et al., 2005]. The window sizes, however, have to be determined by the domain of application.

3.4 Machine learning

Along with the auto-regressive methods from the statistics community and the symbolic methods from bioinformatics, methods from machine learning have been used successfully for time series anomaly detection. Machine learning, in this context, is about building a model of the normal behaviour of the time series in an unsupervised way and detecting deviations from this baseline. Given the challenges of working with time series as a data type, which we already described at length in Section 3.1, in particular the streaming nature of the data and the mismatch between a subsequence and a feature vector, specialized models had to be designed that can take *time* into account. The most prominent are the recurrent neural network topologies. In [Wang et al., 2017b], recurrent neural networks are shown to significantly outperform feed-forward networks, as well as a number of non biologically inspired state-of-the-art models, on time series classification problems.

The main topology for recurrent neural networks is the Long Short-Term Memory (LSTM) network, as first described in [Hochreiter and Schmidhuber, 1997]. Other models try to extend the notion of topology to temporal relationships, such as growing neural gas [Ehrensperger and Conradt, 2015] based on the earlier self-organizing maps, a family of neural networks designed to model the data space topology and extend traditional clustering methods such as k-means.

We focus here on neural network models, as their dominance in today's machine learning is overwhelming. The hope behind such models is that their inherent complexity brings an ability to generalize to different timescales, learn reoccurring patterns, react to abrupt changes, without explicit parameters telling them how to do so. The price to pay for these abilities is a high computational cost and a requirement for large volumes of input data in the training phase. Note that this cost is not so high in the runtime; an example is Google's Tensor Processing Unit, ot TPU, which exists in two variants, one kept in Google's datacentres and used to train models, and the other available as an IoT module that can only run pre-trained models to provide local analysis facilities for sensor networks[2].

3.4.1 Recurrent neural networks

The need for different neural network topologies stems from the difference between time series and other data types. The differences in *meaning* between the dimensions in a "traditional" feature vector and a time series subsequence are profound: in a feature vector, each dimension represents the same concept for each sample, *e.g.* the age of a person, the orientation of a line,

[2] `https://cloud.google.com/edge-tpu`

the weight of a mechanical part... The dimensions are mostly uncorrelated, as they represent different characteristics of a single object. In time series, however, the “dimensions” are a sequence of one-dimensional, standalone values. This differs from the feature vector paradigm in three main ways:

- The dimensions are highly correlated.
- A value in dimension n in one vector appears in dimension $n+1$ in the next vector, $n+2$ in the next, and so on as the sliding window extracts sample vectors [Bagnall et al., 2017].
- The order of the dimensions are meaningful, *i.e.* shuffling the samples in a time series yields a completely different time series, while shuffling the dimensions in a feature vector has essentially no effect. A classifier trained to classify images based on a vector of $(width, height, density)$ will perform equally well if trained on $(height, density, width)$: the vectors have the same meaning. On the other hand, the series $(1, 2, 3, 4, 5)$ represents the first natural numbers, while the series $(4, 2, 5, 3, 1)$ might as well be random.

These differences are the root of the meaninglessness of time series subsequence clustering [Keogh and Lin, 2005]. They are also the reason why the models designed to operate on time series are significantly different from those working on feature vectors.

Given the breakthroughs they have enabled in the last years, artificial neural networks require no introduction [Gurney, 2014]. In their usual form (feedforward neural networks, FNN), however, they are not optimal for time series analysis [Connor et al., 1994]. Instead, recurrent neural networks (RNN), where the output from one layer is fed back into the same layer at the next time step, are more generic models and have the ability to capture long-term dependencies in the input sequence [Elman, 1990]. LSTM are often used in tasks involving time-varying patterns of unpredictable length, such as natural language (at the character level) or speech processing (as raw audio) [Lipton et al., 2015].

The general form a feedforward layer in a neural network is:

$$h_t = \sigma(W \cdot x_t) \tag{3.9}$$

where h_t is the output of the layer at time t, x_t is the corresponding input from the previous layer, W the weight matrix and σ a squashing function.

In a recurrent network, the hidden state at time $t-1$ is fed back into the layer at time t using a second weight matrix U:

$$h_t = \sigma(W \cdot x_t + U \cdot h_{t-1}) \tag{3.10}$$

The typical recurrent network shape is depicted in Figure 3.3.

3.4.2 Long Short-Term Memory

For a long time, the only way to train a recurrent network was to unfold the recurrent units and generate a large, virtual feedforward network which was then trained using backpropagation. This approach, known as ***backpropagation through time*** [Werbos, 1990], was extremely costly in processing time and memory and suffered from exploding and vanishing gradients problems, which made convergence very slow or impossible. These problems were solved by the introduction of long short-term memory networks (LSTM) [Hochreiter and Schmidhuber, 1997], which use

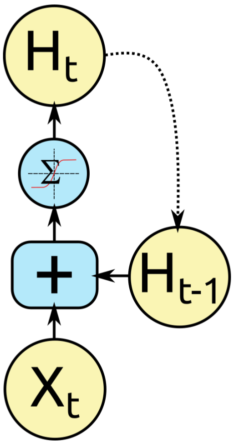


Figure 3.3 – Recurrent Neural Network

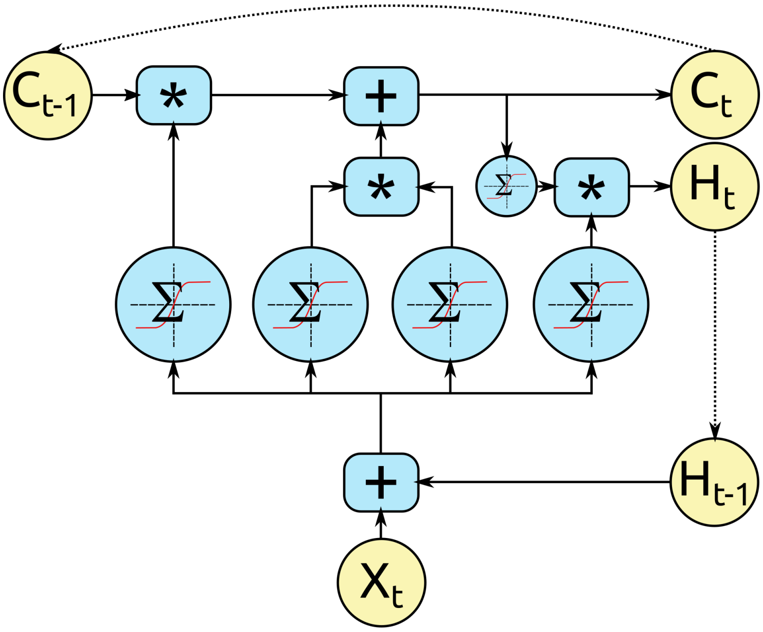


Figure 3.4 – Long Short-Term Memory cell

special gate units to control the error gradient flow. A typical LSTM layer is depicted in Figure 3.4. The values X_t, H_t and C_t are vectors corresponding respectively to the input, hidden state (*i.e.* output) and memory cell.

The notion of a memory cell is unique to the LSTM networks. It is controlled by *gates*: additional weight matrices that help keep the gradient under control. With the circle operator denoting the Hadamard product, *i.e.* $a \circ b = (a_1.b_1, a_2.b_2, ..., a_n.b_n)$, the equations governing an LSTM layer are:

Table 3.2 – Average performance of LSTM and RNN on open datasets

	LSTM	RNN
Precision	0.94	0.86
Recall	0.13	0.11
F-score	0.88	0.77

$$h_t = o_t \circ \sigma(c_t) \tag{3.11}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma(W_c \cdot x_t + U_c \cdot h_{t-1}) \tag{3.12}$$
$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + V_o \cdot c_t) \tag{3.13}$$
$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + V_f \cdot c_{t-1}) \tag{3.14}$$
$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + V_i \cdot c_{t-1}) \tag{3.15}$$

where h_t is the output of the layer, c_t is a memory cell and o_t, f_t and i_t are, respectively, the output, forget and input gates. Each gate is influenced by the input x_t through the W matrices, the previous output h_{t-1} through the U matrices and the memory cell through the V matrices. Note that the memory cell is updated exactly the same way as a recurrent neuron in an RNN.

The memory cell allows values to be retained for a long span of time, allowing the LSTM network to model long-term dependencies. The input gate allows the input value to flow in the cell, the output gate allows it to flow out. The forget gate makes it possible to erase the cell.

Though training an RNN means learning two matrices W and U and training an LSTM means learning 11 matrices, the LSTM has a much shorter training time than the RNN, and better overall results. This is because the backpropagation used for training is limited in its temporal reach: while the weight of the inputs in a full RNN does not decay through time, in an LSTM, past inputs are discounted by the gates. Many slight variations exist in the LSTM architecture, including the removal of gates or activation functions, but no variant appears to globally improve the performance over the “vanilla” model [Greff et al., 2017]. Interestingly, LSTM can be applied to problems traditionally in the realm of feature vector methods. In [Breuel, 2015], LSTM trained on the MNIST dataset of handwritten digits perform on par with early deep networks [Deng and Yu, 2011].

In [Malhotra et al., 2015], an LSTM network is proposed for time series anomaly detection and qualitatively demonstrated on power consumption time series, which are very similar to some network monitoring time series. The neural network is used to forecast the time series several points into the future; the statistical distribution of the forecasting error is then used to compute a likelihood of each point to belong to the normal distribution. The networks are trained using backpropagation. In the experiments the authors present on three open datasets (Space Shuttle Marotta valve time series, Power demand dataset and the qtdb/sel102 electrocardiogram)[3], the anomalies are clearly detected and the LSTM outperform RNN in terms of accuracy and recall. The results are summarized in Table 3.2. The F-score used to average precision and recall is their harmonic mean $F = \frac{p \cdot r}{p+r}$.

[3] These datasets are part of the UCR time series library and are available at `http://www.cs.ucr.edu/~eamonn/discords`

However LSTM, despite improving by a large margin over generic RNN – which is the main factor in their current success – remain expensive models, especially in the training phase. LSTM trained on the TIMIT speech dataset [Garofolo et al., 1993] – a 1993 audio dataset fitting on a single CD – typically take 10 to 35 hours to train [Greff et al., 2017]. Furthermore, learning rules with long-term dependencies, *e.g.* over 5000 samples, is considered a pathological case for recurrent networks. Generally, dependencies this long simply cannot be learnt by RNN, as the inputs are forgotten too quickly [Pascanu et al., 2013, Hochreiter and Schmidhuber, 1997].

3.4.3 Gated Recurrent Unit

The Gated Recurrent Unit networks (GRU) have been developed recently in an effort to overcome the perceived shortcomings of the LSTM. First devised for machine translation [Cho et al., 2014], the gated recurrent convolutional network is simpler that the LSTM and reaches better performance on small training sets [Chung et al., 2014]. Like LSTM and recurrent networks in general, GRU networks are designed with time-varying data in mind, and thus are suited for time series analysis.

The GRU is simpler than the LSTM in that it does not contain an output gate and memory cell; some versions even remove the forget gate. In its first formulation, a GRU layer has the following output:

$$h_t = (I - z_t) \circ h_{t-1} + z_t \circ (W_h \cdot x_t + U_h \cdot (r_t \circ h_{t-1})) \tag{3.16}$$

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1}) \tag{3.17}$$

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1}) \tag{3.18}$$

where z_t is the update gate and r_t the reset gate. The complete diagram of a GRU unit is presented in Figure 3.5.

Given the similarity between these gates, a Minimal Gated Unit (MGU) has been proposed in [Zhou et al., 2016] where both gates are replaced with a forget gate f_t. A further simplification that actually improves the accuracy of the model is to make the forget gate depend only on the previous output of the layer and not on the input [Heck and Salem, 2017]. Thus the output becomes:

$$h_t = (I - f_t) \circ h_{t-1} + f_t \circ (W_h \cdot x_t + U_h \cdot (f_t \circ h_{t-1})) \tag{3.19}$$

$$f_t = \sigma(U_f \cdot h_{t-1}) \tag{3.20}$$

With the LSTM at 11 matrices, the GRU at 6 and the MGU at only 3, the reduction in parameters is significant without any significant degradation in accuracy [Heck and Salem, 2017, Jozefowicz et al., 2015]. However, GRU and MGU are extremely recent. Ongoing research is still focused on validating the models themselves and has yet to propose application outside of classic benchmark problems. The MGU diagram is shown in Figure 3.6.

Overall, recurrent network architectures have made it possible to learn sequential patterns in time series and solved the problem of the time-varying input and exploding/vanishing gradients. Models such as the LSTM are mature and deployed in real-life environments to perform tasks such as language translation [Bahdanau et al., 2014]. Newer variants such as the GRU and MGU

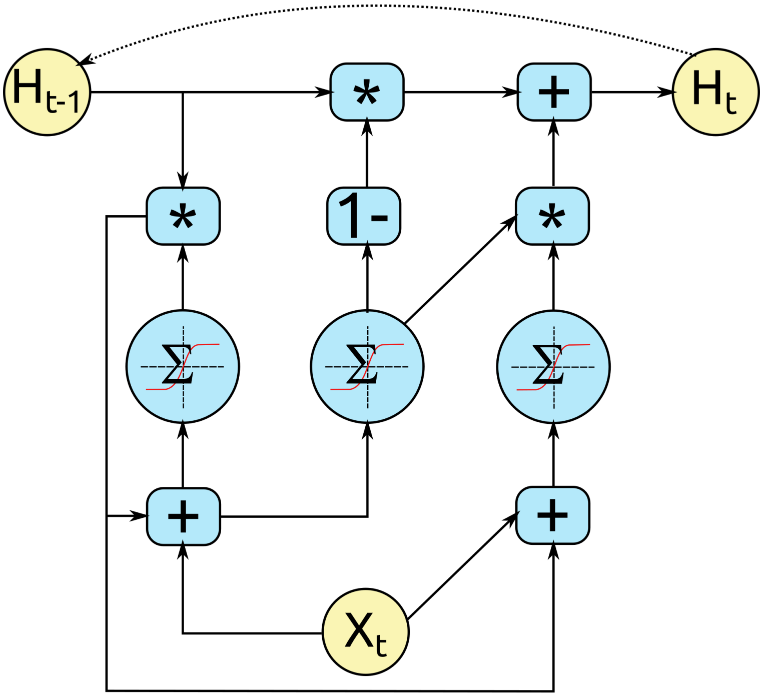


Figure 3.5 – Gated Recurrent Unit

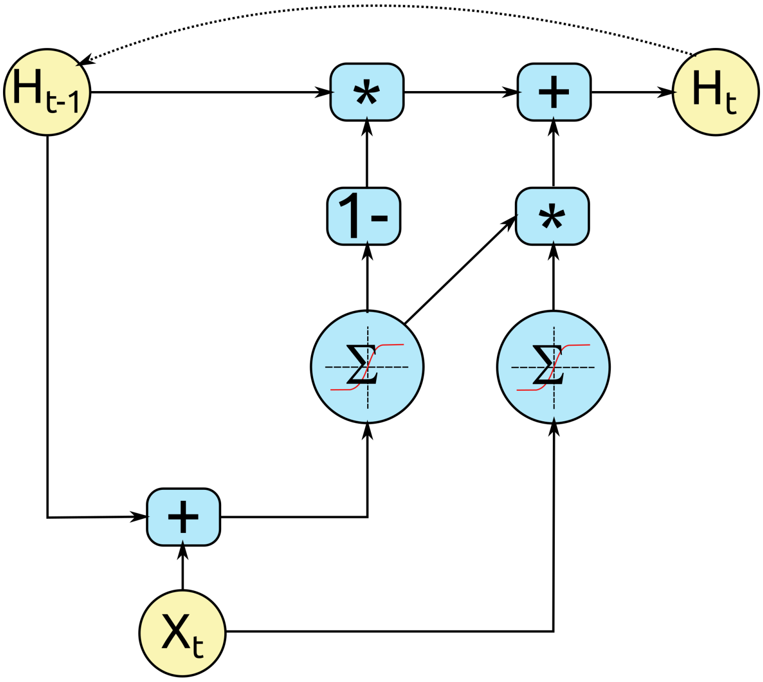


Figure 3.6 – Minimal Gated Unit

promise shorter training costs at the same level of accuracy, but they are yet to pass the test of massive benchmarks on standard datasets and real-world scenarios.

According to our evaluation criteria, the recurrent network topologies can be evaluated as follows:

- *Low cost:* even the MGU, which is the most efficient model, is an expensive one, at least in the training phase. The runtime phase has a linear complexity and requires a single pass on the data, thus can be more efficient.
- *Long-term dependencies:* recurrent network struggle with very long patterns (in the order of thousands of time lags [Pascanu et al., 2013]).
- *Real time:* the raw samples are processed as they are acquired, thus in real time.
- *Configuration-free:* as with other neural networks, the topology (size of the layers, activation functions, etc) has to be entirely specified, along with the learning rate and the training method itself, which also requires additional parameters (momentum, dropout rate, regularization...). This is an expert's job.

3.5 Open challenges

Though the problem of anomaly detection in time series has been studied from a number of perspectives and using methods from different communities, a number of challenges still exist, which have maintained the interest in this topic for decades. A number of trade-offs are associated with anomaly detection, and time series bring their own set of issues. Neural networks trade computational cost and training set size for domain-specific knowledge: they implicitly build a very specialized knowledge based on massive datasets. Auto-regressive models trade high forecasting accuracy for genericity: they only operate well within extremely confined boundaries. Symbolic methods trade speed for lag and accuracy: they buffer large amounts of data and have some unexpected failure modes, as we show in Chapter 9.

The case of network monitoring is complex: the time series have dependencies spanning across thousands of samples at multiple scales, the hardware dedicated to monitoring is typically low-end or ancient, and the alerts are supposed to be both precise and fast, *i.e.* be thrown at the earliest signs of an anomaly.

While auto-regressive models tend to work well for short-term forecasting of time series with a simple seasonal pattern, they scale poorly to long-term dependencies and fail to model multiple nested patterns and non-linear relationships [De Livera et al., 2011]. As time series generated by human activity (electric power demand, server load, network traffic...) tend to follow working days patterns, they exhibit at least daily and weekly patterns, which shift along the year and from year to year.

Recurrent networks, even optimized, take a long time to train and have trouble with long time lags and structured input [Dieleman et al., 2018]. Said otherwise, recurrent networks "lose track" of the long term dependencies of their input and fail to capture exact periodicity, such as tempo in music. They are limited by their high training time, non-negligible runtime cost and difficulty to conceptualize long, structured input.

In conclusion, though the problem of time series anomaly detection has been studied for decades, real-time, high performance models that can integrate long dependencies are still

	Low cost	Long-term dependencies	Real time	Configuration-free
Auto-regressive	✓	✗	✓	~
Distance-based	✗	✓	✓	✓
HotSAX	✗	✓	✗	✓
SAX/Sequitur	✓	~	~	✓
SAX/Chaos Game	✓	~	✗	✓
Neural networks	✗	~	✓	✗

Table 3.3 – Properties of time series anomaly detection models

lacking. Table 3.3 summarizes the properties of the explored models in light of the requirements of monitoring systems. The ~ sign is used to denote properties that depend on the use case or are not clearly mentioned in the literature.

The challenge we will address in this thesis is to define a model that can tick all the boxes of this table, *i.e.* manage long-term dependencies without requiring specific configuration, in real time, and at a low computational cost.

Chapter 4

Memory: Instance- and rule-based learning

The concept of immune memory shares many characteristics with classifiers in general: the basic idea is that *instances*, as they become known to the system, change its behaviour when exposed to similar instances again. In the case of biological immunity, instances are pathogens, and the underlying mechanism is the clonal expansion of lymphocytes: on the first exposure to a new pathogen, only a few antibodies are able to react; however, as they react, they trigger a positive feedback loop that generates multiple mutated copies of themselves. This allows the pathogen to be eliminated, although after a significant delay, and introduces a new population of highly specialized lymphocytes in the immune system. On secondary exposure, this population becomes instantly active, leading to a much faster and more intense immune reaction. This is also the basic principle of vaccination.

In classifiers, instances are training vectors, and the behaviour of the system is, in the case of supervised training, the generation of a class label – and, in the case of unsupervised training, of a label indicating to which cluster the instance belongs. *Training* is the process through which the output of the classifier, which is initially random, is tuned so that the correct labels are generated. In this regard training and first exposure are essentially the same process.

The main difference between immune memory and classification is that the immune system generally exhibits the correct response, except in very particular cases; the stake of vaccination and first exposure is to make the reaction *faster*, not to make it *correct*. In contrast, a classifier has to learn how to generate the correct output, and the time required to classify a data point, or the classification lag, is not improved during training. As far as we know, there exists no literature on the topic of training classifiers to reduce classification lag (which only has meaning for real-time systems); typically, the lag is a property or a parameter of the algorithm itself [Wei et al., 2005].

We define the concept of an *immune* classifier as a classifier whose response time, *i.e.* the time it takes, when an event occurs, to actually detect it, is improved during training. A classifier of this sort should fulfil the *failure detection* requirement of monitoring systems and provide the *memory* property of immune systems.

While there currently exists no such classifier, a number of methods and algorithms in the literature have drawn from similar concepts or offer a path towards their implementation. In particular, we focus on Artificial Immune Systems (AIS), which explicitely try to model processes of the immune system in Section 4.1. We then review a specific class of classifiers: instance-based classifiers, also called memory-based classifiers. As both their names suggest, they do not transform their training data to build a model; the only processing they perform is inserting samples in indexing structures for fast search and choose or discard samples. As such, they provide very clear and understandable decision rules and can be edited manually by adding or removing samples. Finally, we review condition-action systems, *i.e.* classifiers that generate pairs of the form `{input filter}` → `{action}`. The main type of such systems are Learning Classifiers Systems, as described by Holland in [Holland, 1983].

4.1 Artificial Immune Systems

Many types of immune-based metaheuristics have flourished between the late 1990s [Forrest et al., 1994, 1997] and the early 2000s [Aickelin and Cayzer, 2008, de Castro and Von Zuben, 2001, De Castro and Von Zuben, 2002]. All of them share a common design process: desirable properties are observed in the natural immune system; a core process is extracted from the immune system; a classifier is built around a formal specification of this process, *i.e.* an algorithm.

In a way, Artificial Immune Systems (AIS) evolved along with the understanding of human immunity [Jerne, 1974, Greensmith and Aickelin, 2009]. Early models based on the simple self/non-self discrimination were seen as possible alternatives to signature-based antiviruses [Forrest et al., 1994, 1997]. Later models integrated more plausible biological elements, such as PAMP (Pathogen-Associated Molecular Patterns) and danger signals [Aickelin and Cayzer, 2008]. During the 2000s, a number of classifiers based on artificial immune systems emerged, such as the Artificial Immune Recognition System (AIRS) [Watkins et al., 2004, Boufenar et al., 2018], aiNet [de Castro and Von Zuben, 2001] based on the immune network theory, as well as evolutionary-like algorithms based on the clonal expansion principle (CLONALG) [De Castro and Von Zuben, 2002]. More recently, AIS have been applied to collaborative filtering [Chen et al., 2015] and time series anomaly detection [Montechiesi et al., 2015].

However, it is important to note that biological plausibility is not necessarily correlated with performance as a classifier [Timmis et al., 2010]. Realistic simulations of the biological immune system tend to perform worse than algorithms loosely based on immune ideas and designed to fit the problem at hand. As an example, the various versions of the Dendritic Cell Algorithm, which tries to model as precisely as possible the biological role of dendritic cells in the immune system, had to be simplified to prove useful on machine learning problems [Greensmith and Aickelin, 2009].

In [Hart et al., 2003], the authors propose three distinct approaches to artificial immunity:

- A literal approach, where the designed system tries to fulfil the same function as an immune system, *e.g.* perform self/non-self discrimination in a computer,
- A metaphorical approach, where ideas and principles observed in immune systems are used as an inspiration in computer systems design,
- A simulation approach, where experiments are carried out *in silico* to further the knowledge about biological immune systems.

Theirs is the second approach; the apply the metaphor of T-cells moving through the lymphatic system to design monitoring and routing protocols for large scale wireless networks. The strong arguments of the paper, beyond this new model, are twofolds. Firstly, AIS can have different end goals and derive from biological immune systems in different ways. Secondly, when designing systems intended to solve a real-world problem, an engineering perspective on the problem and its constraints is absolutely necessary and must guide the design process of the AIS.

4.1.1 Biological immune systems

The inspiration for the AIS is the human immune system. The motivation for building a metaheuristic based on it stems from the various interesting properties associated with immunity, namely the self/non-self discrimination, memory, self-regulation, and decentralization [Dasgupta and Forrest, 1999]. These properties are associated with different theories and processes:

- The ability to distinguish normal, or "self" proteins and cells is provided by the negative selection process, which takes place in the thymus. The early works of Stephanie Forrest in the 1990s are based on negative selection. Biologically, the naïve T-cells, during their maturation in the thymus, are presented with a collection of proteins from the organism. Those that react, and thus would lead to an auto-immune reaction, are destroyed – hence the *negative* selection.
- The immune memory is maintained by a population of specialized memory T-cells [Omilusik and Goldrath, 2017] and B-cells [Hauser and Höpken, 2015], which are generated after an encounter with a new pathogen. T- and B-cells are the two subtypes of white blood cells; they are produced in the bone marrow and, in the case of T-cells, maturate in the thymus. This maturation process is the basis of the negative selection class of AIS. The long-lived population of memory cells allows for a faster and more efficient immune response upon secondary exposure and is the basis for vaccination. The population itself is generated by the clonal reproduction of effector T-cells upon encounter of a pathogen. This process takes place in the entire body, hence participating also in the decentralization property of the immune system. It is the basis of the CLONALG family of AIS algorithms, which are a particular type of evolutionary algorithms [Brownlee, 2005, De Castro and Von Zuben, 2002].
- The self-regulation property is linked to a number of processes, including but not limited to the negative selection of the thymus and the clonal expansion. Danger theory [Aickelin and Cayzer, 2008, Matzinger, 1994] allows for a very subtle regulation based on antibody activation and a danger signal (*e.g.* the presence of proteins linked to cell stress). Thanks to these regulatory processes, an alien bacteria generating no danger signal will cause no immune reaction. This allows for a model closer to reality, as the human body depends heavily on external organism, such as the intestinal flora [Aickelin and Cayzer, 2008]. This is the meaning of *immune tolerance.*
- Decentralization is a key characteristic of the immune system: T-cells and B-cells are disseminated throughout the body *via* the lymphatic system to detect pathogens as soon as they enter the body. The only centralized operation is the maturation of naïve T-cells in the thymus.

4.1.2 Artificial Immune System families

Most AIS actually applied to real-life problems – in contrast to theoretical work [Aickelin and Cayzer, 2008, Matzinger, 1994] – fall into two categories: negative selection and clonal expansion.

Negative selection is a type of nearest-neighbour algorithm, while clonal expansion belongs to the evolutionary algorithms family.

Negative selection [Forrest et al., 1994, 1997] Neirest neighbour classifiers are a family of supervised classifiers where the class of a new point is determined by the majority class represented in the k closest labelled points. From a geometric standpoint, the k neirest neighbours classifier (k-NN) is equivalent to the Voronoi tessellation of the feature space [Lee, 1982]; the class of a point is determined by the class of its Voronoi cell. Therefore, k-NN maps the entire space, *i.e.* any point can be associated with a class label.

As a special case, one-class classifiers map the feature space into positive detection zones where the distance to a training point is below a certain threshold and negative detection zones beyond [Khan and Ahmad, 2018]. Here, a point can be either mapped to the target class or to no class at all. The feature space is thus divided into positive islands in a negative ocean.

A typical one-class nearest neighbour classifier works in two steps: in the training phase, the training instances are inserted into a proper search structure, such as a k-d tree, that splits the search space so that neighbours can be retrieved efficiently. Then, in the prediction step, new data points are presented to the classifier, which computes their distances to the training instances and returns a positive or negative match. In contrast, the negative selection algorithm inserts an intermediate step:

- Negative training instances, *i.e.* training data that is known *not* to belong to the considered class, are inserted into a fast search structure.
- Random instances, called *detectors*, are generated.
- The detectors that trigger a positive detection are deleted.
- The remaining detectors are inserted into a fast search structure.
- New points that trigger one or more detectors are classified as positive, otherwise as negative.

In this regard, and given that the training samples are considered *negative* rather than *positive*, negative selection is a kind of inverse nearest neighbour classifier. In the works of Forrest [Forrest et al., 1994, 1997], negative selection is viewed as an alternative to traditional signature-based antivirus software. Instead of trying to match a known signature (positive selection), which may only be made available weeks after the first apparition of a new virus, a negative selection-based antivirus generates random signatures that match none of the sane files in a computer. New patterns matched by these detectors are considered as potential threats and trigger an antivirus reaction (warning, quarantine...).

While this approach had little application in real-life antivirus software, it was used successfully on a number of problems such as fault detection in industrial machines [Dasgupta and Forrest, 1999, Montechiesi et al., 2015, Strackeljan and Leiviskä, 2008], heat transfer simulation [Silva-Santos et al., 2018] or document classification [Zhu et al., 2017]. A fast negative selection algorithm based on Voronoi tessellation is described in [Zhu et al., 2017] (see Algorithm 11). It is designed to be parallel and run on Hadoop clusters using the Map/Reduce paradigm, which is typical of Big Data research. It claims superior detection rates and faster running times than competing solutions, including SVM; however, no comparison is attempted with artificial neural networks.

The proposed algorithm does not generate random detectors; instead, it uses the Voronoi tesselation of the feature space, as delimited by the training samples, to determine the ideal center and radius of the detectors so that they do not match any of the training sample, but cover the rest of the space as efficiently as possible.

In the negative selection model, the immune memory is modelled by the set of detectors; however, the faster and more intense secondary response of the immune system does not occur: the detection of a matching data point has no impact at all on the set of detectors.

Algorithm 11 Voronoi Negative Selection Algorithm

```
procedure VorNSA
input:
    $S$ training set of size $n$ in dimension $d$
    $R_s$ self radius
    $\delta$ minimum detector radius
output:
    $D = \langle V, R \rangle$ set of detectors                    ▷ Center and radius
algo:
    Normalize $S$ into $[0, 1]^d$
    Get Voronoi cells $\nu_1, \nu_2, ..., \nu_n = Vor(S)$ with vertices sets $V_1, V_2, ..., V_n$
    Let $VS = \langle V_i, S_i \rangle, i = 1, 2, ..., n$ be the set of all vertices and corresponding training
points
    $D = \emptyset$
    for $\langle V_i, S_i \rangle, i = 1, 2, ..., n$ do
        if $S_i$ is present at least 2 times in $VS$ then
            $R = dist(V_i, S_i) - R_s$
            if $R > \delta$ then
                $D = D \cup \langle V_i, R \rangle$
```

Clonal expansion [De Castro and Von Zuben, 2002, Gong et al., 2009] Clonal expansion is a particular case of evolutionary algorithm. In its most generic form, an evolutionary algorithm optimizes a population based on a fitness function, a selection strategy, a crossover operator and a mutation operator. Individuals with the highest fitness tend to reproduce together and are mutated over multiple generations in order to improve the overall fitness of the population; in the end, the individual with the highest fitness is returned as the solution to the problem expressed by the fitness function, or the individuals forming the Pareto front in multi-objective optimization problems.

Clonal expansion evolves a population of antibodies so as to increase their ability to match antigens. The algorithm uses only a mutation operator, without crossover: the individuals do not use sexual reproduction. The number of clones and mutation rate are proportional to the fitness of the individual.

Being an evolutionary algorithm, CLONALG is suited to data mining, pattern recognition and generic optimization [De Castro and Von Zuben, 2002]. However, clonal expansion-based algorithms model a different natural process (T-cell maturation) and can exhibit a faster convergence than competing evolutionary algorithms. In [Gong et al., 2009], the authors propose a secondary response clonal programming algorithm (SRCPA) described in Algorithm 12 that uses two populations of antibodies to model immune memory and secondary response. The main pool is the complete population of antibodies, while the secondary pool is regenerated at each time step. The worst antibodies of the main pool, as ranked by a fitness function, are incrementally replaced by the best of the secondary pool. This leads to a global optimization of the main population.

Algorithm 12 Secondary Response Clonal Programming Algorithm

```
procedure SRCPA
input:
    $n = int$                                   ▷ population size
    $s = int$                                   ▷ secondary pool size
    $N_c = int$                                 ▷ clonal size
    $F$ fitness function
    $T \in [0, 1[$                              ▷ secondary response activation ratio
Initialization:
    Randomly generate the antibody population $A(0) = \{a_1(0), a_2(0), ..., a_n(0)\}$
    Randomly generate the secondary pool $M(0) = \{m_1(0), m_2(0), ..., m_s(0)\}$
    $k = 1$
    while termination criteria are not met do
Clonal Selection:
        $q_i(k) = \lceil N_c \frac{F(a_i(k))}{\sum_{j=1}^{n} F(a_j(k))} \rceil, i = 1, 2, ..., n$      ▷ number of clones for each antibody
        Generate $Y(k)$ as the set of $q_i$ copies of each antibody $a_i$
        Generate $Z(k)$ by randomly mutating $Y(k)$
        Sort $\xi(k) = A(k) \cup Z(k)$ by decreasing fitness
        Set new population $A(k+1) = \{\xi_1(k), \xi_2(k), ..., \xi_n(k)\}$
Secondary Response:
        Let $a_b$ be the antibody $a_i$ with the highest fitness
        $\delta_{i,j} = dist(m_i(k), m_j(k)), i = 1, 2, ..., s; j = 1, 2, ..., s; i \neq j$ ▷ distances b/w $2^{\text{ndary}}$ AB
        $\delta_0 = min_{i,j}(\delta_{i,j})$                ▷ distance b/w closest pair of $2^{\text{ndary}}$ AB
        $\delta_{a,i} = dist(a, m_i(k)), i = 1, 2, ..., s$   ▷ distances b/w $a_b$ and each $2^{\text{ndary}}$ AB
        $\delta_{a,0} = min_i(\delta_{a,i})$                 ▷ distance b/w $a_b$ and closest $2^{\text{ndary}}$ AB
        $d = argmin_i(dist_j(m_j(k), a_b))$                ▷ index of $2^{\text{ndary}}$ AB closest to $a_b$
        if $\delta_{a,0} < \delta_0$ then     ▷ if the AB is closer to a $2^{\text{ndary}}$ AB than any other $2^{\text{ndary}}$ AB
            $m_d(k+1) = a_b$                               ▷ then replace this $2^{\text{ndary}}$ AB with it
        else
            $w = argmin_i(F(m_i(k)))$                      ▷ worst $2^{\text{ndary}}$ AB
            $m_w(k+1) = a_b$                               ▷ else replace the worst $2^{\text{ndary}}$ AB
        Randomly select $t = \lfloor T \times s \rfloor$ AB from $M(k+1)$ to replace the worst $t$ AB in $A(k+1)$
        $k = k + 1$
Termination:
    return AB from $A(k)$ with highest fitness
```

4.1.3 Artificial Immune Systems for anomaly detection in time series

In [Dasgupta and Forrest, 1995], Dasgupta *et al.*propose, as early as 1995, an artificial immune system to detect anomalies in time series based on the negative selection principle. The value space of real-valued time series is reduced into a binary form using a linear normalization and vector quantization. Non-overlapping sliding windows are used to extract vectors. This encoding strongly resembles SAX [Lin et al., 2003], using a uniform instead of a normal data distribution model. The negative selection principle is then used to generate detectors matching none of the extracted windows.

The authors report good performance on artificial datasets, but no evaluation is carried out on real-world data and, while the possibility of creating the detector set on-line is discussed, no such approach is actually described nor evaluated. Therefore, the set of detectors does not change during the lifetime of the AIS.

In [Montechiesi et al., 2015], an alternative approach based on the immune network, using both positive and negative training samples, is proposed for anomaly detection in mechanical motor bearings. It updates the AbNet algorithm by De Castro *et al.*[De Castro and Von Zuben, 1999] by using real-valued feature vectors and the euclidean distance instead of binary features and the Hamming distance. Rather than using the real data, or a quantized version thereof, this approach uses domain-spacific features computed at each cycle of the engine, *i.e.* using non-overlapping sliding windows of variable length. While the performance seems high, the only benchmark is performed on a very small dataset and does not include running times.

4.1.4 Limitations of Artificial Immune Systems

While the field of Artificial Immune Systems seemed to gain traction in the late 1990s and early 2000s, a number of setbacks and the success of Bayesian methods and artificial neural networks has slowly drained the interest in the AIS paradigm. In [Freitas and Timmis, 2003] and [Freitas and Timmis, 2007], the authors plead for a problem-oriented design of AIS, instead of the model-driven approaches. The lack of a niche where AIS perform consistently better than competing frameworks was already obvious in 2008 [Hart and Timmis, 2008], before the deep learning revolution. In [Stibor et al., 2005], the negative selection algorithm is compared to a positive selection algorithm (logically equivalent to a nearest neighbour classifier with a distance threshold) and a one-class SVM and fails to outperform any of them. Furthermore, the scalability of the negative selection algorithm in high dimension spaces is questioned.

In [Haidar, 2011], an AIS is compared to a naïve Bayesian classifier for document classification and also fails to outperform its competitor. In [Freitas and Timmis, 2003], the negative selection algorithm is found inferior to evolutionary algorithms, and conceptualized as a *random* search whereas the evolutionary approach is viewed as a *guided* search.

In general, AIS suffer from high computational costs and have yet to find an application in which they consistently outperform more mainstream and lightweight algorithms. They also fail to provide the expected properties of natural immune systems as defined in [Dasgupta and Forrest, 1999] (self/non-self discrimination, memory, self-regulation and decentralization).

4.2 Instance-Based Classifiers

Instance-Based Classifiers (IBC, or IBL for Instance-Based Learning) are also known as lazy classifiers: no generalization of the training set is attempted before the classifier actually receives queries [Perrizo et al., 2002]. Then the predicted class of a point is deduced from the classes of its nearest neighbours in the training set. Contrary to most machine learning algorithms, they are model-free: samples are classified based on their proximity to training instances. Their training phase is generally extremely fast, consisting only in inserting the training set into an appropriate search data structure. Class prediction is slower and depends on the efficiency of this structure and the number of lookups required to compute a class label [Aha et al., 1991].

4.2.1 Motivation for instance-based classifiers

Instance-Based Classifiers have a number of desirable features which, historically, led to their development. They require no prior knowledge of the statistical distribution of the samples, provide simple decision rules and require no training phase. In the 1967 foudation paper by Cover and Hart, the 1-NN decision rule is demonstrated to yield a classification error of at most twice the Bayesian decision rule [Cover and Hart, 1967]. This is an impressive result, given that the Bayesian rule requires perfect knowledge of the distribution of each class.

An interesting property of IBL classifiers is that they are almost non-parametric [Cover and Hart, 1967]: the typical case is k-NN, where the only parameter is the number of neighbours k. Thus they are easy to test on many problems as a first approach. Simple as they are, they also offer surprisingly high classification performance. In [Androutsopoulos et al., 2000], a nearest neighbour classifier is compared to a naïve Bayesian classifier on the spam detection problem, which is a typical application of Bayesian classifiers, and found to perform at the same level. In [Mustafa et al., 2012], a nearest neighbour classifier performs on the same level as an artificial neural network in an EEG classification task.

Generally speaking, lazy classifiers are considered "simple and effective" tools [Perrizo et al., 2002]; the research on the topic is mainly focussed on improving the search structures for exact and approximate nearest neighbour search [Perrizo et al., 2002, Aha et al., 1991, Wilson and Martinez, 2000]. The most commun accelerating structures to speed up the nearest neighbour search in high domensional spaces are the k-d tree [Bentley, 1975] and the ball tree [Omohundro, 1989]. These data structures divide the feature space so that the nearest neighbours of a test vector can be found without calculating the distance of this vector to all the others in the training dataset.

Lazy classifiers also offer the advantage to not require the massive computational power and datasets associated with neural networks, yet perform comparatively on various tasks. In fact, we show in Chapter 8 that a k-NN classifier outperforms neural networks, SVM and decision trees in time series classification with very small sample sets.

4.2.2 Instance-Based Classifiers as immune memory

Instance-Based Classifiers are the simplest model for an immune memory. In particular, one-class nearest neighbour classifiers, by using only "normal" examples, are able to provide the type of self-tolerance researched in the AIS field, *i.e.* to define the notion of *self* by opposition to *non-self*. A generic one-class nearest neighbour classifier (JKNN) is described in Algorithm 13 [Khan and Ahmad, 2018].

The JKNN algorithm compares the average distance between a test sample and its j nearest neighbours and the average distance between these neighbours and their k nearest neighbours in the training set. Note that j and k can both be set to 1, creating a very simple 11NN one-class nearest neighbour classifier. In [Khan and Ahmad, 2018], the decision threshold θ is set to 1, meaning that the test sample is classified as an outlier if it is, on average, farther away from its nearest neighbours than the neighbouring training samples. More tolerance can be added by increasing θ. The authors also suggest that the effect of increasing θ in a 11NN classifier is equivalent to fixing $\theta = 1$ and increasing k and, conversely, decreasing θ at $j = 1$ is equivalent to increasing j with $\theta = 1$. They show that, for large values of j and k, JKNN reduces to 11NN with $\theta = \frac{k}{j}$.

Algorithm 13 JKNN algorithm, most generic one-class lazy classifier

```
procedure JKNN
input:
    $T$ training set of size $n$ and dimension $d$
    $j$ number of primary neighbours
    $k$ number of secondary neighbours
    $dist$ distance function
    $\theta$ decision threshold
    $x$ test sample of dimension $d$
output:
    $+1$ for positive match, $-1$ for negative match
algo:
    Let $P = (p_1, p_2, ..., p_j)$ be the $j$ nearest neighbours of $x$ in $T$ under $dist$.
    $\delta_P = \frac{1}{j} \sum_{i=1}^{j} dist(p_i, x)$
    for $i = 1, 2, ..., j$ do
        Let $S_i = (s_{i,1}, s_{i,2}, ..., s_{i,k})$ be the $k$ nearest neighbours of $p_i$
    $\delta_S = \frac{1}{j*k} \sum_{i=1}^{j} \sum_{l=1}^{k} dist(p_i, sii, l)$
    if $\frac{\delta_P}{\delta_S}$ <theta then
        return +1
    else
        return -1
```

4.2.3 Limitations of instance-based classifiers

The simplicity of lazy classifiers comes at the price of certain weaknesses. In [Aha et al., 1991], the authors cite a number of limitations of IBL that have been established over the years and compromise their overall performance:

- Computational and memory cost due to the use of all training samples
- Sensitivity to noise
- Sensitivity to irrelevant attributes as well as missing attributes

- Strong dependence of the performance on the chosen distance function

Along with these issues, they propose mitigation techniques for the storage and computational requirements and noise sensitivity. In [Wilson and Martinez, 2000], a set of algorithms is proposed to reduce the storage demand by dropping the least relevant samples. However, nearest neighbour algorithms remain computationally demanding, specially in high dimensions, a limitation also affecting the Artificial Immune Systems working on the same principles (*e.g.* negative selection algorithms); their inability to distinguish between relevant and irrelevant attributes also limit their scalability to high dimensional spaces.

4.3 Rule induction systems

The generic problem of deciding what action to take (*e.g.* to alarm or not to alarm) in a specific situation summarized as a vector, without human intervention, if equivalent to automatically generating if-then rules that apply to that vector. When these rules are hand-written by humans, then the resulting classifier is an expert system. When the machine is in charge of generating the rules, then the classifier is either a decision tree, where rules are nested in a hierarchical structure, or a learning classifier system, where a flat population of rules is evaluated concurrently and compete to generate an action. Contrary to instance-based classifiers, induction systems build models of the input data and require a supervised training process.

4.3.1 Decision trees

Decision trees are an old and well-known technique for recursively splitting the input space in two. The leaves of the resulting tree contain the class labels. Once built, a decision tree is very fast; however, the building process is far from trivial. In particular, techniques for finding the optimal feature and split point at each node requires specialized heuristics. In [Quinlan and Rivest, 1989], the minimal description length is used: a split is added to the tree whenever it allows for the cumulated size of the tree and the labels to be reduced. The features being boolean, the split point is not a problem.

This hints at a well-known issue with decision trees: they do not scale well to high-dimensional spaces [Do et al., 2010]. Random forests, *i.e.* collections of stochastic decision trees, are able to overcome this problem, at the cost of higher training budgets and low explanatory power. The other problem is that the size of the tree, which is linked to its training and runtime costs and to overfitting, cannot be jointly optimized with its accuracy: one has to optimize one and set a bound to the other [Safavian and Landgrebe, 1991].

Using statistical tests to determine along which feature and at which threshold to split yields the conditional inference tree classifier [Hothorn et al., 2006], which trains models with better accuracy and better explanatory power than traditional splitting/pruning approaches. Splitting along many features at the same time can be accomplished by using k-means clustering to partition the input space [Muniyandi et al., 2012].

Another method to generate the trees is to use genetic programming [Oksel et al., 2016], *i.e.* to stochastically build the trees following the rules of artificial evolution, using the size and accuracy of the resulting tree to derive a fitness function. This allows, in particular, to split the input space obliquely, according to a linear combination of features.

4.3.2 Learning Classifier Systems

The most straightforward way of representing learned patterns that alter the way a system behaves – which is exactly what the immune memory is – is through condition-action systems. In Section 3.3.5, we defined rule mining in time series, which extracts pairs of the form `{pattern1}` → `{pattern2}`, *i.e.* one pattern is followed by another. Action-rule systems, on the other hand, produce pairs of the form `{pattern}` → `{action}`: a pattern *triggers* an action. The classifier is then evaluated on the consequences of this action.

These systems, of which the Learning Classifier Systems (LCS) are the most well-known example, learn rules by trial-and-error and apply them by interacting with an *environment*. Contrary to neural networks, which generate statistical associations, LCS generate knowledge in the form of rules to be applied in specific circumstances. Note that in the IT industry, the term of "knowledge base" is used to describe web pages containing step-by-step instructions to debug, install or configure software in very specific cases.

Learning Classifier Systems were developed by Holland in the early 1980s as a reaction to the perceived brittleness of expert systems [Holland, 1983]. LCS puts an evolutionary algorithm in charge of the rule generation instead of a human expert in an attempt to generate more generic rules that do not reproduce the failure modes of experts systems in corner cases.

In its original form, the LCS uses message-passing to trigger rules, and specific rules can trigger actions. Thus before an action is performed, a long chain of rules may be necessary. Then a reward obtained from the environment allows rules to be reinforced. The rules with the highest credit are triggered in priority. LCS thus belong to the field of reinforcement learning (see Chapter 5).

The message-passing mechanism was later removed with the Zeroth Level Classifier System, or ZCS [Wilson, 1994], making the internal workings of the classifier much simpler and the dynamics more understandable. The ZCS was in turn improved into the XCS[1], which replaces the reward as a fitness function for its classifier population by the ability of each classifier to predict the correct reward; the genetic algorithm used to generate and update rules is applied independantly on *niches* of classifiers matching the same input vector [Wilson, 1995, Butz and Wilson, 2002]. This method makes the XCS more generic than the ZCS; the XCS has since then been successfully used in various research fields, including medicine: in [Skinner et al., 2007], an XCS is used to classify electroencephalographic signals. It performs better than competing methods, with the exception of the naïve Bayesian classifier.

By contrast, classifier systems have been adapted for supervised instead of reinforcement learning. In [Bernadó-Mansilla and Garrell-Guiu, 2003], a sUpervised Classifier System (UCS) is proposed. The resulting classifier is a population of micro-classifiers that associate a condition, *i.e.* an input vector, to a class label. The condition uses the ternary alphabet $\{0, 1, \#\}$, or "zero-one-any", that allows partial matchings of binary strings, as in any LCS. The fitness of each classifier depends on its accuracy in the classification task, and the niche genetic algorithm is applied independently for each class, generating more covering rules.

In [Shafi et al., 2009], UCS and XCS are compared for intrusion detection in computer systems. Overall, XCS performed better, but both algorithms had to be modified to cope with the multivariate, real-valued datasets. The overall accuracy of XCS, averaged over five different datasets, was superior to any competing algorithm, though they only achieved the *best* accuracy on a single dataset amoung the five used. This is an indicator of the XCS' genericity: they

[1] "XCS", to the best of our knowledge, stands for X Classifier System.

perform well on very different datasets, where competing models perform well on one but fail on others.

Just as artificial immune systems, LCS have been limited by their representation of their inputs as binary strings and, as with artificial immune systems [Montechiesi et al., 2015], extensions of the XCS have been developed to cope with non-binary inputs. The representation proposed in [Stone and Bull, 2003] uses intervals, defined by a center and a spread, as symbols: instead of `01011#0`, or "match 0, then 1, then 0, then 1 twice, then anything, then 0", an input might be `(0.3, 0.1), (0.6, 0.2), (1, 0.5)`, *i.e.* "match any number between 0.2 and 0.4, then any number between 0.4 and 0.8, then any number between 0.5 and 1.5". The "dont't care" symbol # can be replaced by a practically infinite interval, *e.g.* `(0, 99999)`.

Ambitious hybrid models based on the XCS have also been developed. In [Bull and O'Hara, 2002], a Neuro Classifier System (NCS) is proposed. Instead of a simple binary or real-valued filter, the input is processed by a neural network whose output is the action to be executed. The NCS is unique in that it evolves a ***population*** of neural networks. It performs similarly to XCS on the classical toy problems of LCS, but is also extensible to function approximation and real-valued input and output.

Models such as the XCS come very close to some artificial immune systems, which has been all but missed by the AIS community [Vargas et al., 2002]. The binary string matching and the evolution, either through a niche genetic algorithm or by hypermutation, of the matched classifiers, is extremely similar, to the point that a classifier using one of these paradigms can be mapped to the other; the main difference comes from the application of genetic operators. The AIS uses only mutation, while LCS use standard crossover and mutation operators.

4.4 Conclusion

The purpose of an immune memory is to build rules that associate a failure with an alerting action. This can be either viewed as a classification or a reinforcement problem. The rules have to be learnt online, but they generally correspond to isolated events, *i.e.* they do not represent classes and do not try to generate a map of the entire input space.

Artificial immune systems have shown their limitations in terms of computational cost and overall performance, and applying classifier systems to the same problems, given their similarity, does not seem promising. In particular, classifier systems shine when the "right" action for any given situation is not known in advance, but can be found by trial and error or by simulation [Abbasi and Naghavi, 2017].

Instance-based classifiers, on the other hand, work well to partition the input space with only a few samples, but are dependent on user parameters, such as the number of neighbours to consider and the maximal search radius. Because of their simplicity, their acceptable performance in high-dimensional space and their ability to handle non-boolean data types, instance-based classifiers seem suited to the notion of immune memory as defined in the context of this thesis. They lack, however, the adaptivity of evolutionary-driven methods.

With respect to our initial proposal that an immune classifier should learn to improve its response time as it runs, the most interesting models are the LCS family of algorithms: they are designed from ground up to operate online and improve as they go. There is, however, a mismatch between the notion of a population of agents interacting with an environment, be it in an AIS or an LCS model, and a set of detectors monitoring many unrelated time series in search of anomalies – which is more similar to the operating mode of simpler Instance-Based Classifiers.

Essentially, IBL, AIS, decision trees and LCS form a continuum in terms of model complexity: IBL are the simplest classifiers, with no model at all; the negative selection flavour of AIS is very close, but more complex, as it requires random detector generation. Decision trees hold a middle ground by introducing an actual model of the data. Decision trees, AIS – in particular the CLONALG variant – and LCS can all be trained by evolutionary algorithms, introducing the highest level of model complexity.

From this study, we conclude that LCS are most suited to model an entire immune system as a multi-agent system, but instance-based classifiers are sufficient for an immune *memory*. In particular, the absence of a training phase mean that online – even real-time – operation can be readily achieved without alteration to the classifier. It also limits the associated computational cost and the amount of tuning required, thus matching the requirements of monitoring systems we defined in Section 1.3.

Chapter 5

Tolerance: Active learning

In the context of immune systems, the notion of expert has no equivalent. However, we hypothesize that expert interaction, in the form of a question/answer dialogue where the algorithm asks for labels on specific data points and the expert provides them, is one way immune tolerance can be introduced in a monitoring system. Tolerance is the process by which reactions of the immune system can be suppressed under certain circumstances, allowing for instance a rich bacterial flora to develop in the intestine. In biological immune systems, it is achieved by the maturation of T-cells in the thymus, which is modelled by the negative selection algorithm in Artificial Immune Systems. Tolerance, in the case of monitoring systems, translates into the ability to ignore conditions that would otherwise trigger an alarm.

This can be accomplished by having the user tag the incoming alerts as either true or false. This information can then be made available to a classifier to avoid future false alarms. Allowing a classifier to be trained efficiently on a small dataset of alerts can also reduce the computational cost of training and running the classifier, which is essential in monitoring systems. If the training samples are carefully chosen to be the most relevant to the problem, then a small set of queries can flow from the classifier to the expert at a limited rate, with the replies increasing the accuracy of the model as time goes on [Thiam et al., 2017]. This type of scenario, using direct expert feedback to drive the classifier, is appropriately named an "expert in the loop" scenario [Malbasa et al., 2017, Veeramachaneni et al., 2016]. The associated algorithms are grouped under the term *active learning* [Tong and Chang, 2001].

In any machine learning endeavour, at least two experts are required: a machine learning expert and a domain expert [Holmes et al., 1994]. They can obviously be the same person, but they play very different roles: the first one sets the parameters of the model, *e.g.* the kernel of an SVM, the topology and learning rate of a neural network, the crossover and mutation operators of an evolutionary algorithm... The second one gives meaning to the inputs and outputs of the system and helps the ML expert set some of the parameters and determine whether the model is performing correctly or not. Typically, in the case of a monitoring system, one would be the *designer* of the system and the other the *user.* In active learning, the designer sets the parameters of the classifier, while the user answers its queries, *i.e.* provides labels: the *user* is the expert who is "in the loop".

Most machine learning problems trivially fall into one of two categories: supervised learning, or prediction, where the goal is to *predict* the label of a vector, and unsupervised learning, or clustering, where labels are implicit and determined by the way points clump together into dense *clusters* [Chaovalit and Zhou, 2005, Sathya and Abraham, 2013]. The former is used when trying

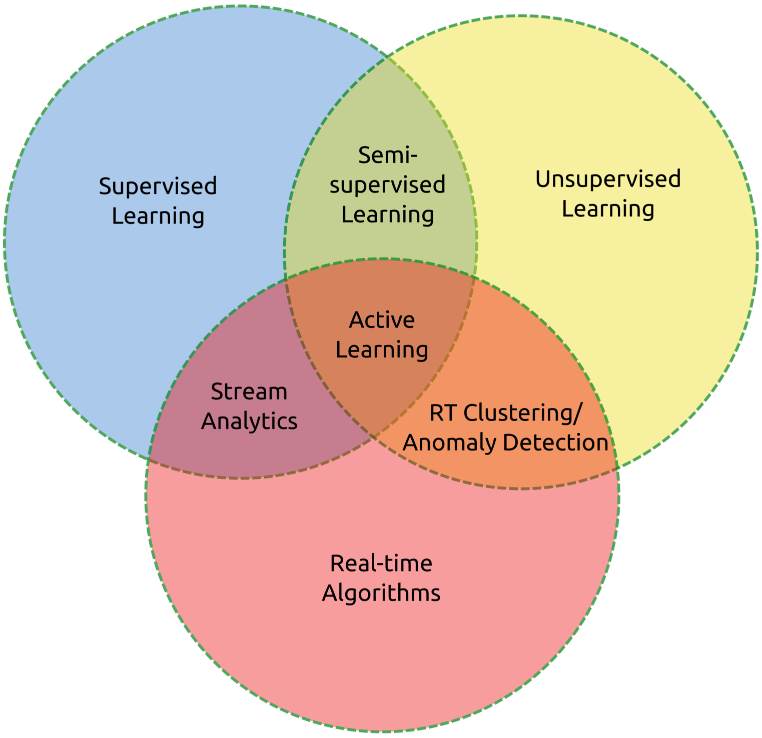


Figure 5.1 – Active Learning position in the Machine Learning landscape

to detect known features in the data (*e.g.* cats in images) and the latter to extract patterns from unknown data.

Problems where only part of the data are labelled are said to be ***semi-supervised***. In such problems, clustering can help train the model, then an output layer can be added to match the belonging to certains clusters to a specific label. Semi-supervised also generally means that the labelled points are essentially random, *i.e.* no specific algorithmic process has led to the choice of the set of points that have received a label, and that all training data and labels are available upfront.

A very specific semi-supervised scenario is that of streaming, unlabelled data being fed in real time into a classifier, but with an expert available to add a label to some (but only a few) vectors. In this use case, the classifier itself *asks* the expert for input, with a constraint on the number of calls it can make per unit of time (*e.g.* no more that one request per hour) [Haertel et al., 2008]. The fundamental difference between this *active* learning style and the more traditional semi-supervised learning is that the points to be labelled are chosen by the classifier, based on its internal representations. The question then becomes: How to decide, when perhaps only one point in a thousand can be labelled, which one it should be [Olsson, 2009]? Of course, it should be the point that will have the most impact on the model depending on which class its label falls in; but quantifying the importance of a point in this way is not trivial. The position of active learning in the spectrum of machine learning flavours is depicted in Figure 5.1.

The problem of expert interaction with a monitoring system is typically a case of active learning: the expert interacts with the classifier on a limited, real-time basis. The challenge addressed by the active learning framework is to make the best possible use of expert time to increase the classifier's performance as fast as possible.

In the following, we review the models used in active learning along with representative applications. Section 5.1 compares the methods used to transform a supervised classifier into an active one. We also compare active learning with two related paradigms: semi-supervised learning in Section 5.2, where labels are available for random points, *i.e.* points that have not be chosen by the classifier, and reinforcement learning in Section 5.3, where an agent evolves in an environment with which it interacts, and receives rewards depending on its actions [Sutton and Barto, 1998]. Finally, Section 5.4 outlines the open challenges in active learning that will be addressed in this thesis.

5.1 What is active learning?

5.1.1 Overview

Active learning is the process by which a classifier is bootstrapped with a very small number of labelled samples and retrained by iteratively querying the expert for more labels, choosing the points that are hardest to classify as queries to the expert [Tong and Chang, 2001]. Generally, two methods exist to determine which point to choose [Olsson, 2009]: uncertainty sampling (see Section 5.1.2) and query by committee (see Section 5.1.3). Uncertainty sampling relies on a single classifier identifying the most relevant points to label, while query by committee uses multiple classifiers and considers the disagreements between them as an indication of uncertainty.

Because it is trained incrementally and relies on only a few labels, active learning is an appealing paradigm in domains where the problems have the following characteristics [Tong and Koller, 2001]:

- The classes are unbalanced, *i.e.* some classes are overrepresented with respect to others. This is typically the case in anomaly detection, where anomalies are, by definition, rare, and normal points are frequent.
- The cost of labelling data points is high, typically because it involves a manual process carried out by a domain expert.
- The data may be available in streaming and not all at the same time [Saunier et al., 2004].

Such domains can be extremely diverse. In [Polewski et al., 2015], active learning is used for dead tree detection in aerial pictures of forests. The images are first segmented, then features are extracted from the resulting shapes. This is a standard technique in image analysis. The model used to classify these portions of images into living or dead trees is a logistic regression model on the extracted features [Friedman et al., 2001]. However, labelling image patches is a time-consuming process, thus only the most informative samples, *i.e.* those that will significantly increase the classifier's accuracy, should be labelled. This is where active learning comes into play: in order to efficiently train the logistic regression model, a first step needs to be applied to select the training set. The unlabelled feature vectors are first filtered by Principal Component Analysis (PCA) in order to remove the most similar ones and keep as much diversity as possible in the training set. Then the active learning procedure itself is applied using Expected Error Reduction (EER, see Algorithm 14).

EER iteratively trains the classifier with an initial training set to which it adds a single point of the unlabelled pool, trying each possible label for this point and choosing the point and label

Algorithm 14 Expected Error Reduction

```
procedure EER
input:
    $T$ Initial training set
    $P$ Conditional distribution of labels given features
    $U$ Pool of unlabelled samples
    $n$ Desired size of the final training set
algo:
    while $|T| < n$ do
        $(c, \hat{P}_T = trainClassifier(T)$
        for $y \in U$ do
            for $y \in \{0, 1\}$ do
                $(c', \hat{P}) = trainClassifier(T \cup (x, y))$
                $e(x, y) = e(x, y) + \hat{P}_T(y|x)E(\hat{P})$
        $(x^*, y^*) = argmin_x(e(x, y))$
        $T = T \cup (x^*, y^*)$
        $U = U \setminus \{x^*\}$
```

that minimize the expected error of the classifier. Thus each iteration adds a new point to the training set, until said set has reached a defined size. Then the expert is asked to provide labels for the added points that are using "dummy" labels.

While the resulting model reaches an accuracy of 89% on the dead tree detection problem using less than 100 labels, it is worth noting that the very large number of classifier retrainings – the classifier being retrained at each iteration twice per unlabelled point – would lead to an extremely large computational cost. The authors circumvent the problem by choosing a classifier that can be ***partially*** retrained when new data are presented to it; however, not every classifier is capable of that.

In [Malbasa et al., 2017], active learning is applied to voltage stability monitoring in the power grid. The proposed model uses a simulation of the considered electrical systems instead of a human expert as the "oracle" generating labels, and new labels, *i.e.* new simulations, are required whenever the predicted label produced by the classifier differs from the observed behaviour of the system. This allows new types of behaviours to be detected and learnt online, which is particularly useful in dynamic, evolving systems such as the power grid.

In [Veeramachaneni et al., 2016], active learning is applied to network security and event analysis, a domain close to network monitoring. The problem encountered in network security is the extremely large amount of data generated by systems logging their activity, and the difficulty of discovering a potential attack in such a volume of data. The AI^2 model proposed by the authors consists in three layers:

- A big data processor parses the log files, extract features and learns the typical behaviour of these features.
- An unsupervised outlier detector extracts the behaviours that deviate from the learnt models and submits them to the human analyst.
- A supervised classifier is trained on the collected labelled data.

The expert receives events generated both by the ***unsupervised*** model and the ***supervised*** one. In the course of months, ss the number of labelled samples increases, the ability of AI^2 to detect new forms of attacks and recognize known ones improves as well. In particular, the authors highlight the high recall, or true positive rate, of the active classifier, reaching a recall of 0.868 against 0.737 only for a purely unsupervised model.

5.1.2 Uncertainty sampling

In the following, the i^{th} sample of a dataset is designated as x_i, its class as y_i. In models based on a confidence measure for its label, this measure is denoted c_i; in models measuring an uncertainty instead, this uncertainty measure is denoted u_i. There are m possible classes.

Uncertainty sampling is based on selecting points for which the predicted class label has the least confidence. The classifier with the most straightforward confidence measure is the Support Vector Machine, which is why early applications of active learning focussed mainly on SVMs [Tong and Chang, 2001, Tong and Koller, 2001]. In SVMs, the feature space is divided by hyperplanes that materialize the borders between different classes. Uncertainty sampling proceeds by selecting, at each step, the point that is closest to the hyperplanes. The basic algorithm of uncertainty sampling is described in Algorithm 15.

Algorithm 15 Uncertainty sampling algorithm

```
procedure UncertaintySampling
input:
    $U$ Pool of unlabelled samples
    $T$ Initial set of labelled samples
    $C$ Classifier
    $n$ Maximal number of requested labels
algo:
    $c_{min} = \infty$                                ▷ Minimal confidence score
    $x_{min} = \emptyset$                              ▷ Sample of least confidence
    while $n < 0$ do
        $C.train(T)$
        for $x \in U$ do
            $(y, c) = C.predict(x)$                    ▷ Class label and confidence
            if $c < c_{min}$ then
                $c_{min} = c$
                $x_{min} = x$
        $y = requestLabel(x_{min})$
        $T = T \cup \{x_{min}, y\}$
        $U = U \setminus \{x\}$
        $n = n - 1$
```

In its simplest form, an SVM tries to discover the best set of hyperplanes dividing the feature space so that samples from one class are separated from samples from any other class by the highest possible margin. It is thus straightforward to compute the distance between any data point and the hyperplane and to infer the degree to which a particular point may be informative to the classifier. The hyperplane is defined by a normal vector w by the equation:

$$\vec{w} \cdot \vec{x} - b = 0 \tag{5.1}$$

The hyperplane is obtained by minimizing $\vec{w}$ subject to $y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1$. Then a vector x is classified by $y_i = sign(\vec{w} \cdot \vec{x} - b)$, with class labels 1 and -1. The distance of a sample to the hyperplane given by $d = \vec{w} \cdot \vec{x} - b$.

In [Osugi et al., 2005], this choice is studied under the exploration/exploitation paradigm: labelling points close to the border between classes is defined as exploitation, as it helps refine the classification. On the other hand, labelling points well within the class boundaries is defined as exploration, as it allows to discover new regions of the feature space belonging to known classes. This is especially important if each class is not a single closed shape in the feature space. A similar trade-off is explored in [Reker et al., 2016].

Uncertainty sampling is not restricted to SVMs, though it requires a measure of confidence that may not exist in all classifiers. Logistic regression and hidden Markov models, for instance, also provide a measurable confidence score [Olsson, 2009]. k-NN classifiers, which are of importance in this work, measure confidence directly as an average distance [Zhu et al., 2008].

Since many topologies of neural networks use a logistic output layer, this means uncertainty sampling can be applied to neural networks as well, though a single label is not enough to be statistically significant: the labels have to be requested in batches [Sener and Savarese, 2018]. In [Wang et al., 2017a], uncertainty sampling is used in a deep convolutional neural network for image recognition. The softmax output layer of the networks associates a probability to each class that can be used in one of the following three ways, depending on the available measures for the classifier:

Least confidence The uncertainty measure is based on the probability of the most probable class; the lower this probability, the more uncertain the sample. The confidence c_i of the i^{th} sample is given by:

$$c_i = max_j p(y_i = j|x_i) \tag{5.2}$$

Margin sampling The difference between the probabilities of the two most probable classes is used as the uncertainty measure; the lower the margin, the more uncertain the sample:

$$c_i = p(y_i = j_1|x_i) - p(y_i = j_2|x_i) \tag{5.3}$$

where j_1 is the most probable label and j_2 the second most probable.

Entropy The Shannon entropy probability of the distribution of classes is used; the higher the entropy, the more *un*certain the sample, thus the uncertainty measure:

$$u_i = -\sum_{j=1}^{m} p(y_i = j|x_i) \log p(y_i = j|x_i) \tag{5.4}$$

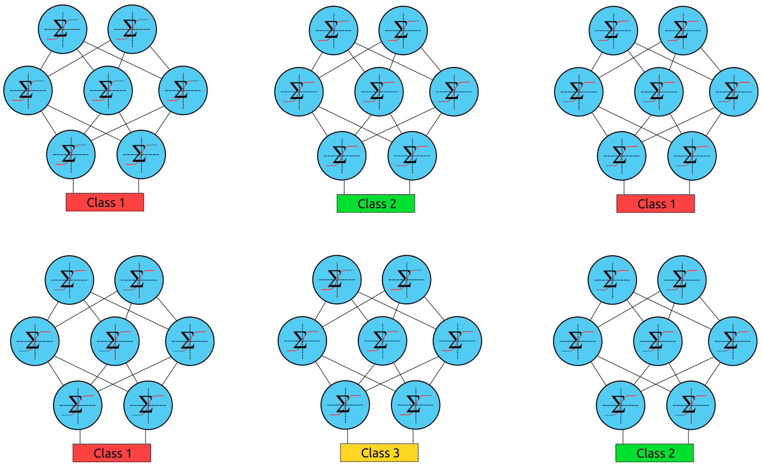


Figure 5.2 – Query by committee example: neural networks trained on the same data with different random initializations predict different classes for uncertain samples.

When c_i is computed, the sample with the lowest c_i is labelled; when u_i is used, it is the most uncertain sample instead. In [Wang et al., 2017a], the three methods are benchmarked and shown to perform equally well.

The requirement for label batches and the difficulty of initially training a neural network with few labelled samples [Wang et al., 2017a] tends to keep the neural network models out of the active learning field. A good example is provided in [Veeramachaneni et al., 2016], where active learning is used to classify the logs of a web server and detect attacks. A neural network is used, but only to detect outliers, and using unsupervised learning. The labels provided by the expert are fed to a random forest supervised classifier, completely independantly of the neural network.

5.1.3 Query by committee

When uncertainty sampling is impossible because the classifier provides no way to measure the confidence in a particular classification, the preferred approach is called query by committee [Bloodgood, 2018]. It consists in training multiple classifiers with the same dataset and detecting samples where they disagree on the label. This process is illustrated in Figure 5.2 and described more formally in Algorithm 16. Note that in the early days of active learning, query by committee was also referred to as uncertainty sampling, which can make the literature confusing, especially in the 1990s [Lewis and Catlett, 1994].

In [Bucurica et al., 2015], a committee of 7 neural networks is initially trained with 10 labelled samples. Then the cross-entropy of the 7 output layers is used as uncertainty score to select more points to be labelled. The authors claim their method achieves performance comparable to fully supervised learning with about 20% of the training set. This result indicates that the very time-consuming process of labelling samples can be reduced five-fold without degradation of the accuracy of the resulting pool of classifiers.

In an earlier example [Saunier et al., 2004], active learning is applied to image analysis in the context of road safety. A naïve Bayesian classifier is applied to a stream of data representing the status of vehicles at an intersection. The training instances are selected using uncertainty sampling, *i.e.* by comparing the outputs of multiple classifiers. In this case, multiple Bayesian models are trained on different initial training sets.

Algorithm 16 Query by committee algorithm

```
procedure QBC
input:
    $U$ Pool of unlabelled samples
    $T$ Initial set of labelled samples
    $C = c_1, c_2, ..., c_m$ Set of classifiers
    $n$ Maximal number of requested labels
algo:
    $c_{min} = \infty$                                    ▷ Minimal confidence score
    $x_{min} = \emptyset$                                 ▷ Sample of least confidence
    while $n < 0$ do
        for $c \in C$ do
            $c.train(T)$
        for $x \in U$ do
            $Y = \{c_1.predict(x), c_2.predict(x), ..., c_m.predict(x)\}$
            $e = H(Y)$                     ▷ Entropy of the vector of predicted labels
            if $e < e_{min}$ then
                $e_{min} = e$
                $x_{min} = x$
        $y = requestLabel(x_{min})$
        $T = T \cup \{x_{min}, y\}$
        $U = U \setminus \{x\}$
        $n = n - 1$
```

5.1.4 Other methods

While the use of query by committee or uncertainty sampling is determined by the type of classifier used and not the problem addressed, additional sampling methodologies can be derived that are domain-specific. In [Malbasa et al., 2017], active learning is used to monitor the voltage stability of the power grid. Contrary to many active learning algorithms, it does not bootstrap the classifier with labelled data. Instead, the voltage time series are used to train an unsupervised classifier that predicts the next measure. Samples generating a high forecasting error are then selected for labelling; these samples are classified into normal and abnormal deviations. This method works for classifiers as diverse as SVMs, neural networks and decision trees. This methods has the advantages of only using samples selected by a model, without bootstrapping, and to operate in real time.

In [Pelleg and Moore, 2005], the classification separates “interesting” from “boring” anomalies. A very interesting feature of this work is that it deals with class imbalance on two different scales: 99.9% of the samples are normal, *i.e.* are not statistical outliers or anomalies. But even among anomalies, 99% of them are irrelevant and only 1% actually carry useful information. Here, the application domain is astronomy, where most anomalies are in fact noise or artifacts. Anomalies are extracted by a Gaussian mixture model. For each component of the model, the samples labelled as belonging to this component are ranked by likehood. The training set presented to the expert is built by selecting the lowest ranking samples from each component in a round-robin fashion. It is up to the expert to classify these anomalies as “boring”, *i.e.*

corresponding to sensor error or software failure, or as "interesting", *i.e.* caused by an actual phenomenon of interest. The datasets used in the paper are part of the UCR time series library[1]. When compared to other selection algorithms in a Gaussian mixture, this method is able to discover all the classes in the Shuttle dataset faster than competing methods and is the only one capable of extracting all the classes of the Abalone dataset. It also learns more classes on the KDD dataset, though it does not discover all of them.

In [Veeramachaneni et al., 2016], active learning is applied to security, and more specifically to intrusion detection. Behavioural analysis is performed by transforming lines of log into time series using counters and boolean indicators, as well as more high-level features such as averages, standard deviations or time elapsed since an event occurred. Outlier detection is then performed on these time series, and the outliers are sent to the expert for labelling. Once labelled, these outliers are used to train a random forest classifier. As samples accumulate, the random forest reaches the accuracy of the human expert, *i.e.* the classification proposed by the random forest (attack or non-attack) is the same as the expert-generated label.

These examples show applications of active learning centered on a particular domain of application, all of which avoid the bootstrap phase by using a domain-specific detector of "interesting" samples. This is also the approach we developed in [Guigou et al., 2017b] for network monitoring: we used alarms from a network monitoring system to detect samples worth labelling and used them to train various classifiers.

5.1.5 Comparison of active learning approaches

A recent paper [Bloodgood, 2018] proposes a set of benchmarks of query by committee versus uncertainty sampling for SVM. It consistently shows that uncertainty sampling, *i.e.* selecting the sample closest to the hyperplane, yields the best performance. This strongly suggests that a single model is better at detecting interesting or difficult samples than a committee.

Interestingly, the same paper compares active learning approaches with boosting and bagging, which are older techniques for combining several weak classifiers into one strong classifier. Bagging – short for ***bootstrap aggregating*** – is a technique for training different classifiers with different random subsets of the dataset. The predictions of the models are averaged during the runtime phase, which provides better stability [Breiman, 1996]. Boosting, on the other hand, uses a chain of classifiers, each one trained on the prediction errors of the previous [Freund and Schapire, 1995]. Both are ensemble meta-algorithms, *i.e.* methods of assembling an *ensemble* of classifiers and combining them. An parallel can be drawn with the query by committee framework: these ensemble methods use fully labelled datasets and use multiple classifiers to maximize accuracy, whereas query by committee uses multiple classifiers on the same dataset to *minimize* label acquisition.

The conclusion of the survey in [Bloodgood, 2018] is that uncertainty sampling yields better classifiers than any of the three ensemble learning alternatives, *i.e.* reaches a better accuracy with the same number of samples, or requires fewer samples for the same accuracy. The metric measuring the number of samples required to reach a certain performance is called *data efficiency* and is maximized by uncertainty sampling, which is also the most efficient method in terms of computational performance, as it only requires a single classifier to be trained, as opposed to an ensemble.

[1] `http://www.cs.ucr.edu/~eamonn/discords`

Given a target level of performance, data efficiency is the ratio of the number of labelled samples in the training set required to reach this performance using active learning *vs.* labelling randomly selected samples [Bloodgood, 2018]. In other words, it quantifies how effective the particular method for choosing points is with respect to random sampling.

5.2 Comparison with semi-supervised learning

Active learning's closest relative is semi-supervised learning, sometimes referred to as "passive learning" in the active learning literature [Bloodgood, 2018, Olsson, 2009, Qian et al., 2017]. Semi-supervised learning, just as active learning uses only a small set of labelled data points. However, the two methodologies differ in three major ways:

- In active learning, the samples are labelled *online, i.e.* as the classifier is being trained. In semi-supervised learning, all labelled samples are available from the start and no expert interaction takes place.
- The points that have an associated label are chosen by the classifier in active learning, but not in semi-sipervised learning: the classifier is trained on whatever labels are available on the target dataset.
- Semi-supervised learning uses the full dataset, including the unlabelled points, whereas active learning only trains a classifier on the labelled part of the dataset.

Direct comparison between the two approaches give a large and consistent advantage to active learning. In [Lewis and Catlett, 1994], active learning achieves performance superior to passive learning with 10 times less samples on a collection of datasets designed to evaluate supervised learning. In [Gal et al., 2017], an active learning neural network reaches state-of-the-art results on the MNIST dataset [LeCun, 1998] using only 1000 out of the 60,000 images of the training set. In [Qian et al., 2017], active learning reaches the same performance as passive learning with only half the number of labelled samples.

In [Zhu et al., 2003], unsupervised, semi-supervised and active learning are combined to exploit as much of the training data as possible: a Gaussian random field model is built; the labelled points have fixed values of strictly 0 or 1 depending on their class, while the rest of the samples have fuzzy labels corresponding to their probability of belonging to either class. The points that minimize the expected error of a Bayesian classifier (Bayesian risk) are then iteratively labelled by the expert and the random field updated. This method shows performance superior to SVM-based active learning and the selection method (risk minimization) also outperforms the simpler uncertainty sampling: the proposed method reaches over 90% accuracy on test datasets in less than 10 samples, while SVM-based active learning fails to reach 80% with 50 samples. On the MNIST dataset of handwritten digits, it differentiates *ones* from *twos* with 98% accuracy using only 5 training samples, while the SVM requires 50 samples to reach 95%.

5.3 Active and reinforcement learning

The reinforcement learning paradigm differs from classical learning, supervised or unsupervised, in that it is built around the concept of interaction with the environment. Instead of trying to iteratively minimize an error, a reinforcement learner, called an *agent*, aims at maximizing a reward that follows its actions in an environment. The environment is acquired through sensors and represented as an input vector, while the action performed is similar to a label. Hence the classifier is presented with a dataset, and tries to draw general conclusions about it; the process is related to active learning in that it requires the *agent* to make decisions that guide its learning process [Mnih et al., 2015].

The typical use case for reinforcement learning, as it was originally devised, it that of a robot exploring an unknown environment. It was later found that this paradigm is, in fact, extremely efficient for teaching an artificial intelligence to play games, *i.e.* Atari video games [Mnih et al., 2013]. This is because the basis of reinforcement learning is the concept of *reward* that follows an action of the agent in the environment, and is very well modeled by the score obtained in a video game.

While reinforcement learning is not directly linked with active learning, the two paradigms combine naturally in a simple scenario: a robot (or any type of autonomous agent) asking a human what to do in a given situation [Lopes et al., 2009, Thomaz et al., 2006]. In particular, active learning comes into play in a particular problem called inverse reinforcement learning: learning a task given the instructions to perform it and inferring the reward function [Ng et al., 2000].

The typical reinforcement learning algorithms are those discussed in Section 4.3: Learning Classifier Systems (LCS) and their extensions, based on the early works of Holland in the 1970s and 1980s [Holland, 1983]. LCS optimize their reward by generating perception-action rules that undergo evolutionary-like transformations (mutations and crossover) [Wilson, 1994]. More advanced models combining reinforcement learning models and deep learning, such as described in [Mnih et al., 2013]. These models use deep neural networks to learn the reward function, or Q. They are known as *Deep Q Networks* (DQN) [Arulkumaran et al., 2017]. DQN are able to learn with a high accuracy the actual reward of actions performed even many time steps in the past [Van Hasselt et al., 2016].

The strongest connection with active learning is the optimizations of classifiers systems to deal with sparse reward, which is equivalent to the limited expert availability. However, where active learning *chooses* the samples to be labelled, reinforcement learning, just as semi-supervised learning, gains reward as a consequence of its decisions and tries to *optimize* this reward. Stated otherwise, the *environment* and the *expert* in reinforcement and active learning do not map beyond the shallow aspect of “what drives the classifier’s learning trajectory”.

5.4 Open challenges

Given the current literature, active learning shows a number of limitations. In [Gal et al., 2017], the authors make the point that a strong mismatch exists between deep learning and active learning, with deep learning requiring very large amounts of labelled data to be efficient and active learning trying to reduce the number of labels as much as possible. Generally, the inability to leverage the most powerful classifiers such as deep neural networks due to the small size of the training sets is still a strong limitation. This limits active learning to algorithms with a smaller parameter space, such as SVM or nearest neighbour models [He and Carbonell, 2008]. In fact, the choice of a classifier for a problem involving active learning can often be guided not by the performance of this type of classifier on this class of problem, but the difficulty associated with bootstrapping the classifier, determining confidence values and retraining the model [Polewski et al., 2015]. This partly explains why SVM have success in active learning: uncertainty sampling with SVM is very simple [Zhu et al., 2008, Tong and Koller, 2001, Tong and Chang, 2001].

In [Das et al., 2017], the authors propose a graphical user interface to support expert-algorithm interaction. This raises a number of points: most research in active learning is only focused on the algorithms, not on the actual way to implement them in real-life scenarios. Many active learning papers [Reker et al., 2016, Akusok et al., 2017] are only benchmarked on fully labelled datasets, thus alleviating the need for actual interaction with an expert.

Another challenge with active learning is that it makes little used of unlabelled data. Typically, the classifier is bootstrapped with some labelled instances on trained on these, then it receives more labelled samples and trains on those. This bears stark contrast with semi-supervised learning, where most of the training is performed on unlabelled data, and the conclusions from unsupervised learning adapted with labelled samples in a second step [Zhu, 2011]. Typically, an active classifier trained on 50 labelled samples from a 1000 point training set never uses the 950 unlabelled samples.

As far as our requirements (see Section 1.3) are concerned, active learning, in its current state, is limited by two factors. The first one is the heuristic used to select the next point to label. The query-by-committee approach can be universal, *i.e.* is feasible with any type of classifier, but is a heavyweight method, as it required to train multiple instances of the same classifier. Uncertainty sampling, on the other hand, is very dependent on the classifier: each classifier can have multiple ways to measure its uncertainty, but no universal uncertainty sampling method exists that can be decoupled from the underlying classifier.

The second limitation is the computational cost. Active learning is generally benchmarked against supervised learning, with evaluations giving results of the form “classifier X trained with active learning can reach the same level of accuracy than traditional supervised learning on dataset Y using only Z% of the labels”. However, only the economy of labels is considered, not the computational efficiency. This is not a limitation of the paradigm itself, but of the evaluations carried out on the classifiers.

The first limitation introduces an undesirable trade-off: to be effective, active learning has to either violate a requirement (low computational cost) or orient the choice of classifier not based on what fits the ***problem***, *i.e.* the type of data considered, but what fits the active learning algorithm, *i.e.* what classifier provides the best uncertainty heuristic. The second limitation should be addressed by benchmarks focussed on the computational cost of various classifiers under active learning.

Part II

The Artificial Immune Ecosystem

In Part I of this thesis, we established the properties and requirements of Immune Systems (IS) and Monitoring Systems (MS):

- Anomaly detection (IS + MS),
- Memory (IS), or failure prediction (MS),
- Tolerance (IS), or low false positive rate (MS),
- Real-time operation (MS),
- Scalability (IS + MS),
- Low configuration overhead (MS).

We also reviewed the literature on the models that could potentially fulfil these requirements. In the conclusions of Chapters 2, 3 and 5, we proposed to address some of the challenges we discovered with these models. In particular, we identified the following problems:

- In monitoring, anomaly detection and prediction,
- In active learning, the lack of a lightweight heuristic to choose points to label, and the general disregard to computational efficiency in favour of label efficiency,
- For time series, scalable real-time anomaly detection,

The model we propose to address these problems is the Artificial Immune Ecosystem (AIE). The AIE is built as a meta-algorithm capable of fulfilling the requirements of monitoring systems as well as immune systems. In particular, it can bring anomaly detection, memory and tolerance to existing monitoring systems.

In Chapter 6, we present the global AIE model, beginning with a simple open-loop model, introducing incremental improvements in the various subsystems and data flow, and finally a more comprehensive model that verifies the immune properties of anomaly detection, memory and tolerance, and fulfilling the requirements of monitoring systems: scalability, real-time operation, low computational cost and low configuration overhead.

In Chapter 7, we describe Cloud-Monitor, a proto-AIE model with a fully feed-forward, open-loop data flow. Cloud-Monitor provides immune memory and anomaly detection and prediction to existing monitoring systems and allows them to detect new types of anomalies, *i.e.* structural anomalies that are not revealed by a thresholding system; however, because of its open-loop structure, it does not verify the immune tolerance property, a shortcoming that manifests itself in the form of a high false positive rate.

In Chapter 8, we study the use of active learning to improve the immune memory of Cloud-Monitor and introduce a notion of immune tolerance. We propose ALPAGA (Actively Learnt Positive Alert GAteway), a model designed to provide tolerance by applying the active learning paradigm to any detection system, without a classifier-specific heuristic. We show that it can lower the false positive rate of a traditional monitoring system and achieve very low computational costs, well below one millisecond per sample.

In Chapter 9, we propose SCHEDA (Sampled Causal Heuristics for Euclidean Distance Approximation), a fast model-free, distance-based anomaly detection heuristic for streaming time series with a linear time complexity, scalable enough for network monitoring.

Table 5.1 – Proposed models and properties

	Cloud-Monitor	ALPAGA	SCHEDA	AIE
Anomaly detection	✓		✓	✓
Memory	✓	✓		✓
Tolerance		✓		✓
Real-time	✓	✓	✓	✓
Scalability		✓	✓	✓
Low configuration	✓		✓	✓

Finally, in Chapter 10, we evaluate the complete AIE model on real data extracted from a monitoring system.

In the following chapters, we will consider the AIE models through the lenses of both network monitoring and artificial immune systems in general, and show how they solve the problems in both domains. The properties verified by each of the proposed models are summarized in Table 5.1.

Chapter 6

Overview of the AIE

This chapter is an in-depth introduction to the concepts that will be developed in the second part of this thesis. It introduces the concepts that constitute its core and their relationships: the immune detection system, immune tolerance system and the fully integrated artificial immune ecosystem. Each one represents a step towards the solution of a particular problem linked to the domains of artificial immunity and network monitoring. Cloud-Monitor, the immune detection system presented in Section 6.1, is the core of the AIE: a real-time monitoring system that simulates the unknown pathogen detection and the secondary immune reaction of the adaptive immune system. It aims at reducing the detection time of an incident. In threshold-based monitoring systems, the time it takes for a metric to reach and cross the alerting threshold might be high, and the incident might be detected hours after its first signs. This in turn leaves less time for the support teams to react and counter the incident before it leads to damageable consequences. Cloud-Monitor can improve the incident detection and resolution time and limit its consequences.

In Section 6.2, we propose an active learning model that introduces a feedback loop between the domain expert and the monitoring system. This process allows the immune detection system, Cloud-Monitor, to become tolerant, and aims at reducing false alarms in support centres. By filtering out false positives, it allows for a sensitive detection step without generating an unacceptably high false positive rate. The model is called ALPAGA, for Actively Learnt Positive Alerting GAteway; it acts as a *gateway* in the sense that it is a filter trained by its users to protect them from false alerts. It classifies time series subsequences that have generated alerts in the detection system, hence the *positive alerting*.

Section 6.4 finally describes the Artificial Immune Ecosystem (AIE) model, which integrates these two subsystems. The AIE uses a notion of immune memory that is common to both detection and tolerance, using Cloud-Monitor to detect incidents faster and ALPAGA to remove the induced false positives.

Note that our focus here is to describe the concepts involved and give the reader a first idea of the overall architecture of the AIE so as to ease the reading of the following chapters, where these models are exposed in great detail and evaluated.

6.1 Immune detection system: Cloud-Monitor

Cloud-Monitor is the core of the AIE: it constitues its detection system. As such, it provides the *anomaly detection* and *memory* properties of immune systems. The Cloud-Monitor model was first published in [Guigou et al., 2015].

A typical monitoring system has a number of polling modules that are able to extract performance metrics from all kinds of networking devices. These metrics are compared against alerting thresholds; an alerting mechanism is triggered when the threshold is crossed. This makes the system highly sensitive to a specific tuning: for each metric, a threshold set too high causes false negatives, while a low threshold causes false positives.

Cloud-Monitor is designed to accelerate the reaction to anomalies by introducing memory and anomaly detection. In this model, alerts can be generated not only by a threshold crossing, but also by a positive match with the signature of an earlier failure or by a deviation from the expected behaviour of the time series. These properties answer the challenges of behavioural analysis and predictive monitoring, as discussed in Section 2.4.

This model can be compared with artificial immune system algorithms. An AIS would typcally encode a data frame into some representation appropriate to compute an antibody/antigen matching, *e.g.* a binary string [Bersini, 2002], then try to match said data frame, or “antigen”, against the full population of “antibodies”, which has a prohibitive computational cost because of the high number of random detectors necessary to avoid underfitting [Stibor et al., 2005]. Cloud-Monitor, on the other hand, only computes matches of a data frame against frames corresponding to *known, i.e.* previously encountered, failure modes. The size of this population is much smaller, hence the computational load is as well. This is possible because, contrary to traditional AIS, Cloud-Monitor does not use random detectors to detect the *first* occurrence of a failure mode: instead, anomaly detection and threshold crossings are used. This removes the need for random detectors to cover the entire feature space.

Cloud-Monitor operates on the raw time series and uses an auxiliary signature database to record failure modes. This database is the equivalent of the immune memory. The processing of a single time series being independant of the others, it can be parallelized and distributed at will. The architecture is depicted in Figure 6.1. It is tiered in three layers: on the first one, the monitored devices, which are arguably not part of the system itself, communicate with the probes, on the second layer, sending health metrics. The probes (P) carry out the analysis and share common information, such as signatures, with the third layer, a bus on which a replicated knowledge base (KB) records these informations; this bus is also used by the probes to send alerts to the expert.

Probes, *i.e.* simple devices or “dumb nodes”, are distributed throughout the network. They can be placed close to the monitored devices, bypassing WAN links and firewalls, or stay in the datacentre. Their role is to collect the metrics and to perform basic analysis. Typically, a probe can monitor up to a few thousands of devices, collecting tens of thousands of time series at a rate varying between one sample per minute and one sample every five minutes. Threshold comparisons, at this scale, are not problematic; complex anomaly detection can become rapidly infeasible [Keogh et al., 2005].

Cloud-Monitor uses threshold crossings just as any other monitoring system, but the actions it takes when a threshold is crossed are different. Not only does it throw an alert, it also records a short window of the time series before the threshold was crossed and stores it in its failure database, *i.e.* the knowledge base in Figure 6.1. Then, whenever a subsequence is found to match

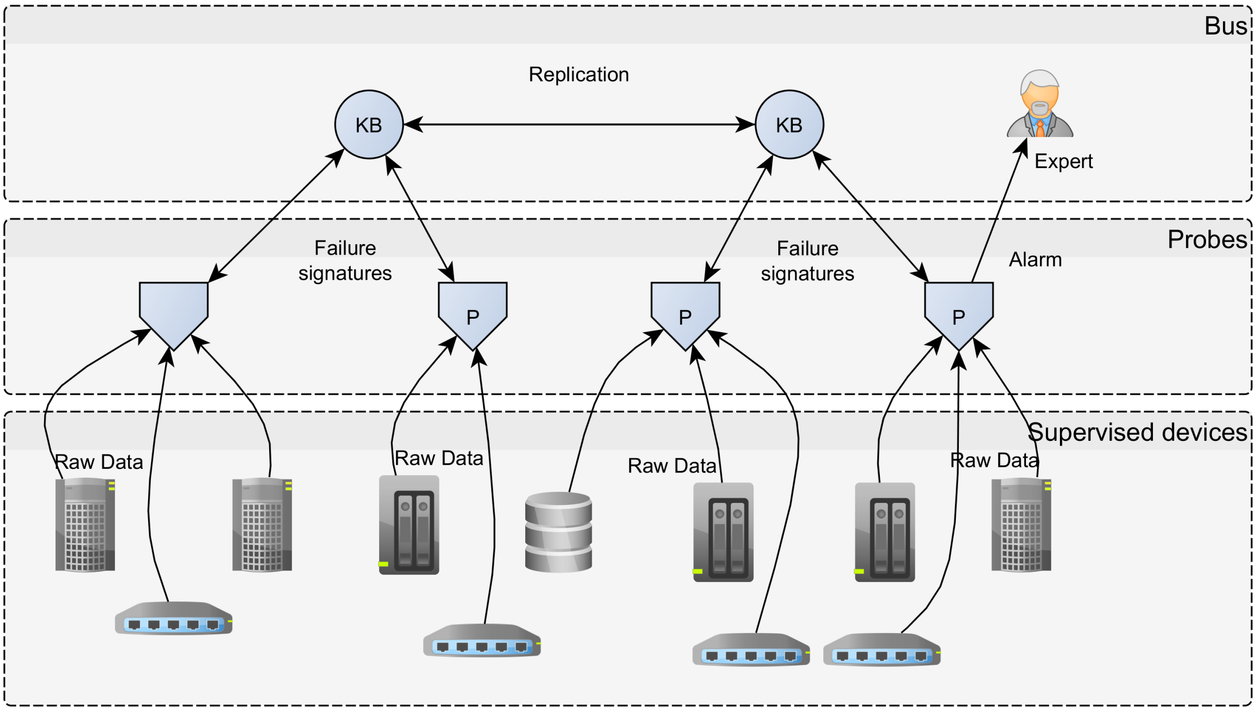


Figure 6.1 – Architecture of Cloud-Monitor

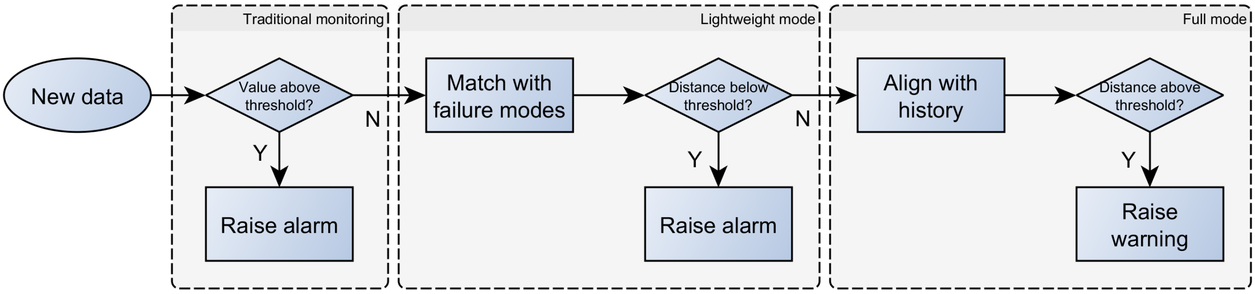


Figure 6.2 – Data flow in Cloud-Monitor

one of these signatures, an alert is raised. This matching is, in our implementation, a distance threshold under the euclidean distance. This allows for a faster reaction to known anomalies or failure modes (in AIS terms, *secondary exposure*), and thus predictive monitoring. A third step uses real anomaly detection to detect new patterns and send alerts in unknown cases as well. In our implementation, anomaly detection is performed by thresholding the euclidean distance; an optimized heuristic for fast anomaly detection under the euclidean distance is proposed in Chapter 9. The data flow is illustrated in Figure 6.2.

Note that this model does not attempt to solve all the problems associated with monitoring systems. In particular, it does not tackle the issues of false positives and hand-tuned thresholds. Its target are anomaly detection and shorter detection delays. The three steps, as depicted in Figure 6.2, can be separately enabled or disabled. The "traditional mode" is similar to any threshold-based monitoring system. Adding the lightweight mode provides an immune memory that can accelerate the detection of known failures; finally, enabling the full mode, which adds the computational burden of anomaly detection, provides the ability to react to unseen patterns.

Cloud-Monitor implements a distributed immune detection system that can be layered on top of a regular monitoring system – a theme we will explore again with immune tolerance. This is important for integration into existing software ecosystems: the traditional mode can be replaced by any monitoring system capable of logging performance data and performing custom alerting actions, *e.g.* Nagios.

Considering the requirements of monitoring systems, Cloud-Monitor implements the following:

- Real-time analysis,
- Anomaly detection,
- Failure prediction.

It does not, however, verify the low false alarm rate requirement.

The cost of anomaly detection and signature matching can be high without the right accelerating structures; however, both can be optimized, as shown in Chapters 7 and 9. However, the increased detection abilities, without regulation, lead to false positives.

The last requirement, low configuration overhead, should be discussed: the traditional mode is based on thresholds, and as such requires just as much configuration as any regular monitoring system. The other two steps of the pipeline, however, only require a sensitivity parameter that can be fixed globally for the entire system – not necessarily for individual time series. As such, the immune memory and anomaly detection steps do fulfil this requirement.

The details of the statistical models used to select the thresholds, the optimization of the signature matching step and the evaluation of Cloud-Monitor are presented in Chapter 7.

6.2 Immune tolerance system: ALPAGA

Cloud-Monitor, as a simple immune memory model, is sufficient to decrease the failure detection time and detect anomalies that do not manifest when only applying alerting thresholds; however, it generates false positives, which breaks one of the essential monitoring requirements. To lower the false alarm rate, we need to complement our immune detection system with an immune tolerance system. The way to do so is to involve the final users, the experts who receives the monitoring alerts, and give them the opportunity to tag false alerts as such. To that end, the immune memory must include labels so that it can determine whether a time series subsequence that triggered an alarm corresponds to a true or a false one. This labelling process is what turns a supervised classifier into an *active* one. The corresponding data flow is depicted in Figure 6.3.

Just as Cloud-Monitor, this new subsystem, which we called ALPAGA (Actively Learnt Positive Alerting GAteway) is layered on top of an alerting system and uses the alert and the corresponding time series subsequence as a basis on which it performs further analysis. ALPAGA was first published in [Guigou et al., 2017b].

By providing immune tolerance, ALPAGA fulfils the requirement of low false positive rate while maintaining real-time operation and low configuration. In particular, it can fit on top of Cloud-Monitor itself, taking its alerts – issued by the underlying monitoring system and by Cloud-Monitor itself – and the historical performance data that preceded the alert as inputs. The basic principle of active learning is to incrementally refine the learnt model instead of requiring either a fine hand-tuning or a labelled dataset; the total configuration required is only the set of hyperparameters of the classifier, which can be extremely small. Low CPU cost is

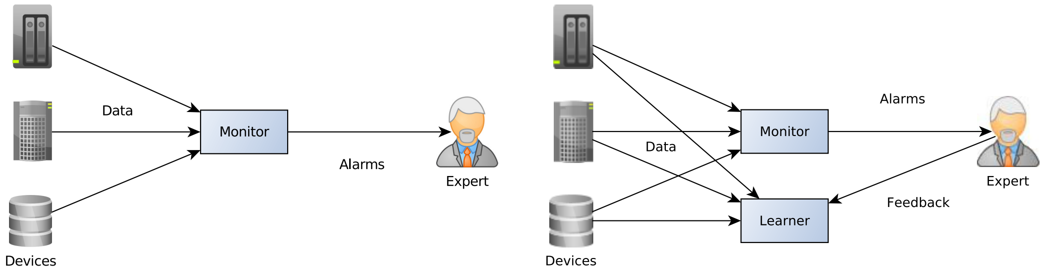


Figure 6.3 – Data flow for basic monitoring (left) and active learning (right)

achieved by exploiting the ability of active learning to reach high accuracy with small training sets [Bucurica et al., 2015].

Indeed, the experiments run in Chapter 8 show that the most efficient classifiers for active learning on time series subsequences are nearest-neighbours classifiers, which require no parameter at all and also have a very low computational cost.

Just as the lightweight mode of Cloud-Monitor, ALPAGA implements a form of immune memory, though for a different purpose: where Cloud-Monitor maintains a repertoire of *reactive* samples, *i.e.* samples that *should* cause an immune reaction, ALPAGA builds a repertoire of *non-reactive* samples, thus providing tolerance to normal phenomena that would otherwise trigger an undesired immune response.

In both cases, the knowledge of which samples should be reactive or not is built up in time, as the system grows. However, two key differences separate the two processes.

Firstly, the memory of Cloud-Monitor is constructed automatically, without any external control or regulation. It simply remembers the instances that caused alerts in the past. The memory of ALPAGA, on the other hand, is built in conjunction with a human operator: contrary to Cloud-Monitor, it uses labels to distinguish true alerts from false ones.

The intervention of such an external agent deviates from the immune metaphor as it is generally understood, insofar as the immune system is generally considered, at least in the AIS community, to be an autonomous system. AIS have mostly been used on labelled datasets or for tasks such as anomaly detection that fall within the category of unsupervised learning. To the best of our knowledge, AIS have never been considered as potential classifiers for active learning. Involving an external agent does, however, offer certain advantages:

- Contrary to an auto-regulation model, or any model using no runtime input, it can be controlled and corrected when anomalous behaviour is detected [Zhang and Nakamura, 2006].
- Its output is not strongly correlated with finely tuned activation levels [Aickelin and Cayzer, 2008, Aickelin et al., 2004].
- It does not require complex signals, *e.g.* danger and aggression levels, to be expressed in the application domain (*i.e.* monitoring) and tuned to acceptable levels [Montechiesi et al., 2015].
- It may capture expert knowledge on-the-fly, without requiring explicit modelling [Tan, 2017].

Secondly, Cloud-Monitor is an open-loop, or feedforward system: the detection of an incident leads to a new signature being generated, which always leads to more detections. ALPAGA, on the other hand, is a closed-loop system: it uses negative feedback to suppress irrelevant, false alerts. A false alert leads to a new signature, which leads to similar alerts being silenced as well. ALPAGA sorts alerts into real ones and false ones and allows most of the false ones to be discarded after their first discovery.

The benchmark of classifiers, the study of the impact of expert availability on the performance of ALPAGA and the detail of its meta-algorithm are exposed in Chapter 8.

6.3 Scalable anomaly detection: SCHEDA

In Cloud-Monitor, the anomaly detection step is performed by comparing the most recent samples to all the history of the time series. This is highly inefficient and would make a naïve implementation of the model unscalable. To overcome this limitation, we propose SCHEDA (Scalable Causal Heuristics for Euclidean Distance Approximation), a set of approximations of the euclidean anomaly score that can be computed at a much lower cost.

The euclidean anomaly score, as used in Cloud-Monitor, is expressed by:

$$\forall i \in \{n, n+1, ..., N\} A_i = min_{j=n}^{N} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{j-k})^2} \tag{6.1}$$

where T is the considered time series, N its length and n the window sized used. The $min_{j=n}^{N}$ requires a full scan of the time series for each i, and there are $N - n$ possible values for i, making the computation of the full euclidean anomaly score time series $[A_n, A_{n+1}, ..., A_N]$ quadratic in complexity.

SCHEDA is a set of 3 heuristics, one also quadratic and two linear, that make this distance computation more lightweight for long time series. It ensures that anomaly detection and scalability can be verified simultaneously. The detail of these heuristics, as well as the benchmarks challenging their ability to approximate the euclidean anomaly score and to detect anomalies in monitoring time series, are presented in Chapter 9.

6.4 The Artificial Immune Ecosystem

The Artificial Immune Ecosystem is the system resulting from the interaction of Cloud-Monitor, ALPAGA and SCHEDA. The first provides memory, the second tolerance and the third anomaly detection – the three main immune properties on which we focus in this thesis. ALPAGA can run on top of Cloud-Monitor, which can itself run on top of any monitoring system. The basic principle is illustrated in Figure 6.4. The feedback loop in which the expert is integrated is highlighted in red. The dotted boxes show the roles of the various contributions presented in this thesis: Cloud-Monitor, for detection, in Chapter 7, ALPAGA, for alert filtering, in Chapter 8, SCHEDA, for fast anomaly detection, in Chapter 9, and finally the overall AIE.

The final model of the AIE, including modifications and optimizations made possible by the results of the following chapters, is presented and evaluated in Chapter 10.

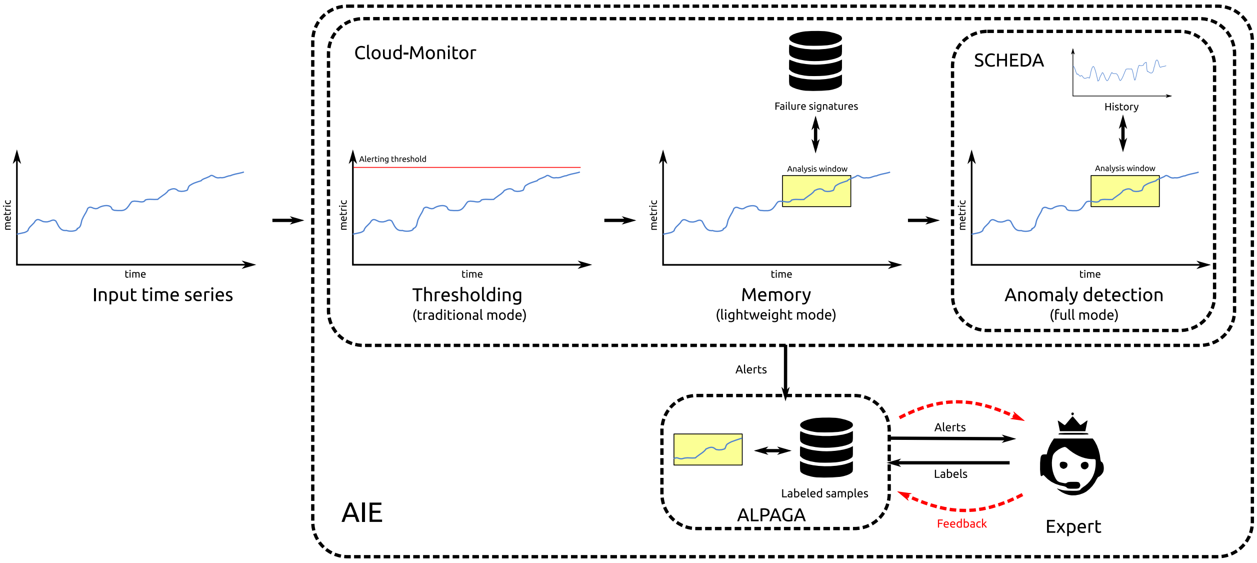


Figure 6.4 – The AIE as a combination of Cloud-Monitor, ALPAGA and SCHEDA

Chapter 7

The immune detection system: Cloud-Monitor

In this chapter, we study the question of whether an Artificial Immune System (AIS) can be used to improve the detection ability of a monitoring system. Behavioural analysis such as anomaly detection is still a challenge in the monitoring community, as we saw in Section 2.4; fast detection is also considered important [Ahmad et al., 2017]. We established in Section 1.4 than anomaly detection and memory are fundamental properties of immune systems; it is our hypothesis that building an AIS for network monitoring can be an answer to the need for anomaly detection and fast response times.

Cloud-Monitor [Guigou et al., 2015] is a proposal for a model that integrates the detection abilities of immune systems, *i.e.* anomaly detection and memory, in the realm of monitoring systems. It enforces these properties by adding an immune-like system on top of an existing monitoring system.

The Cloud-Monitor model aims at reducing the detection time for failures that have already happened at least once in the past. In this regard, it is similar to the notion of secondary response in immune systems. Contrary to negative selection algorithms, it does not try to *anticipate* future failures by generating random detectors; contrary to clonal expansion algorithms, it does not perform hypermutation on the recorded signatures. Instead, it extends threshold-based detection by adding an early recognition system and an anomaly detector. It implements the following properties of immune systems:

- Memory: the immune reaction (alerting) is amplified after a first encounter with a pathogen (failure mode),
- Anomaly detection: new behaviour in the monitored system triggers an immune reaction.

The immune tolerance and adaptation are left out from this first model, which is solely focussed on *detection*.

The rest of this chapter is organized as follows: in Section 7.1, we review the issues with current implementations of artificial immune systems. In Section 7.2, we describe the Cloud-Monitor model and how it enforces the properties of *memory* and *anomaly detection*. In Section 7.5, we show that Cloud-Monitor is able to considerably speed up failure detection, demonstrating the effects of these properties.

7.1 Open problems

One of the identified weaknesses of artificial immune systems is their high computational cost due to the number of detectors required to avoid underfitting the dataset [Stibor et al., 2005]. Whether random feature space partitioning – *i.e.* negative selection – or evolutionary computation – *i.e.* clonal expansion – is used, the model is randomly initialized and trained to fit the data. Instead, we propose a model in which the detectors, or antibodies, are directly derived from the data. This is a particular case of an instance-based classifier, as opposed to a model-based one. Note that not using randomly generated detectors is only possible because the detectors in Cloud-Monitor are not expected to detect *new* anomalies: this is the role of the threshold-based monitoring system and the anomaly detection step. Instead, their role is to rapidly detect patterns that have *previously* been encountered and classified as anomalous.

Model-based artificial immune systems suffer from a high training cost and a high runtime cost. The training cost is high because of the stochastic generation of antibodies and the resulting testing required to maturate them through clonal expansion or negative selection. The runtime cost is also high because each input vector is matched with each antibody, just like an instance-based classifier. In contrast, the training time of a pure instance-based classifier is basically zero, save for the cost of inserting the training vectors into whatever accelerating structure is used, *e.g.* a k-d tree or ball tree (see Chapter 4).

Another perceived weakness of artificial immune systems is their poor representation of input data. The typical matching process of an AIS counts identical bit strings in the binary representation of an "antibody" (detector) and an "antigen" (input vector). If a run of at least n consecutive bits is identical in both, then the matching is positive; otherwise it is negative. Domain-specific representations, such as real-values time series, have to be encoded in a binary scheme partially dictated by the distance function used by the AIS [Strackeljan and Leiviskä, 2008]. Real-valued AIS using the euclidean distance exist but are seldom studied [Montechiesi et al., 2015], mostly because the reference algorithms, such as AbNet [De Castro and Von Zuben, 1999], are only defined for binary representations.

The promise of distributed AIS has also failed: while the distribution of the lymphocytes throughout the body via the lymphatic system was seen as a strength of the immune system that should be transferred to AIS, no such research has been seriously attempted. Distribution is viewed as a side effect of the design of AIS [Hofmeyr and Forrest, 1999], *i.e.* given the computations carried out, the system *could* be distributed. However, this property is never studied nor actually demonstrated in the literature. By contrast, distribution is a *requirement* of monitoring systems, where the computational load would be too high for a single central server. Centralized architectures in monitoring systems, as we saw in Section 2.2.3, are limiting, and the current systems tend towards decentralization.

In the context of network monitoring, detection time is also paramount. Thus the challenge of Cloud-Monitor is to bring detection time down with respect to a simple threshold system while maintaining a distributed architecture and a low computational cost.

7.2 Immune detection pipeline

Cloud-Monitor is an attempt to provide a more lightweight immune-like system for network supervision. As such, it *extends* rather than replaces the more traditional monitoring systems.

The model is a 3-stage pipeline, with each step extending the analysis performed on the collected data: first thresholding, then immune memory matching, and finally anomaly detection. The first step is thus equivalent to any threshold-based monitoring system. The second step uses the immune memory to match the signatures of known failure modes detected by the first and third steps. The third step performs anomaly detection, which compares the subsequence containing the most recent values of the considered time series with its past values. When no similar subsequence is found, an anomaly is detected and inserted in the immune memory.

A typical monitoring system such as Nagios is configured to perform service checks using a specific script for each service; a "warning" and a "failed" threshold are provided as arguments to this script and are configured specifically for each service. Default values can be set based on the documentation of the service under consideration, the manufacturer's datasheet or simply accumulated experience. This knowledge can be used to configure the thresholds of the first step, or an existing monitoring system can be used to provide the raw time series and the alerting state (**OK** or **warning/failed**).

The basic behaviour of a threshold-based system is as follows, with $\mathcal{A}$ denoting the alert condition:

$$\neg\mathcal{A} \wedge T_n > \theta \vdash \mathcal{A} \tag{7.1}$$

$$\mathcal{A} \wedge T_n < \theta \vdash \neg\mathcal{A} \tag{7.2}$$

which reads as: a value superior to the threshold when not in alert conditions triggers the entry into alert condition, and a value inferior to the threshold while in alert condition triggers an end of alarm. For stability, consecutive runs of values superior or inferior to the threshold may be used to trigger a state change. This is standard in commercial monitoring systems and can be expressed as follows:

$$\neg\mathcal{A} \wedge \nexists i \in \{0, 1, ..., m-1\} T_{n-i} < \theta \vdash \mathcal{A} \tag{7.3}$$

$$\mathcal{A} \wedge \nexists i \in \{0, 1, ..., m-1\} T_{n-i} > \theta \vdash \neg\mathcal{A} \tag{7.4}$$

The hysteresis this method introduces avoids false alerts caused by polling errors or transient overloads that are no threat to the device. It will be used in the first step of Cloud-Monitor throughout this chapter, with a confirmation lag of 15 minutes ($m = 15$). This is, once again, the standard for monitoring systems.

7.3 Immune memory

When entering the alert condition, Cloud-Monitor records a signature of the encountered failure in the form of a short subsequence of the time series prior to the event. However, instead of taking the *last* received values, it introduces a gap, as illustrated in Figure 7.1. This gap represents the early warning time during which the first signs of a problem may be visible, but the alerting threshold is not yet reached. Obviously a gap of zero means that the signature should be matched at the exact time when the alerting threshold is crossed, making the whole signature system perfectly useless. On the other hand, too long a gap introduces the risk of

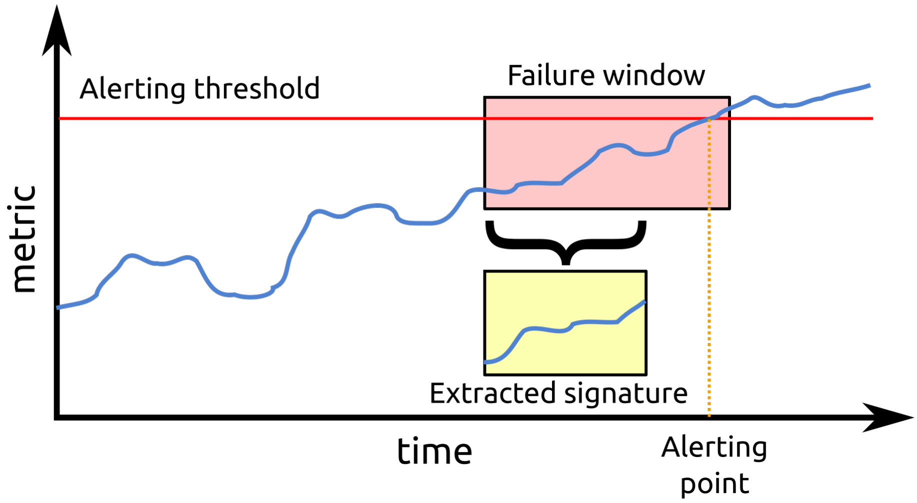


Figure 7.1 – Cloud-Monitor signature recording

matching normal behaviour, ***i.e.*** extract a signature of the ***normal*** behaviour before the failure becomes noticeable. A study of the impact of the early warning gap is discussed in Section 7.5.

A nearest neighbour search is performed at each time step between the subsequence containing the latest data and the signature database. This lookup is depicted in Figure 6.4 in the previous chapter. The search radius is determined by the variability of the signal, which is assumed to have the same characteristics as the signatures, since they are extracted from it. Here we assume that a short subsequence of the considered signal follows a normal distribution. Let us then consider the ***difference*** between two subsequences U and V of length m drawn from a time series T with a standard deviation σ:

$$D = U - V \tag{7.5}$$

$$E[D] = 0 \text{ assuming } U \text{ and } V \text{ have the same mean} \tag{7.6}$$

$$\sigma(D) = \sqrt{\sigma(U)^2 + \sigma(V)^2} \text{ for Gaussian processes} \tag{7.7}$$

$$\sigma(D) = \sqrt{2}\sigma(T) \text{ assuming } U \text{ and } V \text{ have the same variance} \tag{7.8}$$

By squaring Equation 7.8 to get the variance instead of the standard deviation, we see that if $T \sim \mathcal{N}(\mu, \sigma^2)$, then $D \sim \mathcal{N}(0, 2\sigma^2)$. However since we study distances, we are interested in $|D|$, which is a half-normal distribution. By definition:

$$E[|D|] = \sigma \cdot \frac{2}{\sqrt{\pi}} \tag{7.9}$$

$$\sigma(|D|) = \sigma \cdot \sqrt{2(1 - \frac{2}{\pi})} \tag{7.10}$$

Then the distance itself, d, is the sum of all the values of the difference subsequence. By the central limit theorem, d follows a normal distribution:

$$d = \frac{1}{m}\sum_{i=1}^{m}|D_i| \tag{7.11}$$

$$d \sim \mathcal{N}(E[|D|], \sigma^2(|D|)) \tag{7.12}$$

Thus, we can derive the mean and standard deviation of d as a function of σ, the standard deviation of the time series T:

$$\mu_d = \frac{2\sigma}{\sqrt{\pi}} \tag{7.13}$$

$$\sigma_d = \frac{\sigma(|D|)}{\sqrt{m}} = \sigma \cdot \sqrt{\frac{2}{m}(1-\frac{2}{\pi})} \tag{7.14}$$

Finally, we can define a similarity threshold $\delta = \mu_d + a\sigma_d$, a being a sensitivity parameter, to be used as a search radius; any neighbour found within that radius means that the considered subsequence matches a failure signature S from the database DB, and therefore that the alerting state must be entered:

$$\neg\mathcal{A} \wedge \exists S \in DB, \sum_{i=1}^{m}|T_{n-m+i} - S_i| < \delta \vdash \mathcal{A} \tag{7.15}$$

Considering the distribution of d is normal, we can use the 3-sigma rule [Pukelsheim, 1994] to set $a = 3$ as a default setting. In a normal distribution, this ensures that 99.7% of the matches should be detected.

7.4 Anomaly detection

The last detection step of Cloud-Monitor is independant of user-set thresholds and memory: it only uses the recorded history of the metric under consideration. The anomaly detection step allows the monitoring system to trigger alerts based on a behaviour that would significantly differ from the baseline of the signal, ***i.e.*** that does not match its usual recorded characteristics.

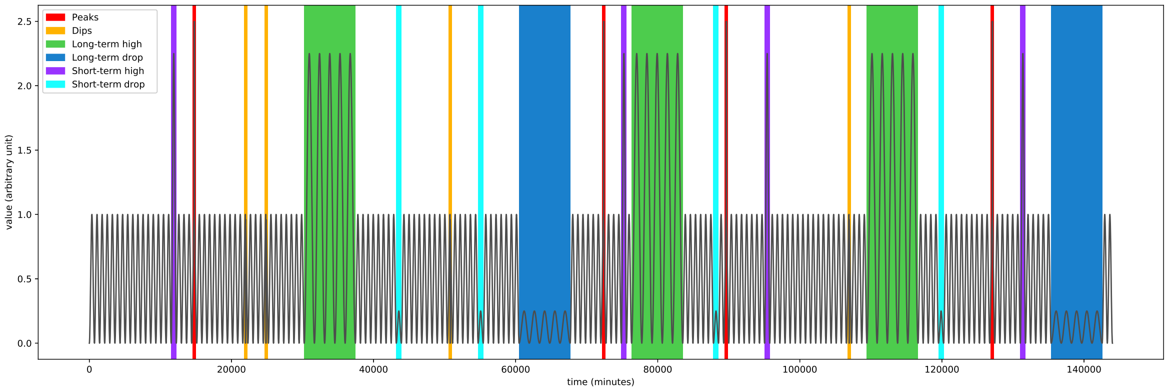
The same equations used to determine the statistical distribution of the distance between a subsequence and a database of signatures give us the characteristics of the distance between the subsequence containing the last samples and any other subsequence of the same time series. The only difference is that the alerting should be triggered by a ***high*** distance instead of a low one:

$$\neg\mathcal{A} \wedge \nexists k \in \{m+1, m+2, ..., n-m\}, \sum_{i=1}^{m}|T_{n-i} - T_{k-i}| < \alpha \vdash \mathcal{A} \tag{7.16}$$

Equation 7.16 reads: ***"An alert should be sent if none is active and there exists no index k such that a subsequence starting at $k-m$, m being the window size, is closer to the current subsequence than a threshold α"***. The anomaly alerting threshold α can be derived from the distribution of distances: $\alpha = \mu_d + b\sigma_d$. As with the immune memory step, we can apply the 3-sigma rule and set $b = 3$ as a first approximation.

Note that trying every possible subsequence, ***i.e.*** every possible value of k, is not efficient. We study inexpensive approximations of $min_{k=m+1}^{n-m}\sum_{i=1}^{m}|T_{n-i} - T_{k-i}|$ in Chapter 9.

Figure 7.2 – Artificial anomalies introduced in a dummy time series



7.5 Evaluation

7.5.1 Methodology

To validate the Cloud-Monitor model, we need to validate the memory and anomaly detection properties it should exhibit. Memory should manifest itself by a decrease in the detection time of a failure if it has previously been observed. Anomaly detection should allow anomalies that do not generate a threshold crossing to be detected. To test these properties, we set up a first experiment using a very simple dummy time series containing only a sinusoid. Then, at very specific point in time, we introduce specially crafted anomalies of one of six types [Mdini et al., 2017]:

- Peak: 15-minute rise above the threshold, just long enough to trigger the alerting,
- Dip: 15-minute fall,
- Short-term high: full period higher than normal, with a threshold crossing,
- Short-term low: full period below normal levels,
- Long-term high: 5 full periods higher than normal, with multiple threshold crossings,
- Long-term drop: 5 full periods below normal.

The series is "at scale": a period contains 1440 samples, just as a day contains 1440 minutes. The test series is plotted in Figure 7.2, with the six types of anomalies highlighted in diferent colours.

For each of the six anomaly types, we measure, depending on the modules activated (only thresholding, *i.e.* traditional mode, thresholding and memory, *i.e.* lightweight mode, or thresholding, memory and anomaly detection, *i.e.* full mode), the number of samples between the onset of the anomaly and its detection. Each anomaly is present four times. The reported time is the average detection lag on the *last three* apparitions. This is done so that the memory can be trained by the *first* occurrence of each anomaly type. While such artificial data are extremely unrealistic, this experiment allows for a precise measurement of the effect of each mode. This would be much harder with real data, that are often too noisy to pinpoint the exact start or end of an anomaly.

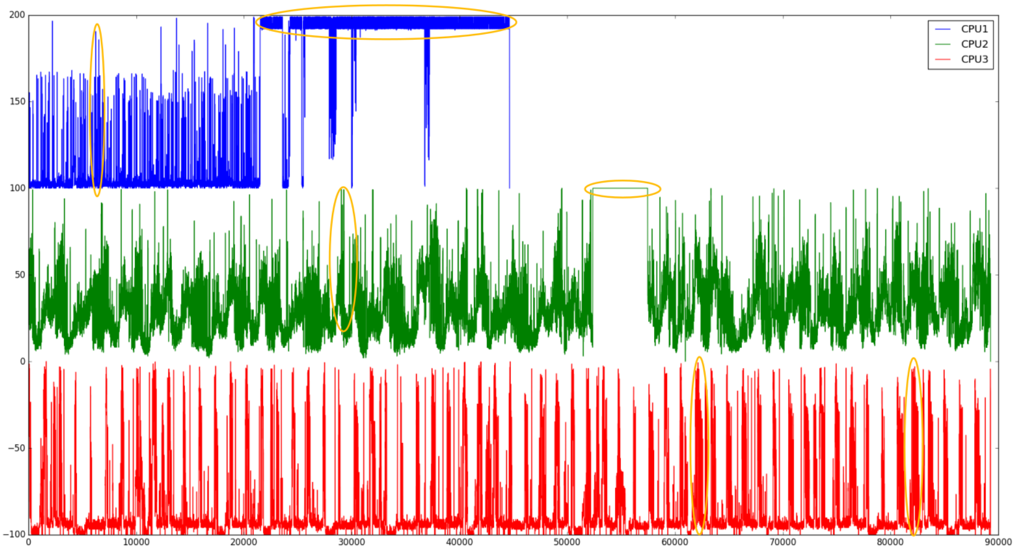


Figure 7.3 – Example time series used to test Cloud-Monitor. Typical anomalies are highlighted in orange.

Table 7.1 – Delay before anomaly detection

Mode	Peaks	Dips	Short highs	Short drops	Long highs	Long drops
Traditional	16	N/D	360	N/D	662	N/D
Lightweight	16	N/D	242	N/D	542	N/D
Full mode	16	23	57	56	61	54

In a second experiment, we benchmark Cloud-Monitor as a real monitoring system. For this we use 32 time series extracted from a monitoring system and hand-annotated with normal and abnormal regions. These regions are aligned on daily boundaries, so each day of data can be labelled as containing an anomaly or not. This makes it simple to measure the true and false positive rates without considering every single sample individually. Some example time series corresponding to measured CPU loads are depicted in Figure 7.3.

7.5.2 Results

Artificial data

In our first experiment, we evaluate Cloud-Monitor on its ability to detect failures faster than a thresholding model. To do so, we use a specially crafted time series in which anomalies are introduced into real measurments at specific time points, and we measure the time lag between the anomaly setting in and Cloud-Monitor triggering an alarm in each of its three operating modes (traditional, lightweight and full). The time lags measured for the three modes described are reported in Table 7.1. The measure is in samples, or minutes; the mention N/D stands for “no detection”, as activity *lower* than usual cannot be detected by a thresholding mechanism.

Table 7.2 – Overall performance of Cloud-Monitor

Mode	Learning strategy	TPR	FPR	AUC	Time
Traditional	None	0.37	0.29	0.53	1.6 μs
Lightweight	Local	0.52	0.45	0.54	11.1 μs
Lightweight	Global	0.83	0.86	0.49	18.7 μs
Full	Local	0.85	0.62	0.62	20.9 μs
Full	Global	0.98	0.95	0.51	27.7 μs

The first obvious result is that, as expected, the types of anomaly (drops) that do not cause a threshold crossing are not detected either in traditional or lightweight mode. We also notice that the peaks are detected in 16 time steps, which corresponds to the hysteresis described in Section 7.2. Both short and long highs are detected about 2 hours (respectively 118 and 120 minutes) earlier in lightweight mode than in traditional mode. The full mode, however, can detect all six types of anomalies and, for the short and long highs, is significantly faster, taking just about an hour (57 and 61 minutes) to detect them.

Real data

In our second experiment, we consider Cloud-Monitor in the context of actual network monitoring, on real data. We record the total running time as well as the alerting data. We compare the performance of the traditional mode with the lightweight mode and the full mode. For the lightweight and full modes, two learning strategies are considered: global and local. The global strategy uses a shared signature database, *i.e.* anomalies from other series are analysed when processing each sample. The local strategy, in contrast, builds one memory base per series. For each settings, we measure the total running time, which is averaged to give a processing time per sample and the true and false positive rates. The plot of the True Positive Rate (TPR) against the False Positive Rate (FPR) define the Receiver Operating Characteristic (ROC) curve. This curve represents the trade-off between false alerts and missed anomalies, *i.e.* how many anomalies are missed at a given error rate (ideally none), or how many false detections occur if all the anomalies are caught (again, ideally none), and how TPR is traded for FPR for a particular configuration of the model. The Area Under the ROC Curve (AUC) defines the overall cost of that trade-off [Aidos et al., 2013]. A perfect detector would have an AUC of 1, while a random detector has an AUC of 0.5. The results are displayed in Table 7.2.

We first notice that, as the more complex modes are activated, the true positive rate increases, meaning more anomalies are detection. However, the false positive rate rises too, and the area under the ROC curve is not always improved. In particular, the global learning strategy, where failure signatures are shared between all the time series, generates so many detections that it can become adversarial in lightweight mode: the FPR is superior to the TPR, and the area under the ROC curve passes below 0.5, which is the score expected for a random classifier. In full mode, the FPR is 0.95, indicating that Cloud-Monitor detects almost every subsequence as anomalous. Using a local learning strategy, however, can significantly improve the AUC, lower the false positive rate while retaining a high TPR. The best result is achieved in full mode with a local learning strategy. The AUC is 0.62, compared with only 0.53 in traditional mode, which is equivalent to a commercial monitoring system. The TPR is 0.85 for the full mode against 0.37 for the traditional mode, which means 85% of anomalies are correctly detected agains

37%. However, the FPR of the full Cloud-Monitor is also more than twice that of the simple thresholding model, meaning it generates about twice more false alerts.

7.5.3 Discussion

These two experiments were designed to test two different aspects of Cloud-Monitor. The first one, on artificial data, validates the hypothesis that an immune-like memory makes failure prediction possible. On average, the lightweight mode detects known errors two hours earlier than the thresholding method. When adding anomaly detection, the full mode is able to detect new types of error (drops, not only peaks) and to detect known errors five to ten hours before the alerting threshold is crossed. This makes it 6 to 11 times faster than the traditional mode, or 4 to 9 times faster than the lightweight mode.

The second one measures the actual performance in production-like settings. Clearly, from the true positive rate, we see that Cloud-Monitor is able to detect more failures or anomalies than the simple threshold model. However, the high false positive rate of 62% (*i.e.* 62% of "negatives" are detected as positive, where a "negative" is a full day without anomaly) can be a problem for monitoring. The high false positive rate generated by anomaly detection in time series is a well-known problem in the monitoring community [Wang et al., 2016b]; in [Laptev et al., 2015], the authors acknowledge it as part of the state of the art and use a filtering step to remove most of the false positives after the anomaly detection step.

Another result worth mentioning is that maintaining a separate memory for each time series yields better accuracy, as measured by the area under the ROC curve, than building a global model. This makes the construction of a distributed architecture easier, as no central database is required.

The CPU cost remains very low: when extrapolated to a 4-core server receiving data points every minute, the 20.9 μs per sample gives a limit of 11.5 million series per node. This is clearly sufficient for the scale of network monitoring.

If we consider the requirements of monitoring systems, as described in Section 1.3, we can state the following:

- *Scalability:* Cloud-Monitor can scale to millions of series per node.
- *Real-time analysis:* Each sample is analysed immediately, without buffering, and the memory and anomaly detection make the detection much faster than what a thresholding system can achieve.
- *Anomaly detection:* The full mode has been shown to be able to detect anomalies even when they do not generate a threshold crossing.
- *Failure prediction:* The immune memory and anomaly detection make it possible to detect failures before a threshold is crossed.
- *Autonomy:* The alerting threshold of each series has to be defined; however, it is only required for the first step of the analysis, which can be carried out by an already existing – thus already configured – monitoring system.
- *Low false alarm rate:* The false positive rate of Cloud-Monitor is over twice that of a traditional monitoring system, which does not fulfil this requirement.

7.6 Conclusion

Cloud-Monitor has been shown to fulfil five of the six requirements of monitoring systems. It allows incidents to be predicted or detected before a threshold crossing makes them obvious and does so at a very low computational cost. Il also detects typologies of incidents that traditional monitoring is unable to. The increase in false positive rate, from 29% in traditional mode to 62% in full mode, needs to be addressed to make it more usable in a real network monitoring situation. This is the object of the next chapter.

Our initial question was whether using the immune system as a basis to construct a monitoring system with memory and anomaly detection abilities could be an answer to the challenge of behavioural analysis, and whether it could accelerate the detection process. Cloud-Monitor has been demonstrated to detect anomalies that could not be picked by traditional monitoring systems and to improve the detection time of those that could. In this respect, it has shown that the properties of immune systems can be a sound basis for monitoring.

Chapter 8

The immune tolerance system: ALPAGA

One of the difficulties in network monitoring, especially when using anomaly detection, is the high false positive rate [Wang et al., 2016b]. In this chapter, we introduce ALPAGA (Actively Learnt Positive Alert GAteway) [Guigou et al., 2017b], a gateway, or filter, that allows the experts, when a monitoring alarm is raised, to accept or dismiss it in such way that the generated knowledge is reused, and similar false alerts are filtered and removed over time as the knowledge base grows. Essentially, ALPAGA is an active learning meta-algorithm. Active learning is discussed in detail in Chapter 5.

If we come back to the main properties of immune systems, the Cloud-Monitor model discussed in the previous chapter, with its failure signature database and anomaly detection, is able to generate a high volume of alerts; it embodies the anomaly detection and memory properties of the immune system, but lacks the immune tolerance property. As Cloud-Monitor was designed to add memory and anomaly detection to existing monitoring systems, ALPAGA's purpose is to provide tolerance to monitoring, the same way immune tolerance regulates the immune system and prevents auto-immune diseases. In the context of monitoring, this means reducing the false alarm rate of an existing system. The equivalence between immune tolerance and false positive rate reduction has been studied in [Kim and Bentley, 2002].

The main questions ALPAGA attempts to address are how to levereage classifiers for time series subsequence classification under active learning, and how to identify suitable classifier models for efficient active learning. *Active learning* implies many classifier retrainings and generally small datasets, which can result in high computational costs and underfitting, *i.e.* the classifier not correctly learning the distribution of the dataset because of its too small size.

ALPAGA is designed as a shim, a component inserted between a monitoring system and its final user. It has been independently designed, *i.e.* not specifically within the Cloud-Monitor or the AIE framework, and thus is evaluated separately. The combined use of ALPAGA and Cloud-Monitor is discussed in Chapter 6 and studied more in depth in Chapter 10.

The rest of this chapter is organized as follows: Section 8.1 summarizes the challenges faced by active learning in the context of the AIE.Section 8.2 studies the role of the human expert in active learning and in monitoring systems. Section 8.3 describes the proposed meta-algorithm, ALPAGA. Section 8.4 presents and discusses the evaluations of ALPAGA. Section 8.5 concludes this chapter.

8.1 Open problems

Although Cloud-Monitor is able to reduce false negatives and detection lag, it has no way of influencing false positives. False positives are generally accepted as part of the trade-off when performing anomaly detection [Wang et al., 2016b] and are later filtered in a secondary step, either by correlating various events happening at the same time [Begnum and Burgess, 2007], by using multiple anomaly detectors [Wang et al., 2016b] or by querying the end user [Laptev et al., 2015], which is the approach we take with ALPAGA. This type of real-time interaction between a user and a classifier is a heuristic-based form of active learning (see Section 5.1.4).

A number of challenges are associated with such a task, mainly because alerts are a rare phenomenon, and thus only have a few examples:

- The classes (normal/abnormal) of the time series subsequences are highly unbalanced.
- Classifiers are expected to work on high dimension data (time series subsequences) with only a small training set.
- The total computational cost should remain small.

8.1.1 Imbalanced classes

The usual way of dealing with imbalanced datasets is to artificially balance them, either by undersampling the majority class [Liu et al., 2009], oversampling the minority class [Mi, 2013], or both. However, undersampling at random results in a loss of potentialy important points, while oversampling increases the size of an already large dataset. Our approach is to selectively undersample the majority class, i.e. the *normal* class, by only considering those samples that generate a false alarm. In this regard, we use the underlying monitoring system as a heuristic to determine on which points to train the classifier.

It is a well-known result in active learning that not all points carry useful information to determine the inter-class border, and that selecting points with a high discriminatory power in a learning set is paramount [Osugi et al., 2005, Reker et al., 2016]. This strategy is the basis of the active learning approach. A small set of well-chosen points can carry the same information, and hence train a predictor of equal performance, as a large set of randomly selected points [Mi, 2013]. Points contribute to the training when they are close to the border between two classes. Conversely, a point far from the border is of little help when it comes to discriminating a sample. This problem is highlighted in [Akbani et al., 2004] as a cause for boundary skew in SVMs.

8.1.2 Small training set

Because alerts are rare, and the experts have a limited availability to label samples, the training set of the classifier used in ALPAGA will necessarily be small. This is generally the case in active learning (see Chapter 5) and has been shown to limit the usability of certain types of models such as deep neural networks [Wang et al., 2017a]. Conversely, others such as naïve Bayesian models have been shown to perform well with small samples sizes [Forman and Cohen, 2004]. We compare the effectiveness of multiple types of classifiers in Section 8.4.4.

The other factor to be considered is expert availability. This is an important issue in active learning [Wang et al., 2017a], where one of the main goals of the research is to reduce the number of annotations, *i.e.* labels, queried from the experts. We study in Section 8.4.3 the impact of having a limited access to the expert.

8.1.3 Low computational cost

We have already established in Section 1.3 that a low computational cost is an important requirement in network monitoring. This is one more constraint that must be taken into account when designing a system that will be used for this purpose. It creates yet another incentive to work on small datasets with only a few classifiers. In particular, it rules out the query-by-committee approach to active learning, as training multiple classifiers on the same data leads to a duplicated effort and a multiplication of the computational cost. Other approaches like uncertainty sampling, that only require *one* classifier, are more efficient. This is a problem that is rarely studied in the literature, where the *efficiency* or the *cost* of active learning are measured by the number of labels it uses, not the computational overhead it generates [Wang et al., 2017a, Thiam et al., 2017].

8.2 Role of the expert

While machine learning has been used in the past to learn patterns and to discover anomalies, most implementations work under the assumption that either labelled – or at least partially labelled – data is available (supervised learning, *a.k.a.* classification), or that no label at all can be obtained (unsupervised learning, *a.k.a.* clustering). The former requires a field expert to manually go through the painstaking process of actually labelling the comprehensive time series [Das et al., 2017], while the latter uses no expert knowledge at all, relying instead solely on structural variations to detect change [Dasgupta and Forrest, 1995].

In many applications such as network infrastructure monitoring, experts may not be available for the batch job of building a labelled dataset; anomaly detection software may also be available, though having a rather low accuracy [Chowdhury et al., 2014] or a high computational cost [Wang et al., 2016b, Macit et al., 2016].

The role of field experts in network monitoring is twofold. Firstly, when the monitoring is set up for a new host or device, the expert tunes the alerting thresholds of the polled metrics for this device. Secondly, when the monitoring system raises an alarm, the experts have to determine whether that alarm corresponds to an actual incident or is a false alert. The first diagnostic tool is the time series corresponding to the values of that series over some time, *e.g.* a week[1]. However, once the alert has been classified as true or false, this label is not reused: there is no way for the monitoring system to know that it raised a false alarm.

This is where a property like immune tolerance can have an impact: alerting is the equivalent of an immune reaction for a monitoring system and, just like an immune reaction, it needs a mechanism to suppress it. ALPAGA's aim is to provide such a mechanism by using the labels created by the experts as they receive and classify alerts to learn the false alerts' patterns and not pass them along when they are detected.

Note that this part of our research is only focused on the false alarm events. We have shown in Chapter 7 that enabling a form of immune memory and anomaly detection reduces the false *negative* rate – *i.e.* increases the true positive rate – enough so that it does not become the main

[1] This is why all commercial monitoring systems have an interface to display historical data.

issue. ALPAGA is not a detection system: it can only act on the false *positive* rate by silencing false alerts. An event or anomaly not detected by the underlying monitoring system can in no way be detected by ALPAGA.

8.3 The meta-algorithm

In order to rebalance the classes and avoid dealing with a very high number of points – potentially one per minute and per time series – that no human could possibly label, ALPAGA queries labels on subsequences that have caused an alert. This represents a small dataset that can reasonably be expected to be labelled by network operators, since they *already* deal with these alerts anyway.

ALPAGA classifies the short subsequences of a time series that have triggered an alert as true or false positive. In the initializating phase, it lets all alerts through and learns the classification thanks to the expert feedback (accept or dismiss the alert). In the running phase, new alerts pass through the gateway and those that are classified as false alerts are automatically dismissed. While any classifier could, in theory, be used, not all perform equally well. We benchmark various candidates in Section 8.4.4.

ALPAGA is capable of building a training dataset online, *i.e.* while operating and sending alarms when required, to use expert feedback to label this set and to train a classifier in order to filter out the false alarms. When the dataset is large enough and the classifier is trained, it is used as a predictor to decide whether an alarm should be raised or ignored, *i.e.* to classify it as a true or false alarm.

Algorithm 17 The ALPAGA meta-algorithm

```
procedure ALPAGA(T raw time series, N lookback, k training set size, C classifier)
    Initialize empty training DB
    while $|DB_{true}| < k$ and $|DB_{false}|wk$ do
        for all alarm do
            Send alarm to expert
            Collect previous N samples from T
            Get expert feedback (true or false alarm)
            Add tuple (samples, label) to DB
    Train C on training DB
    for all alarm do
        Collect previous N samples from T
        if Predicted class for samples is TRUE then
            Send alarm to expert
        else
            Discard alarm
```

The performance gain is obtained by pre-selecting the learning set. When enough alarms have been classified, a number of points of the time series have been consumed. We show experimentally that learning on the small pre-selected dataset yields better performance than learning on all consumed points. Empirically, about 10,000 points are required to provide a roughly balanced dataset of 20 to 50 points per class, *i.e.* 20 to 50 false alerts and 20 to 50 real ones.

8.4 Evaluation

In the following, we present an evaluation of ALPAGA in network monitoring scenarios. In particular, we study the following properties:

- *Data efficiency:* how does the availability of the expert, *i.e.* the maximal number of labels, impact the performance of ALPAGA?
- *Universality:* does ALPAGA perform equally well with all types of classifiers?
- *Computational cost:* how much CPU time does ALPAGA require to classify a time series subsequence?
- *Learning transfer:* can a classifier trained on one series correctly classify another one?

We also compare ALPAGA with supervised learning, *i.e.* with an expert capable of labelling each possible subsequence.

8.4.1 Meta-algorithm implementation

We implemented an ALPAGA prototype as 3 interacting components: a monitoring system, *i.e.* a system capable of reading time series data and raise an alert if ont point crosses a given threshold, a simulated expert applying rules to classify the alarms, and a classifier. Data is fed into the monitoring system, which calls the expert when an alarm occurs and sends him the last 15 minutes of data. The expert's decision is recorded and associated with this data. Six different classifiers are tested: a nearest-neighbour classifier, two variants of decision trees, a naïve Bayesian model, a neural network and an SVM. Their performance is studied in Section 8.4.4.

8.4.2 Settings

We first tested our system on CPU load time series. The load is sampled every minute for two months, yielding a total length of about 90,000 real-valued data points between 0 and 100%. To simulate real-time acquisition, we process the points one by one, without any batch processing. To evaluate our approach, we use the following rules:

- The alerting threshold of the monitoring system is set at 80%: if the recorded CPU load goes above 80%, then an alert is generated.
- The alerts and the associated last 15 minutes of data are then sent to the simulated expert, which applies a simple rule: if the *mean* of the 15 minutes as above 90%, then the alert is considered real, otherwise it is considered false. This is also the ground truth used to distinguish between true and false alerts at the output of ALPAGA.

The dataset used is a subset of that used to evaluate Cloud-Monitor in the previous chapter. The full dataset contains many different metrics; in this test, we only use the CPU load time series. An example is shown in Figure 7.3.

Table 8.1 – Impact of limited expert availability

Max. expert calls	20	30	50	100	500	Unlimited
F-score	0.36	0.70	0.72	0.72	0.78	0.78

Since any anomaly detection dataset is naturally heavily biased towards “normal” behavior, a metric such as accuracy is unable to efficiently discriminate between a good and a poor classifier: always predicting the “normal” class is sufficient, in our case, to reach a 90 to 95% accuracy. Instead, we use the F-score, defined as the harmonic mean of precision and recall, to evaluate the performance of our various settings. The F-score has been shown to reflect the performance of a classifier on a highly imbalanced dataset better than other metrics [Chinchor and Sundheim, 1993]. An alternative is the area under the ROC curve, as used in the previous chapter; however, it is has been shown on highly imbalanced datasets to be less informative, *i.e.* to make the difference in performance between different classifiers less visible, than the F-score [Saito and Rehmsmeier, 2015].

8.4.3 Data efficiency

We first evaluate the importance of having a highly available expert, *i.e.* a large training set. In a real scenario, it is unrealistic to expect a human expert to deal with thousands of errors. The limited availability of the expert is simulated by stopping his feedback after he has classified a fixed number of alarms as real or false. The results are summarized in Table 8.1. The classifier used here is a nearest neighbour classifier. We observe a rapid increase in F-score, from 0.36 with 20 points to 0.70 with 30. The rise is less spectacular afterwards, with an F-score increase to 0.78 at 500 points and no further improvement when the expert is always available. We observe no phenomenon of overfitting, as expected given the size of the training set.

8.4.4 Universality

In this section, we compare the performance of various classifiers within the ALPAGA meta-algorithm. We measure the F-score and running time of six classifiers on the dataset discussed in Section 8.4.2. The total running time is divided by the number of samples of the time series in order to give a computational time per point, which is presented in the next section. The classifiers studied are:

- k-NN: a simple, well-known nearest neighbour classifier,
- Naïve Bayesian: a very versatile model that has been shown to work well with small training sets [Forman and Cohen, 2004],
- Decision trees: an efficient rule-bases system, see Section 4.3.1,
- Extremely random trees: an ensemble method to improve decision trees [Geurts et al., 2006],
- Neural network: though not known to work well with small training sets, neural networks are known for their generalization ability,

Table 8.2 – F-score and percent improvement over the detector for each predictor and training set size

Size	k-NN	Naïve Bayesian	Decision trees	Extra trees	Neural net	SVM
10	0.719 / +15.41%	0.104 / -83.38%	0.727 / +16.69%	0.724 / +16.18%	0.548 / -12.12%	0.361 / -42.11%
20	0.724 / +16.16%	0.18 / -71.18%	0.713 / +14.44%	0.753 / +20.82%	0.43 / -31%	0.252 / -59.56%
30	0.796 / +27.79%	0.084 / -86.49%	0.767 / +23.04%	0.739 / +18.56%	0.448 / -28.17%	0.236 / -62.2%
40	0.8 / +28.35%	0.168 / -73.05%	0.785 / +25.95%	0.739 / +18.66%	0.375 / -39.86%	0.238 / -61.77%
50	0.791 / +26.97%	0.174 / -72.07%	0.797 / +27.94%	0.801 / +28.48%	0.359 / -42.46%	0.249 / -60.02%
60	0.78 / +25.1%	0.271 / -56.45%	0.748 / +20.05%	0.761 / +22.06%	0.378 / -39.36%	0.272 / -56.39%
70	0.773 / +24.06%	0.275 / -55.85%	0.763 / +22.51%	0.766 / +22.94%	0.463 / -25.71%	0.288 / -53.74%
150	0.81 / +30.03%	0.586 / -5.9%	0.731 / +22.51%	0.726 / +16.55%	0.319 / -48.85%	0.467 / -25.13%
250	0.791 / +26.86%	0.627 / +0.67%	0.753 / +17.33%	0.688 / +10.42%	0.391 / -37.24%	0.514 / -17.52%

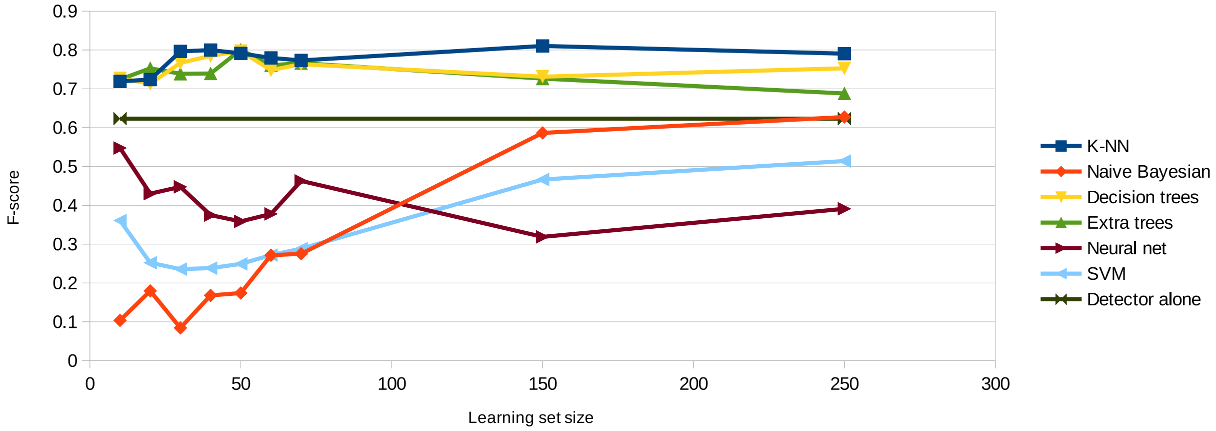


Figure 8.1 – Comparison of F-score with various classifiers and learning set size

- SVM: very versatile classifier with good generalization abilities in high dimensional spaces [Tan et al., 2010].

Table 8.2 summarizes the F-scores obtained for each of these classifiers when the training set size varies between 10 and 250 points per class (true of false alarm). We measured the F-score of the monitoring system on its own at 0.62, thus the table also includes the percentage of improvement of ALPAGA, with each given classifier and training set size, over the monitoring system alone. The same results are plotted in Figure 8.1.

The F-score figures show an obvious divide between on the one hand the naïve Bayesian classifier, neural network and SVM and, on the other hand, the two variants of decision trees and k-NN. The first three remain below the performance of the monitoring system, while the rest are consistently above. We also notice that, for the three best performing classifiers, the F-score is only weakly correlated with the training set size. The absolute best score is 0.81, reached by a k-NN classifier with 150 points per class; however, a very similar score of 0.80 is obtained by the same classifier at 40 points, or by the extremely randomized trees at 50 points.

8.4.5 Computational cost

The computational cost of ALPAGA, depending on the classifier used and the number of samples in the training set, was measured in the previous experiment. Table 8.3 contains the running time per point of each classifier, in microseconds; they are displayed in Figure 8.2.

Table 8.3 – Classifier running time (μs) vs. training set size

Size	k-NN	Naïve Bayesian	Decision trees	Extra trees	Neural net	SVM
10	82.0	39.9	36.7	36.4	43.4	35.6
20	196.0	41.6	36.1	36.2	43.2	35.9
30	194.6	42.9	38.4	35.7	42.9	36.1
40	191.6	41.9	36.6	37.7	42.7	37.3
50	192.6	42.3	37.7	37.1	43.8	38.3
60	199.2	42.8	39.0	37.8	42.1	38.5
70	195.9	44.4	37.8	38.2	44.3	39.1
150	201.3	54.1	42.8	41.6	48.1	43.9

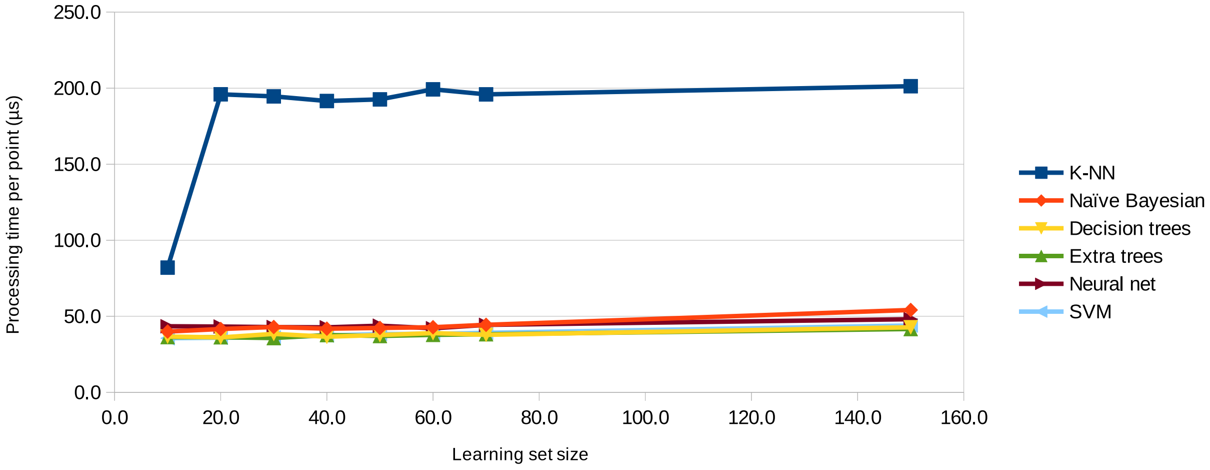


Figure 8.2 – Classifier running time vs. training set size

In terms of running time, the k-NN classifier is noticeably more costly than its five competitors, with processing times in the order of $200\mu s$. The others vary between 35 and $55\mu s$, with only slight variations between them and with respect to the training set size.

8.4.6 Learning transfer

In more complex settings, we wish to train a single classifier to be reused on different time series with similar range and "shape", such as CPU loads from multiple servers and devices. We examine the performance of a detector trained with one series on another.

We switch to a different classifier, trading off speed for generalization and accuracy. The following experiment uses a k-NN classifier (with $k = 5$). We conducted a series of benchmarks using 3 different CPU load time series recorded over 2 months from different devices, each time training ALPAGA on one and evaluating the classification on the other two. The results are shown in Table 8.4.

As we saw in the previous experiment, the monitoring system, on its own, has an F-score of 0.62. ALPAGA, with a classifier trained on any of the 3 series, performs very well (F-score above 0.99) on the CPU1 and CPU2 series. On the CPU3 series, however, classifiers trained on CPU1

Table 8.4 – Results of classifier reuse: F-score vs testing/training series

Train Test	CPU1	CPU2	CPU3
CPU1	—	0.991	0.997
CPU2	0.996	—	0.999
CPU3	0.323	0.158	—

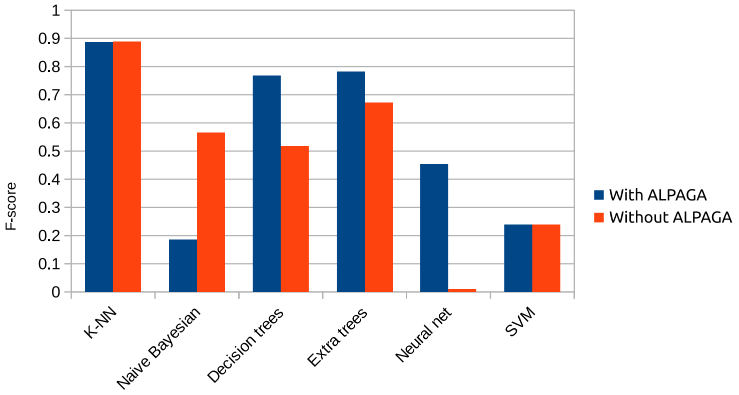


Figure 8.3 – F-score with ALPAGA preselection (40 points labelled by expert) and supervised learning (first 10,000 points labelled)

and CPU2 perform poorly (F-scores of 0.323 and 0.158 respectively), worse than the monitoring system alone.

8.4.7 Comparison with supervised learning

We then study the performance gain provided by ALPAGA over supervised learning. To this end, we set the training set size to 40 samples per class, which we found to be optimal, and compare the performance of the classifier when trained with the points selected by ALPAGA and on the first 10,000 points of the raw time series, *i.e.* over 100 times more points. Generating the corresponding amount of labels is possible thanks to the simulated expert. The results are summarized in figure 8.3.

This experiment shows that, with the exception of the naïve Bayesian classifier, all classifiers perform equally well or better when trained on a small, relatively balanced dataset selected by the detector than on the original time series. This is very apparent on the two decision trees models and spectacular on the neural network, that shows an F-score below 0.01 in supervised learning mode and 0.45 with the ALPAGA active learning meta-algorithm.

8.4.8 Discussion

From these results, we can start drawing conclusions about the data efficiency, universality and computational efficiency of ALPAGA. We also discuss its ability to transfer learning and compare it to classical supervised learning. Finally, we summarize the limitations of the model.

Data efficiency

The first experiment with expert availability, as summarized in Table 8.1, shows that the performance of a monitoring system can be greatly improved by a relatively small amount of expert feedback, from an F-score of 0.62 to 0.72 with only 50 labels. However, increasing the number of labels quickly reaches a region of diminishing returns: moving from 50 to 100 increases the F-score up to 0.78, but no further improvement is achieved after 100 labels. This behaviour is confirmed in the second experiment (see Figure 8.1) for the nearest neighbour and decision trees classifiers. The SVM and naïve Bayesian models also appear to slow down their performance increase after 250 labels. The neural network shows only a small correlation between training set size and F-score. The ideal number of labels for the best models, *i.e.* nearest neighbours and decision trees, is around 50 samples per class, or 100 samples total – which translates into 0.11% of the total size of the time series.

Universality

The support of various classifiers was studied in 8.4.4. To give a meaning to those results, we compare them to the performance of the detector alone, which yields an F-score of .62. The results in table 8.2 show that k-NN and the extremely randomized trees (extra trees) metaheuristic [Geurts et al., 2006] are most suited, with an optimal learning set size of 40 to 50 points per class. However, other algorithms, such as the naïve Bayesian classifier, neural network or SVM classifiers, can perform worse than the thresholding model. Fine tuning may be required to adapt them to ALPAGA.

This can be explained by the nature of ALPAGA: it is a filter, thus it removes alerts from the output of the monitoring system. If more true positives than false positives are filtered, then the performance drops. This is a case of underfitting due to the small size of the training set.

Computational efficiency

The experiments show that ALPAGA can provide an important boost in classification for a very limited computational cost. All classifiers, whatever the size of the training set, require less than a millisecond per point, training time included; with the exception of k-NN, they only require about $50\mu s$ to process a single point. This translates into a processing ability of 20,000 points per minute per core for the server used in the benchmarks, using an Intel Xeon clocked at 2.2 GHz. In a 4-core server, this allows for 80,000 time series in parallel, which is compatible with the scalability requirements of the AIE.

Learning transfer

From the results in Table 8.4, we can see that ALPAGA performs very well on series statistically similar to the one it was trained on, but also that some notion of “difficulty” exists: the CPU3 series is only poorly predicted by an ALPAGA instance trained on other series, but an instance trained on CPU3 successfully predicts the other two. These results indicate that some generalization ability exists, but it does not seem to be perfect, and the degree to which learning transfer can occur should be further studied.

Comparison with supervised learning

When compared with a classifier trained on the raw time series data, with the exception of the naïve Bayesian classifier, ALPAGA consistently provides a substantial boost, as depicted in figure 8.3. This confirms that ALPAGA provides an important gain not only in computational cost, but also in accuracy, over a naïve use of a standard classifier on the raw time series. In this case, it performs better than the classifier alone with a training two orders of magnitude smaller. This makes ALPAGA a potentially powerful tool to deal with large unbalanced datasets.

Limitations

One limitation we just mentioned is the underfitting problem: some classifiers simply cannot learn a distribution with only a few dozen samples of each class. We have shown that k-NN and decision tree classifiers can; naïve Bayesian models can provide a slight boost over the raw monitoring system with 250 samples per class, while SVM and neural network do not reach such level at all.

Another limit is that ALPAGA is only trained once: after enough data have been collected to form a training set, the classifier is trained and becomes active as an alert filter (see Algorithm 17). Thus if the time series changes significantly in time, *i.e.* exhibits concept drift, then the classifier does not learn the new distribution. This could be addressed by keeping a running training set, discarding the oldest samples and retraining the classifier regularly. However, this implies that the expert is also queried for more labels.

8.5 Conclusion

The ALPAGA model has demonstrated its ability to perform such filtering at a low computational cost, well below one millisecond per point. The benchmarks show that even very simple algorithms, *e.g.* nearest neighbour classifiers can provide a significant false positive reduction on time series data. In fact, they show that the simplest models outperform the more complex ones on small training sets.

We have shown that ALPAGA is extremely data-efficient and lightweight, and trains classifiers with a limited ability to generalize from one time series to another. The model is applicable to various unrelated classifiers, in particular nearest neighbours classifiers and decision trees, and to some extend to Bayesian models; though not completely universal, it makes active learning applicable with the same heuristic to different classifiers.

Because ALPAGA is able to silence the false alerts from a monitoring, or an alerting system in general, it can be considered an immune tolerance system; in the context of the Artificial Immune Ecosystem, using ALPAGA to filter the alerts generated by Cloud-Monitor could result in a system verifying the three main immune properties of anomaly detection, memory and tolerance. This stacked architecture, with ALPAGA on top of Cloud-Monitor on top of a monitoring system, is depicted in Figure 6.4.

However, the experiments with ALPAGA reveal a very interesting property: it works well with instance-based classifiers, *e.g.* k-NN. This opens up the possibility of implementing both the immune memory, as discussed in Chapter 4, and a false alarm filter, *i.e.* ALPAGA, within a single classifier. This architecture, illustrated in Figure 10.1, leads to a tighter integration, making the AIE a single model rather than a stack of different ones. This approach is detailed in Section 10.1.

Chapter 9

Heuristics for scalable anomaly detection: SCHEDA

We have shown in Chapter 7 that anomaly detection is paramount in the Cloud-Monitor model. It is also one of the three core properties of immune systems, along with memory and tolerance. However, network monitoring mandates a set of requirements that constrain the possible implementations of anomaly detection, namely:

- *Low computational footprint:* the processing of a new sample should have a constant complexity ($O(1)$) and not depend on the number of samples already collected.
- *Real-time:* data buffering, as required by some algorithms (see Section 3.1), is not an option: the detection time of incidents is critical.
- *No configuration:* scalability prevents the models from being hand-tuned for each series.

In particular, we saw in Section 3.1 that fulfilling these requirements, in particular with the type of time series that are encountered in network monitoring, *i.e.* series exhibiting long-term dependencies, is still a challenge that no model appears to perfectly address. In this chapter, we propose SCHEDA (Scalable Causal Heuristics for Euclidean Distance Approximation), a set of three heuristics accelerating the evaluation of the euclidean anomaly score and suitable as an anomaly detection model for Cloud-Monitor and the Artificial Immune Ecosystem. They are based on three sampling methods: subsampling for PES (Periodic Euclidean Sampling), random sampling for RES (Random Euclidean Sampling) and sparse sampling for SES (Sparse Euclidean Sampling). Together, PES, RES and SES form SCHEDA.

The rest of this chapter is organized as follows. In Section 9.1, we study the shortcomings of naïve euclidean methods, auto-regressive models and SAX-based algorithms for network monitoring and artificial immunity. We show that neither can provide the level of timeliness, accuracy and performance (CPU-wise) that are required in the context of network monitoring. In Section 9.2, we propose three different heuristic approximations of the euclidean anomaly score that reduce the computational burden associated with anomaly detection while maintaining detection accuracy and real-time, low-lag operation. In Section 9.3, these heuristics are evaluated and compared with auto-regressive and euclidean methods on the dataset used to benchmark Cloud-Monitor in Chapter 7 and ALPAGA in Chapter 8.

Table 9.1 – ARMA running time against maximum lag

Max. lag	Running time (s)
1	13.9
2	17.2
5	102.7
8	150

9.1 Open problems

Anomaly detection in time series, as we already saw in Chapter 3, is a complex issue, in particular when specific requirements have to be met. Table 3.3 lists some of the reference methods, and how well they fulfil the requirements of low computational cost, ability to handle long-term dependencies, as found in monitoring time series, real-time operation, and absence of configuration. None appears to achieve all of them at the same time. In the following, we study the limitations of the best contenders: distance-based methods, auto-regressive models and SAX-based heuristics.

9.1.1 Shortcomings of the euclidean anomaly score

The generic formula for causal – *i.e.* without lookahead in the future – anomaly detection based on the euclidean distance (see Section 3.2.1) is as follows, for a time series $T = [T_1, T_2, ..., T_N]$, considering a window of size n:

$$\forall i \in \{n, n+1, ..., N\} A_i = min_{j=n}^{N} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{j-k})^2} \tag{9.1}$$

A is referred to as the euclidean anomaly score. This computation has a quadratic complexity with the number of points and thus cannot scale to the requirements of network monitoring. Its computational cost makes it too heavy to scale to thousands of series [Keogh et al., 2005]. In Section 9.3.5, we evaluate the computational cost of SCHEDA and show that the euclidean anomaly score is 98 times slower than the most expensive of the three heuristics, and over 60,000 times slower than the fastest one.

9.1.2 Limitations of auto-regressive models

Auto-regressive models such as ARMA rely on the time series having a very short autocorrelation – typically less than 5 samples. Table 9.1 summarizes the training time of an ARMA model as a function of the lag, *i.e.* the maximal time dependency. Seasonal models are able to suppress the effect of *one* periodic motif, but not more. This implies that long time dependencies cannot be entirely captured by ARMA (or SARMA).

9.1.3 Qualitative study of SAX-based methods

We have shown in Section 3.3 that the Symbolic Aggregate approXimation (SAX) of time series has given rise to a number of heuristics for various operations, including anomaly detection.

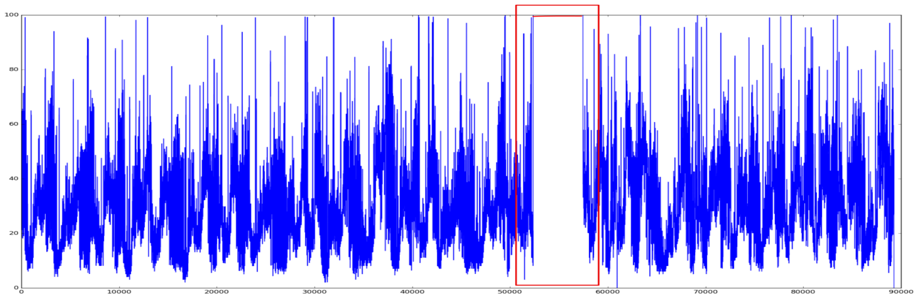


Figure 9.1 – Anomaly in CPU load stuck at 100%

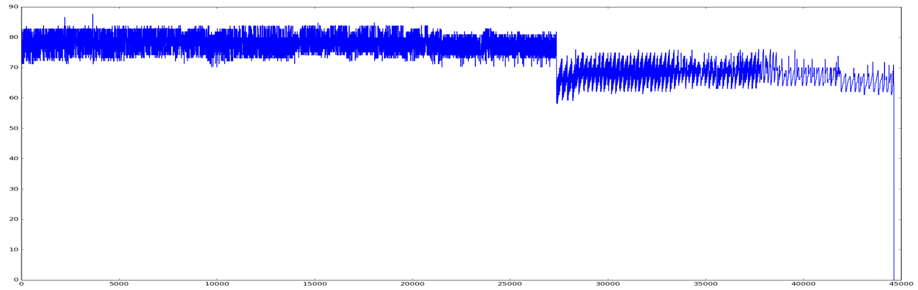


Figure 9.2 – Transition in RAM usage (high and aperiodic to low and periodic)

However, the proposed models still appear to suffer from certain limitations, in particular in terms of required computational power and detection lag (see Table 3.3 in Section 3.5).

We propose a qualitative benchmark of SAX-based algorithms on real monitoring data in order to challenge these apparent limitations. This benchmark was published in [Guigou et al., 2017a]. In the following, we present a simple dataset used for these experiments, the parameters used for the SAX encoding and the various methods and show that, while these methods may seem promising, their shortcomings are such that they cannot be used for real-time anomaly detection in the context of network monitoring.

Methodology

For this benchmark, we use the same dataset as we did in the Cloud-Monitor evaluation in Section 7.5, *i.e.* a set of 32 time series recorded over 2 months and representing the CPU load, memory usage, process count and active TCP sessions of production servers and firewalls. Examples are shown in Figures 9.1 and 9.2. The metrics are acquired with a temporal resolution of one point per minute.

Anomalies such as depicted in figure 9.1 are easy to detect even for conventional, commercial monitoring software. However, more subtle changes (figure 9.2) that can be symptomatic of a critical component crashing are harder to detect and generally overlooked; instead, the crash might be detected later when it causes other symptoms and impact on other parts of the system.

In this test, we evaluate the ability of the HotSAX, SAX/Sequitur and SAX/Chaos Game heuristics to detect anomalies in real monitoring time series. In particular, we evaluate their runtime cost, the detection lag they introduce and their typical failure modes.

Table 9.2 – Runtime comparison of anomaly detection algorithms based on SAX)

Nb of points	Chaos Game	Sequitur	HotSAX
44k	6.66 s	6.62 s	651 s
88k	12.67 s	12.07 s	N/A

Results

Running time We first study the running time of the algorithms. Since we expect to monitor thousands of time series on each monitoring server, a low running time is of paramount importance. The results, summarized in Table 9.2, show that SAX/Sequitur, described in Section 3.3.6, and SAX/Chaos Game, presented in Section 3.3.6, yield about the same running time, between 6 and 7 seconds per month of data, *i.e.* , with one point per minute, a little over 44,600 points. The running time, as predicted, is linear with the size of the series. HotSAX, discussed in Section 3.3.6, on the other hand, displays a much larger, and quadratic, running time. In a series containing one month of data, it took over 10 minutes to produce a result. On a time series of two months, the algorithm ran for more than an hour without producing any result, ruling it out as a candidate for network monitoring.

Detection lag As stated in almost all papers by Keogh *et al.*, the parameters used for SAX encoding (cardinality and word size) have little impact on the behaviour of any algorithm. At most, they can change the sensitivity and the running time, *e.g.* trade off accuracy for faster execution, or move along the precision-recall curve. Window size, however, must generally be adapted to the scale of any feature in the series, *i.e.* the size of the anomaly we try to detect, or the period of the signal.

This is one of the major shortcomings we observed in all SAX-based algorithms: with mostly weekly periodic signals, the best settings would delay any analysis until after at least 7 *days* after a data point is acquired. This is due to the mismatch between the scale of the observed phenomena and the desired detection time. In most use cases, SAX is used to study phenomena of the scale of a second, with a few dozen points per occurrence. We study weekly patterns with thousands of points per occurrence; therefore, the periodic signal is mostly perceived as concept drift from the SAX perspective. Most anomalies can effectively be detected, but not in real time.

Failure modes

HotSAX As HotSAX, by its design, always outputs the mathematically defined worst anomaly, in the sense of the maximum of the euclidean anomaly score, it has inherently no failure mode. It does, however, have three weak points:

- Its computational cost is prohibitive for network monitoring,
- It can only detect *one* anomaly in any given time series,
- In any time series, there is always a *most anomalous* point, *i.e.* one maximum of the euclidean anomaly score. It does not necessarily correspond to an anomaly, and HotSAX provides no decision policy to determine whether the point it detected is an anomaly or not.

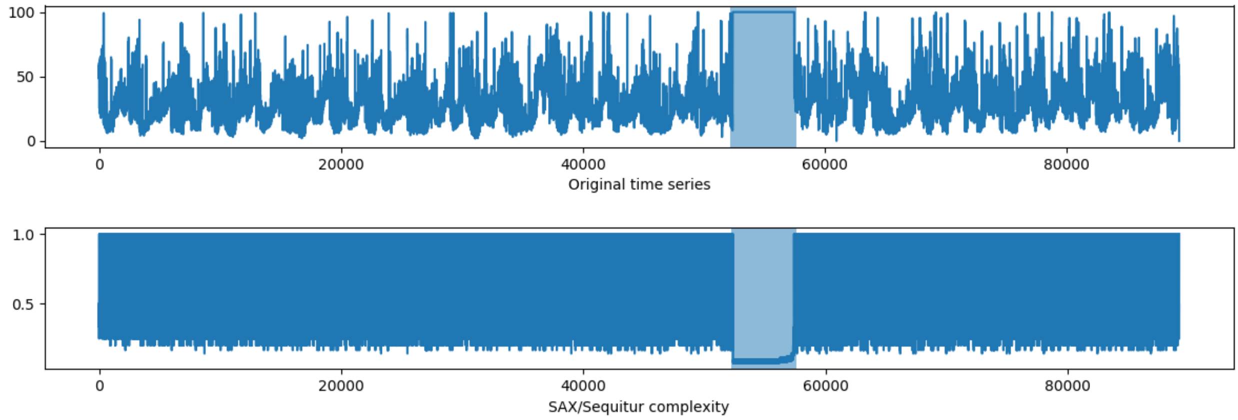


Figure 9.3 – Anomaly undetected by SAX/Sequitur

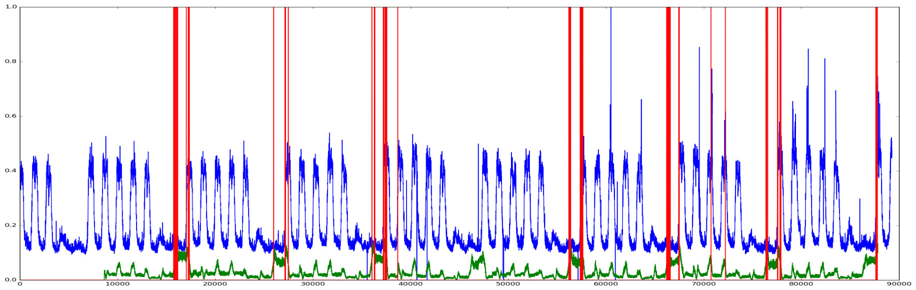


Figure 9.4 – Wrongly detected anomalies due to a weekly cycling pattern (Chaos Game)

SAX/Sequitur While fast and capable of detecting most anomalies, as well as unaffected by the periodic nature of the time series, SAX/Sequitur relies on the low compressibility of anomalies to spot them. Therefore, it completely fails to detect any anomaly that results in a *simplified* pattern, like the one in Figure 9.1. In this particular case, the CPU of a server becomes stuck at 100%, thus changing from a very chaotic pattern of random peaks to a constant value. This is of course an anomaly, but it makes the time series *more* compressible, violating the key assumption of the SAX/Sequitur model. This failure mode is illustrated in Figure 9.3: the anomaly (highlighted in blue) generates a very low complexity, *i.e.* anomaly score, in SAX/Sequitur.

SAX/Chaos Game The SAX/Chaos Game algorithm is the most precise anomaly detector with acceptable runtime performance. We found no failure mode in our dataset. The running time is strictly proportional to the length of the time series, and therefore predictable. We found, however, that the look-ahead required to correctly perform anomaly detection (at least twice the feature window) is unacceptably long, and a short feature window makes the algorithm very sensitive to cyclic patterns. Figure 9.4 shows an example of false positive already discussed in the previous paragraph.

Discussion

HotSAX As described above, HotSAX is extremely slow and its result is of little use in any real-time settings. In our tests, it ran for 10 minutes for one single month of data while the other algorithms only needed half a dozen seconds, as shown in Table 9.2. This violates the scalability requirement of network monitoring.

SAX/Sequitur An interesting feature of the SAX/Sequitur heuristic is that the running time of the algorithm depends more on the complexity (*i.e.* the number of rules generated by Sequitur) than the number of data points. We found it to be, performance-wise and in terms of detection speed, the best algorithm, requiring little to no look-ahead (in contrast to Chaos Game). However, the failure mode we observed is not compatible with a real production environment, in which it can indicate a serious failure or an attack.

SAX/Chaos Game We could not find a correct tuning of the SAX/Chaos Game heuristic, *i.e.* a tuning that would not generate a lot of false positives, as shown in Figure 9.4, without lengthening the window – and hence the time between an anomaly happening and it being detected – to over a day. This sort of lag is incompatible with the real-time requirement of monitoring systems.

9.1.4 Conclusion

The experiments on ARIMA and SAX-based heuristics experimentally confirm the theoretical conclusions drawn from the literature, as presented in 3.1. In particular, they lead to the following conclusions about SAX-based algorithms:

- HotSAX incurs a computational cost incompatible with the scale of network monitoring, where tens of thousands of analyses should be able to run in parallel.
- SAX/Sequitur includes failure modes that are incompatible with network monitoring, in particular the inability to detect the transition to a *simpler* pattern as an anomaly.
- SAX/Chaos Game leads to unacceptably high latencies, with lags in the order of a day, incompatible with real-time requirements.

9.2 Heuristics

We have just confirmed in Section 9.1.4 the limitations found in the literature [Keogh et al., 2005, Wei et al., 2005, Senin et al., 2015], and highlighted in Section 3.5. Namely, current models are incapable of fulfilling at the same time the requirements (see Section 1.3) of:

- Low computational cost,
- Support for long-term dependencies,
- Real-time operation, and
- No configuration.

In this section, we propose three algorithms to approximate the euclidean anomaly score while verifying these requirements. They are grouped under the name of SCHEDA, for Sampled Causal Heuristics for Euclidean Distance Approximation. “Sampled” refers to the fact that only a handful of distances are computed; “Causal” implied that the anomaly score for any given point is computed before the next point is sampled. This is a requirement for real-time processing. We first present the basic assumption on which they operate, *i.e.* that monitoring time series are periodic and have known periodicities, in Section 9.2.1, then the conceptual framework in which they operate, namely euclidean anomaly score subsampling, in Section 9.2.2, and finally the heuristics themselves in Section 9.2.3.

9.2.1 Periodicity of monitoring time series

Most of us sleep at night, work about 9-to-5 Monday to Friday and have lunch sometime around noon or 1pm. We like our summer and Christmas holiday. This has a profund impact on the way power demand, network traffic or server CPU load behave in the course of a day, a week or a year. Many time series measuring metrics linked to human activity show a set of periodic patterns at the scale of the day and the week. Network monitoring is no exception. Figure 9.5 shows the autocorrelation plot of the CPU load of a server and the session count of a firewall measured during two months at a one point per minute sampling rate. The daily and weekly peaks at offsets multiple of 1440 (one day) and 10,080 (one week) are clearly visible.

This periodicity, or seasonality, can be used to determine which distances to calculate and which to ignore to compute an approximation of the euclidean anomaly score: the nearest neighbour, *i.e.* the subsequence with the smallest distance to the *current* subsequence, should be located at an offset corresponding to an autocorrelation peak, *e.g.* 1440 points in the past. Conversely, the nearest neighbour is very unlikely to be found in an autocorrelation valley. This property can be used to limit the search to a small part of the time series.

9.2.2 Euclidean anomaly score subsampling

The problem of anomaly detection can be seen as a particular use of the join profile matrix, noted J [Yeh et al., 2016] (see Chapter 3). This matrix contains, at each coordinate (i, j), the distance between the subsequence starting at index i and the subsequence starting at index j of the considered time series. The length of the subsequence is the only parameter required to compute it.

Figure 9.6 describes the process of computing the anomaly score as the acquisition of the matrix J, where each cell corresponds to the distance between two subsequences, followed by the reduction of each column to a single value ***via*** the min operator:

$$J_{i,j} = \sqrt{\sum_{k=0}^{n-1}(T_{i-k} - T_{j-k})^2} \tag{9.2}$$

$$A_i = min_{j=1}^{N} J_{i,j} \tag{9.3}$$

This matrix J is depicted in Figure 9.6. We can point out three interesting properties it verifies:

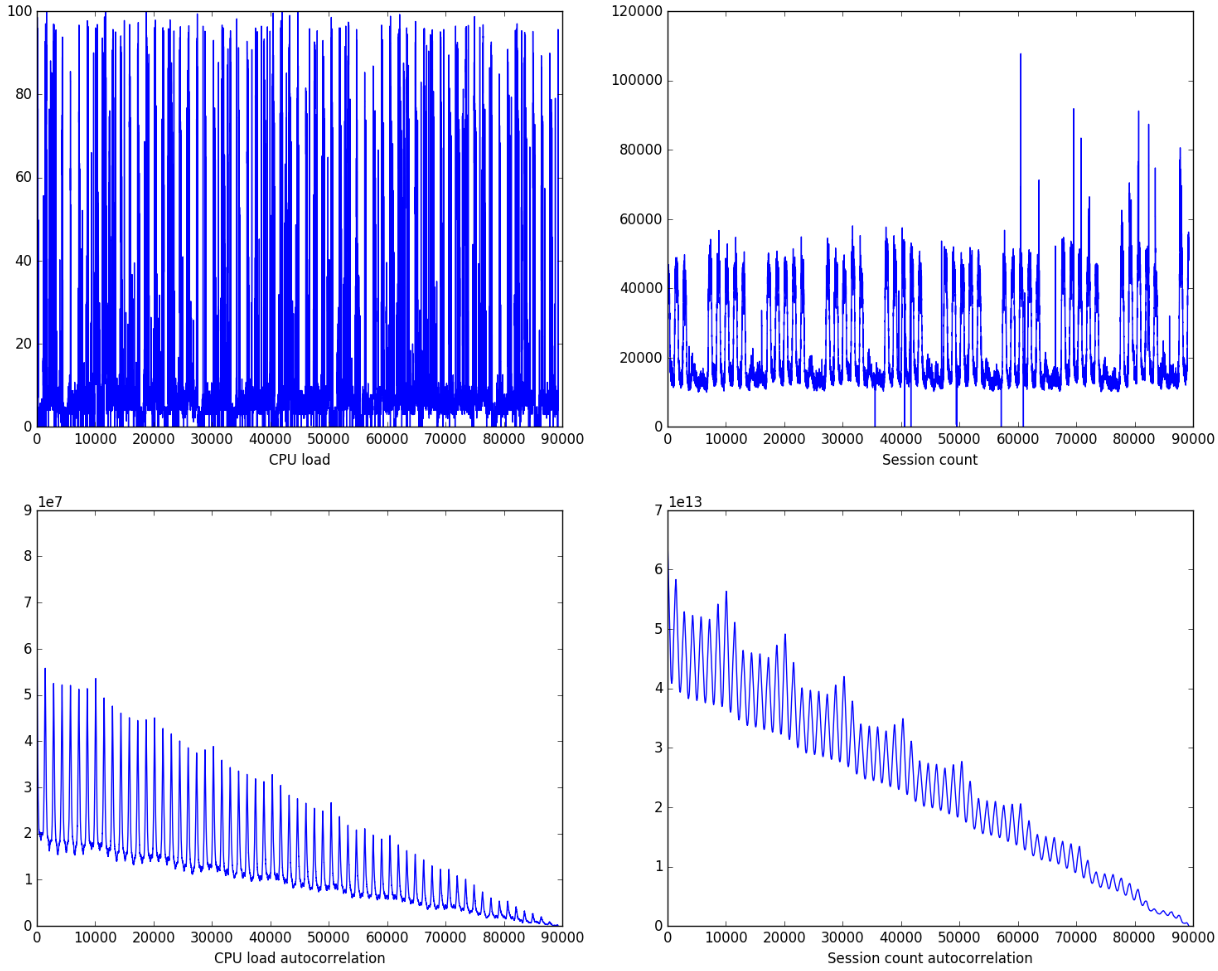


Figure 9.5 – Two IT monitoring time series, with the associated autocorrelation plots

- *Symmetry:* by definition, the distance between subsequence i and subsequence j is the same as between subsequences j and i, thus $J_{i,j} = J_{j,i}$,
- *Undefined diagonal:* self-joins, ***i.e.*** the distance between overlapping subsequences, is meaningless [Keogh et al., 2005], thus the diagonal defined by $|i - j| < m$, where m is the chosen subsequence length, or window size, has undefined values,
- *Local similarity:* successive subsequences only differ by their first and last points, ***i.e.*** $\sum |T_{i:i+n} - T_{i+1:i+n+1}|$ is small. Therefore, for small integers k, l, $|J_{i+k,j+l} - J_{i,k}|$ is expected to be small too. In other words, $J_{i,k}$ is a good approximation of $J_{i+k,j+l}$.

Importantly, the anomaly score is ***not*** the join profile. It is only the ***reduction*** of the join profile along one axis using the min operator. In Figure 9.6, the reduction happens along the vertical axis. Thus the ***exact*** result can be obtained as long as the join profile is not subsampled along the horizontal axis.

Figure 9.7 shows a graphical representation of the join matrix for a time series obtained by polling the memory use of a firewall for a month. We can clearly see the darker diagonal where

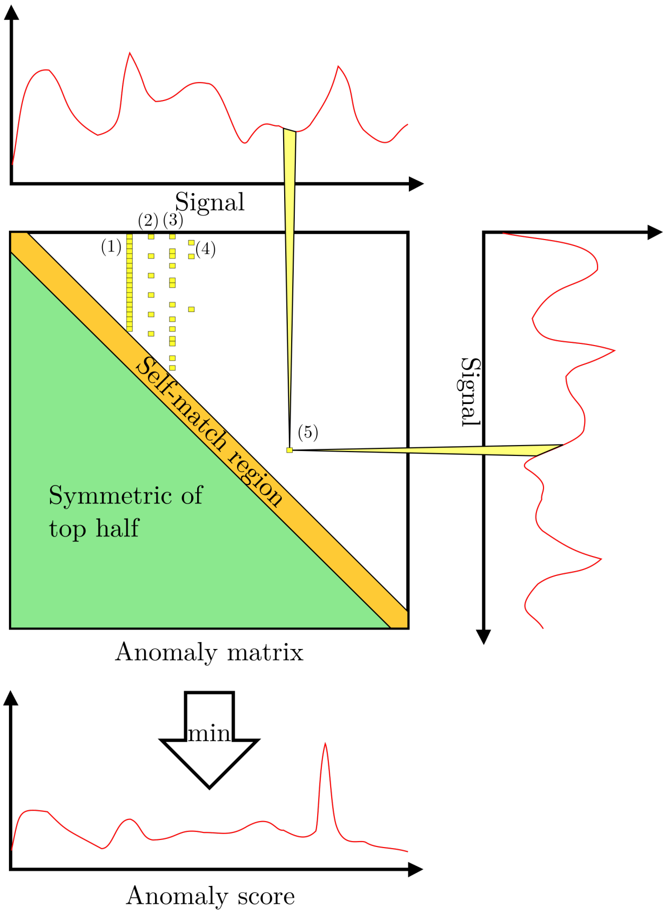


Figure 9.6 – Anomaly detection as a compression/reconstruction problem

values are not computed (and thus set to zero), the symmetric nature of the profile and the mostly slow, low-frequency nature of the variations along each axis.

9.2.3 SCHEDA heuristics

Having established the basic hypotheses on which we operate, we can now describe the three heuristics proposed to achieve real-time, scalable anomaly detection in time series:

Periodic Euclidean Sampling (PES) The first SCHEDA heuristic we present is the Periodic Euclidean Sampling, or PES. The principle of PES is that the distances between a subsequence and two subsequences that are very close to each other (and therefore considerably overlapping) should be very close. As such, it may be enough to compute only one distance in five, or ten, and thus to reduce the runtime cost by an order of magnitude or more. The anomaly score for PES requires one parameter: the subsampling factor s. The anomaly score at index i of a time series T is computed as:

$$A_i = min_{l=1}^{\frac{i}{s}} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{i-l*s-k})^2} \tag{9.4}$$

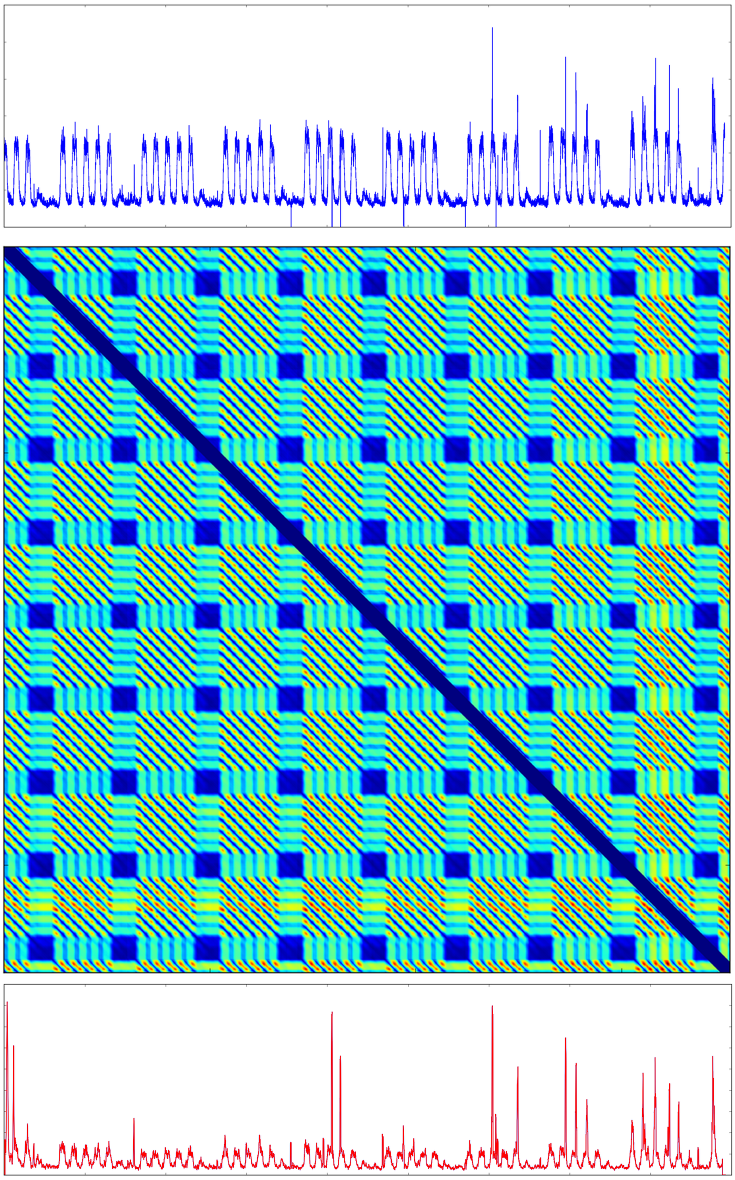


Figure 9.7 – The join profile matrix (middle) of the session count time series on a firewall (top) and corresponding euclidean anomaly score (bottom)

where s is a pruning factor, *i.e.* a (potentially large) integer determining the sparsity of the estimation. It is worth noting that, while faster than the naïve computation of all the distances, this method still has a quadratic complexity in $O(\frac{N}{2s})$.

Random Euclidean Sampling (RES) The Random Euclidean Sampling heuristic computes a set of *random* distances; its only parameter is the number of distances to be computed, r. Because of its random nature, it can be used on any signal, without stationarity assumption. It is defined as:

$$A_i = min_{l=1}^{r} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{i-R-k})^2} \tag{9.5}$$

where R is a random integer generated for each computation. Since the number of distances to be computed is fixed, the complexity of processing a single sample is $O(1)$, therefore the complexity of processing a complete time series of size N is $O(N)$.

Sparse Euclidean Sampling (SES) Finally, we propose Sparse Euclidean Sampling as a very efficient heuristic for time series of stationary periodicity. SES only samples a few distances at very specific lags, where the minimum euclidean distance is expected to be found. Our benchmarks show that 6 lags can be sufficient; see Section 9.3.5. It requires the periodicity to be known ahead of time (which is possible for many cases, *e.g.* for Cloud monitoring) or estimated using some other method. It is defined as:

$$A_i = min_{l=1}^{m} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{i-L_l-k})^2} \tag{9.6}$$

where $L_1, L_2, ..., L_m$ are a set of predefined lags at which the distance is to be sampled. These lags can be known in advance, *e.g.* by noticing that most time series acquired in monitoring scenarios have daily and weekly seasonalities, or computed by spectral analysis, *e.g.* using the autocorrelation of a part of the series. This needs only be done once during an initialization phase, and perhaps refreshed on a regular basis – *e.g.* each month, or whenever the anomaly score starts triggering false alerts – and therefore cannot be directly incorporated in the runtime complexity of the algorithm, which remains constant for a single sample, or linear for a complete series. This is ideal in the case of a signal with a known and stationary periodicity: the minimal distance will always be between subsequences separated by an integer number of periods. Typically, signals related to human activity, such as power demand [Espinoza et al., 2005] or server resource (disk and CPU activity) display daily and weekly patterns; therefore, sound values for $l_1, l_2, ...$ should correspond to a day, a week, and multiples of these periods. For instance, with a sampling rate of one point per minute, a minimal lag vector could be $L = \{1440, 10080\}$: a day and a week. However, with these settings, anomalies generate an echo: the return to the baseline behaviour is detected as an anomaly. Therefore, it is preferable to also add the double of each lag, *e.g.* $L = \{1440, 2880, 10080, 20160\}$, so that the last normal occurrence of the normal pattern is also used in the nearest neighbour search. An example of this echo phenomenon, and how to remove it, is depicted in Figure 9.8 and discussed in Section 9.3.5. Note that SES relies on a very important assumption: the seasonality must remain strictly constant. SES would fail in contexts where motifs exist but can shift in time, *e.g.* in an electrocardiogram, which is highly regular but not exactly constant in time [Wei et al., 2005].

Illustration Let us consider the join profile matrix, as depicted in Figure 9.7. The cells on the lower left half of the image are a mirror of the top right part, while the cells in the top-left to bottom-right diagonal are not computed. The captions `(1)`, `(2)`, `(3)` and `(4)` illustrate columns computed using respectively the euclidean anomaly score (all values computed), Periodic Euclidean Sampling (1 value out of N is computed), Random Euclidean Sampling (a fixed number of random values are computed) and Sparse Euclidean Sampling (a handful of values at specified offsets are computed). In this particular example, Periodic Euclidean Sampling acquires one complete *row* of the matrix out of 5 (vertical downsampling), while Sparse Euclidean Sampling acquires 3 rows.

The final anomaly score is obtained by taking the minimal computed value in each column. In the following, T refers to the considered time series of length N, and n is the window size used for the distance computations, *i.e.* the length of the subsequence considered.

Threshold estimation

All three heuristics output an anomaly score. To perform anomaly detection, a threshold must be applied to that anomaly score. In [Ahmad et al., 2017], the anomaly threshold is established on the probability for a prediction error to come from the predicted model *vs.* the data. The best threshold is a probability of 10^{-5} which, for a normally distributed error, corresponds to about 5 standard deviations. Therefore, we might consider any value of the anomaly score superior to its mean by more than 5 standard deviations to indicate an anomaly.

However, the true standard deviation of the series cannot be known at runtime: only a running estimate can be used. Instead of a global, or true, mean and standard deviation, we can only rely on a *local* mean and deviation. Empirically, we found that a five-sigma rule works well when the true distribution is known, *i.e.* when it can be computed from the complete series, but leads to many false positives when applied on local estimations. Instead, we use an eight-sigma rule: we detect an anomaly if the score $A_i \geq \hat{\mu} + 8\hat{\sigma}$. This process is described in Algorithm 18. The lines beginning with `PES:`, `RES:` or `SES:` correspond to the parts specific to one or the other heuristic. The algorithm should be read by picking one of the three and ignoring the lines specific to the other two. For ease of reading, the heuristics have been colored differently (red for PES, olive for RES and blue for SES).

Algorithm 18 SCHEDA Anomaly Detection

```
procedure AnoDet(S, L)
input:
    $S = [s_1, s_2, ..., s_N]$                         ▷ real-valued time series
    PES: $P : integer$                                 ▷ pruning factor
    RES: $R : integer$                                 ▷ number of random samples
    SES: $L = [l_1, l_2, ..., l_m]$                    ▷ lags for nearest neighbour search
    $w : integer$                                      ▷ window size for distance computation
    $r : integer$                                      ▷ length of the mean/deviation estimation
algorithm:
    $A := [a_1, a_2, ..., a_N] = [0, 0, ..., 0]$       ▷ anomaly score
    PES: $i_0 := P + w$
    RES: $i_0 := R + w$
    SES: $i_0 := max(L) + w$
    for $i := i_0$ to $N$ do
        PES: $a_i := min_{k=1}^{i/P} \sqrt{\sum_{j=0}^{w-1} (s_i - s_{i-kP})^2}$
        RES: $r_1, r_2, ..., r_R = rand(w, i - w)$
        RES: $a_i := min_{k=1}^{R} \sqrt{\sum_{j=0}^{w-1} (s_i - s_{i-r_k})^2}$
        SES: $a_i := min_{k=1}^{m} \sqrt{\sum_{j=0}^{w-1} (s_i - s_{i-l_k})^2}$
        if $i < i_0 + r$ then
            next
        $\hat{\mu} = mean(A_{i-r:i})$
        $\hat{\sigma} = std(A_{i-r:i})$
        if $a_i \geq \hat{\mu} + 8\hat{\sigma}$ then
            signal_anomaly()
```

9.3 Evaluation

In the following, we evaluate the three SCHEDA heuristics, first as pure approximations of the euclidean anomaly score, then as anomaly detection functions. In particular, we evaluate how well they fulfil the requirements of low computational cost and low configuration. By design, they operate in real-time, and the periodicity of monitoring time series are one of their design hypothesis (see Section 9.2.1).

9.3.1 Method

There are multiple ways to evaluate how well these heuristics perform, both as euclidean score approximations and in the general problem of anomaly detection. The aspects of SCHEDA we need to evaluate are the following:

- *Euclidean score approximation:* Ideally, a good approximation of the anomaly score should have a low distance when compared to the actual score, as well as a similar statistical distribution and overall "shape".

- *Anomaly detection:* The objective of SCHEDA is to provide the anomaly detection property of Cloud-Monitor and the AIE. Therefore it should be able to detect anomalies in time series accurately, ***i.e.*** at a level of performance competitive with the euclidean anomaly score.
- *Computational cost:* The reason SCHEDA was developed to overcome the limitations of the euclidean anomaly score due to its quadratic complexity; thus it should be computationally inexpensive.

These properties are evaluated on the dataset of real monitoring time series used to evaluate Cloud-Monitor and ALPAGA. The metrics used are described in the rest of this section.

9.3.2 Anomaly score approximation

Let us consider two time series T_1 and T_2, both of length N. We wish to determine how well T_2 approximates T_1. The mean square error is a well established measure of the amount by which a value in series T_2 deviates from its value in T_1. It is defined as:

$$MSE = \sum_{i=1}^{N} \frac{(T_{1,i} - T_{2,i})^2}{N} \tag{9.7}$$

However, this measure depends on the magnitude of the time series: small relative errors in a time series with large values will generate a higher MSE than large relative errors in a time series with small values. Another drawback of the MSE is that it is inconsistent with respect to linear transformations: if $T_2 = \alpha \cdot T_1$, the MSE remains unpredictable. In contrast, the Mean Absolute Percentage Error (MAPE) is insensitive to magnitude and behaves consistently when linear transformations are applied. It is simply the average percentage of difference between a point of T_1 and the corresponding point in T_2:

$$MAPE = \frac{100}{N} \sum_{i=1}^{N} \left| \frac{T_{1,i} - T_{2,i}}{T_{1,i}} \right| \tag{9.8}$$

Thus we will use the MAPE instead of the MSE as an error measure in all benchmarks.

From a geometric standpoint, we also need to know whether the series have similar shapes, ***i.e.*** whether the peaks and troughs of both series correspond. This can be evaluated by computing the dot product of the series. The dot product measures the degree to which two vectors are ***aligned***. In the context of vectors representing time series, this means that relative highs and lows in both series match – if not in terms of value, at least in their temporal location. It is defined as:

$$T_1 \cdot T_2 = \frac{\sum_{i=1}^{N} T_{1,i} T_{2,i}}{||T_1||.||T_2||} \tag{9.9}$$

This product can be converted to an angle (in degrees):

$$\widehat{T_1, T_2} = \frac{180}{\pi} cos^{-1}(T_1 \cdot T_2) \tag{9.10}$$

Finally, the Kullback-Leibler Divergence (KLD), or cross-entropy, evaluates the quality of a statistical distribution as a predictor for another. In this case, we use it to quantify how well the proposed heuristic approximates the real euclidean anomaly score. The KLD is computed between the statistical distribution histograms of the euclidean score and its approximation. This method has been used for clustering in [Lee et al., 2014]. To compute the KLD, the values taken by T_1 and T_2 are binned in histograms of size m and counted, generating two vectors H_1 and H_2 of size m. Then the KLD is measured as:

$$D_{KL}(T_1||T_2) = \sum_{j=1}^{m} H_{1,m} log_2 \frac{H_{1,m}}{H_{2,m}} \tag{9.11}$$

These metrics would be hard to interpret individually, without context. They are, however, useful to compare the quality of the approximation provided by each of the heuristics. In particular, the dot product indicates how similar the shapes of the series are, the KLD measures the difference in the statistical distribution of the values, and the MAPE shows the actual differences in raw values. Importantly, the dot product and KLD are invariant to linear transformations: if $T_2 = \alpha \cdot T_1$, then $T_1 \cdot T_2 = 1$ and $D_{KL}(T_1||T_2) = 0$. The MAPE is not.

9.3.3 Anomaly detection

Evaluating the anomaly detection performance requires a labelled dataset. Three metrics are relevant: the ROC curve (false positive rate vs false negative rate), false detection rate (FDR) and missed alarm rate (MAR), or false negative rate. The ROC curve is a visual representation of the trade-offs between sensitivity and specificity when the parameters of the classifier vary. In particular, the area under the ROC curve (AUC) measures how good a trade-off a classifier allows; an area of 1 means the classifier never misclassifies a sample; an area of 0.5 corresponds to a random classifier. An area inferior to 0.5 is characteristic of an adversarial classifier that inverts positive and negative detection.

The FDR should not be confused with the FPR (False Positive Rate): the FPR is the ratio of the number of false positives over the number of negative instances (FP/N), while the FDR is the fraction of the detected positives that are false, *i.e.* the probability that an alert is a false positive $(FP/(TP + FP))$. The false positive and false negative rate are aggregated in the AUC measure.

9.3.4 Computational cost

Since SCHEDA is intended as a lightweight replacement of the euclidean anomaly score in Cloud-Monitor and the AIE, the cost of these heuristics in terms of memory and CPU time are of the utmost importance. To get the best estimate, open the time series as CSV files and compute the approximate anomaly scores for various settings of PES, RES and SES, recording the memory footprint, wall clock and CPU time. The results are averaged over all the series. The UNIX `time` utility is used to collect the metrics. In order to assess the cost of a single operation, the times are divided by the number of points in the series, giving an estimated time per point that can then be used to predict the maximal number of points a server can process in a given amount of time.

Table 9.3 – Complexity of various methods for equivalent approximation power

Method	Complexity	Total dists (2 months)	Dot	KLD	MAPE
Euclidean distance	$O(N^2/2)$	3.97 billion	1	0	0
Random noise	N/A	N/A	0.52	6.26	122.1%
PES $(p = 10)$	$O(N^2/20)$	397 million	0.98	0.16	24.1%
PES $(p = 240)$	$O(N^2/480)$	16.6 million	0.93	0.39	74.2%
PES $(p = 1000)$	$O(N^2/2000)$	3.97 million	0.90	0.50	95.1%
RES $(r = 500)$	$O(500N)$	44.6 million	0.97	0.21	36.6%
RES $(r = 100)$	$O(100N)$	8.9 million	0.94	0.37	62.1%
SES $(l = \{120, 1440, 2880, 4320, 10080, 20160\})$	$O(6N)$	535,000	0.93	0.29	79.5%
SES $(l = \{120, 1440, 10080\})$	$O(3N)$	267,000	0.91	0.41	101.8%

9.3.5 Results

We summarize here the results of the evaluation of PES, RES and SES on our network monitoring dataset. We include the euclidean anomaly score [Keogh et al., 2005] and ARIMA [Celenk et al., 2008] as a comparison. The quality of PES, RES and SES as approximations of the euclidean anomaly score is evaluated using the mean square error, dot product and vector angle and Kullback-Leibler divergence, also known as cross-entropy. The anomaly detection capabilities of the heuristics are evaluated using the area under the ROC curve, false discovery rate and missed alarm rate. The CPU, clock time and memory usage are also measured for each model and compared with reference methods.

Euclidean distance approximation

The quality of each approximation is measured by the dot product between the approximation and the exact euclidean score, their Kullback-Leibler divergence and the mean absolute percentage error. Table 9.3 illustrates the various algorithms' performance and complexity. The total number of calls to the distance function is also included, based on a time series running over two months with one sample per minute.

The results in Table 9.3 show that both SES and RES perform well as euclidean score approximations. It is interesting to note that approximations computed by sampling every 240th distance – *i.e.* with a period of four hours –, 100 random distances or 6 well-chosen distances are about equivalent, whichever similarity metric (*i.e.* Kullback-Leibler divergence, dot product or MAPE) is considered. However, with these settings, PES generates over 30 times more computations than SES, and RES almost 17 times as many computations as SES.

These results have the following implications:

- A dot product close to 1 implies that the SCHEDA series "follows" well the euclidean score shapewise: the peaks and valleys match and their height with respect to the noise floor are about the same. Preserving shape is key to anomaly detection, where the absolute value of the anomaly score is of little importance but its contrast, *e.g.* how well the peaks are distinct from the surrounding clutter or noise, is paramount. The best dot product is obtained by PES with a low subsampling factor, followed by RES. SES is the worst performer, with the exception of PES with $p = 1000$. However, SES with 6 lags reaches the same dot product as PES with $p = 240$ and about the same as RES with 100 random lags, but at a much lower cost (535,000 distance computations for SES versus 8.9 million for RES and 16.6 million for PES.

- A low Kullback-Leibler divergence indicates that the statistical distributions of the euclidean score and its approximations are close. This is another important aspect of anomaly detection: since the detection threshold is directly derived from the statistical distribution of the anomaly score, it should remain as close as possible to the euclidean score. PES with $p = 10$ still wins, but the KLD of PES degrades rapidly when the subsampling factor p increases. RES, even at 100 lags, is closer to the statistical distribution of the euclidean score than PES with $p = 240$. SES performs on par with RES, with 6 lags giving a KLD of 0.29, only bettered by PES at $p = 10$ and RES at $r = 500$.
- A low MAPE means that a point in the approximation is close to its corresponding point in the reference series. This is a very generic measure to quantify the quality of prediction or compression algorithms. While hard to interpret by itself, its variations are more important than those of other metrics and illustrate well the convergence of RES and PES towards the true euclidean score as more distances are computed, while SES remains generally less accurate than the other two heuristics, though using much fewer computations. We observe the same pattern as with the KLD when comparing PES and RES: RES beats PES, except with $p = 10$. However, SES performs significantly worse than RES and PES: even with 6 lags, it is only better than PES with $p = 1000$, *i.e.* sampling every 1000th distance.

We have previously notes that the dot product and KLD are invariant to linear transforms, but not the MAPE. We also note that, under the dot product or the KLD, SES performs at the same level or slightly worse than RES and PES; however, under the MAPE, SES is clearly worse. This indicates that SES probably computes a linear transform of the euclidean anomaly score, *i.e.* $SES(T)\ \alpha EAS(T)$. This makes it the worst approximator, but does not necessarily imply that it performs worse for anomaly detection.

Anomaly detection

Demonstration on artificial data RES and SES are first qualitatively tested on artificial data, so as to demonstrate the upsides and downsides of each flavour. These tests are shown in Figures 9.8 and 9.10. Figure 9.8 pictures a toy example of a structural, or collective anomaly: one occurrence of a pattern is shifted in time with respect to the others. The pattern *in itself* is normal, as are the individual values, but it happens too early. In this case, the euclidean score is unable to detect anything, while SES readily picks up the anomaly. Note that such shifts are likely to happen twice a year in countries implementing daylight savings time: a pattern caused by human activity and expected to repeat every 24 hours will be shifted an hour because Monday, 9 a.m. will occur 47 (or 49) hours after Saturday, 9 a.m...

This figure shows a number of interesting phenomena. First, we see that the euclidean anomaly score, for the whole duration of the anomaly, is zero. This is an effect of window size: the *actual* anomalous parts of the series are the two flat areas on both sides of the peak, the left one being too short and the right one too long. However, if the analysis window is shorter than

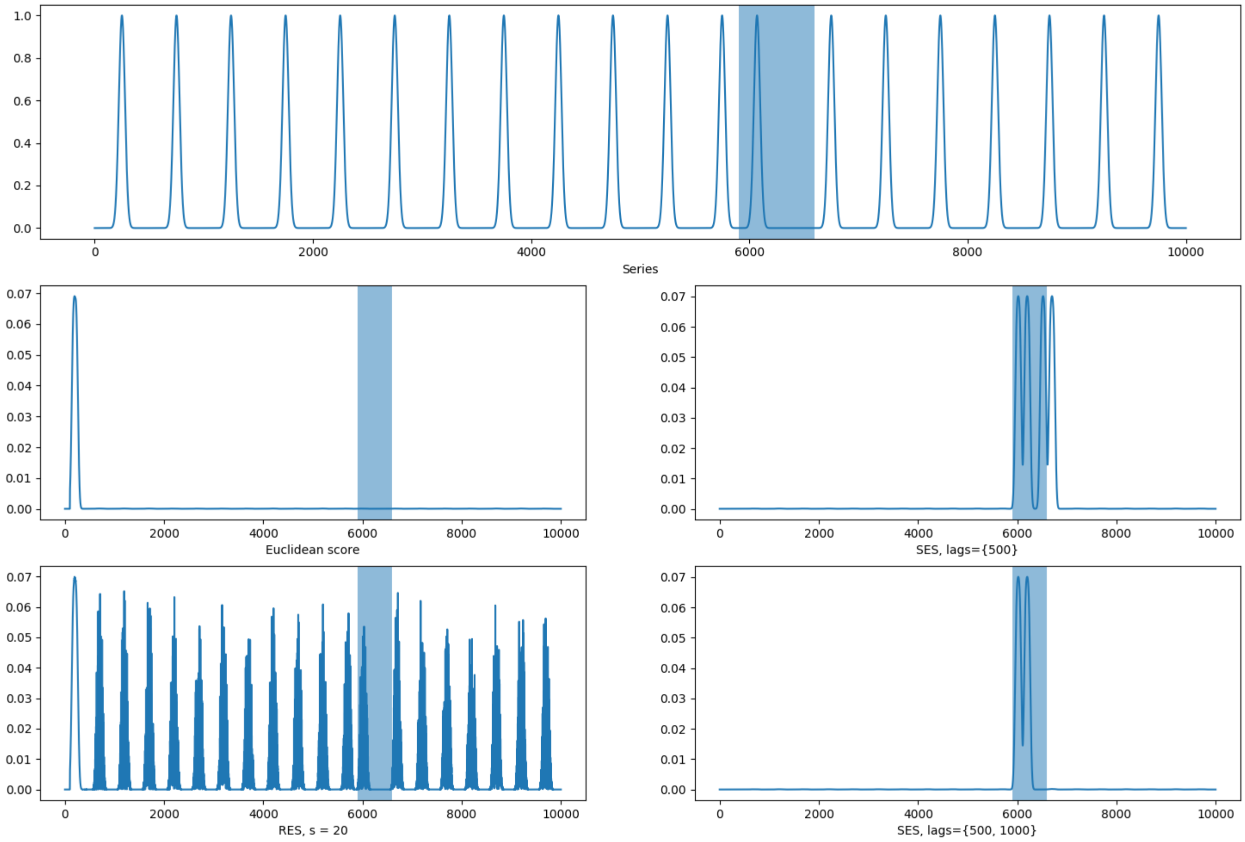


Figure 9.8 – Dirac train-like series with structural anomaly

the normal duration of a flat pattern, the anomaly cannot be captured. The influence of window size is demonstrated in Figure 9.9.

Secondly, we can see how the RES score is chaotic, at least with a small sample size. Here the anomaly does not appear to generate more effect than the normal patterns.

Thirdly, the two SES plots pick up the anomaly very clearly, but the version with a single lag equal to one period actually detects it *twice:* the return to a normal pattern is detected as an anomaly as well. Adding a second lag equal to two periods gets rid of the echo, hence the recommendation in Paragraph 9.2.3 to always include at least *two* lags per period of the signal when using SES.

Figure 9.9 shows the same time series with the same anomaly and the anomaly scores obtained under the eucldiean distance for different window sizes. The anomaly is completely missed until the window size becomes as large as the period of the signal. Note that this phenomenon is not experienced by SES, as we have just shown. This is a strength of SES over the euclidean score for anomaly detection.

Figure 9.10 depicts a realistic – though artificial – time series containing four distinct anomalies that can represent real-life scenarios, namely:

- A high peak of activity during an active period (point anomaly) that could indicate a server overload or an attack,

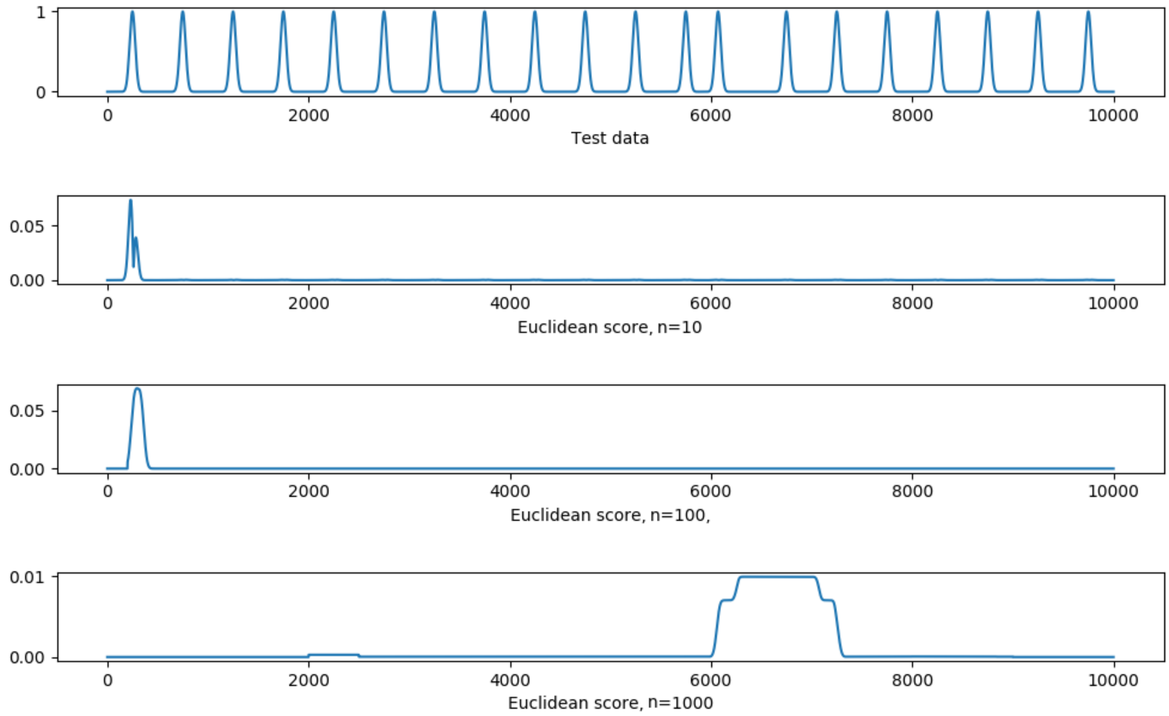


Figure 9.9 – Euclidean anomaly score computed using different window sizes

- A pattern of low activity in a normally active period (collective anomaly) that could indicate a network failure or simply depict the unexpected drop in activity during a holiday,
- A very short peak of high activity in a normally active period (point anomaly) that could indicate an overload or an attack,
- A short peak of activity in a normally inactive period (contextual anomaly) that could indicate a server failure or an attack.

With this second time series, we can see that the euclidean score shows clear peaks at the location of the anomalies. RES closely resembles the euclidean score, though it offers less contrast: anomalies tend to be closer to the noise floor.

SES, on the other hand, has a higher contrast but suffers again from the echo phenomenon: out of four anomalies, the first two, which happen in the middle of a week, are correctly detected. The last two, on the other hand, taking place just after and during a week-end, where two of the reference lags (one day before and two days before) are much different from what is observed, are much more blurry and bleed further in time. The third anomaly, on a Monday, makes the Tuesday anomalous. The last Monday, which shows a high anomaly score, is a very interesting case: of its 4 reference lags, 3 are useless: the previous 2 days are a week-end, which means a totally different statistical distribution, and the previous Monday contains an anomaly. Therefore, the anomaly score is estimated from a single reliable point, and is much higher than it should be. This, in turn, generates a false positive. It illustrates the most prominent failure mode of SES and confirms the importance of using multiple lag values for each periodic pattern of the time series considered.

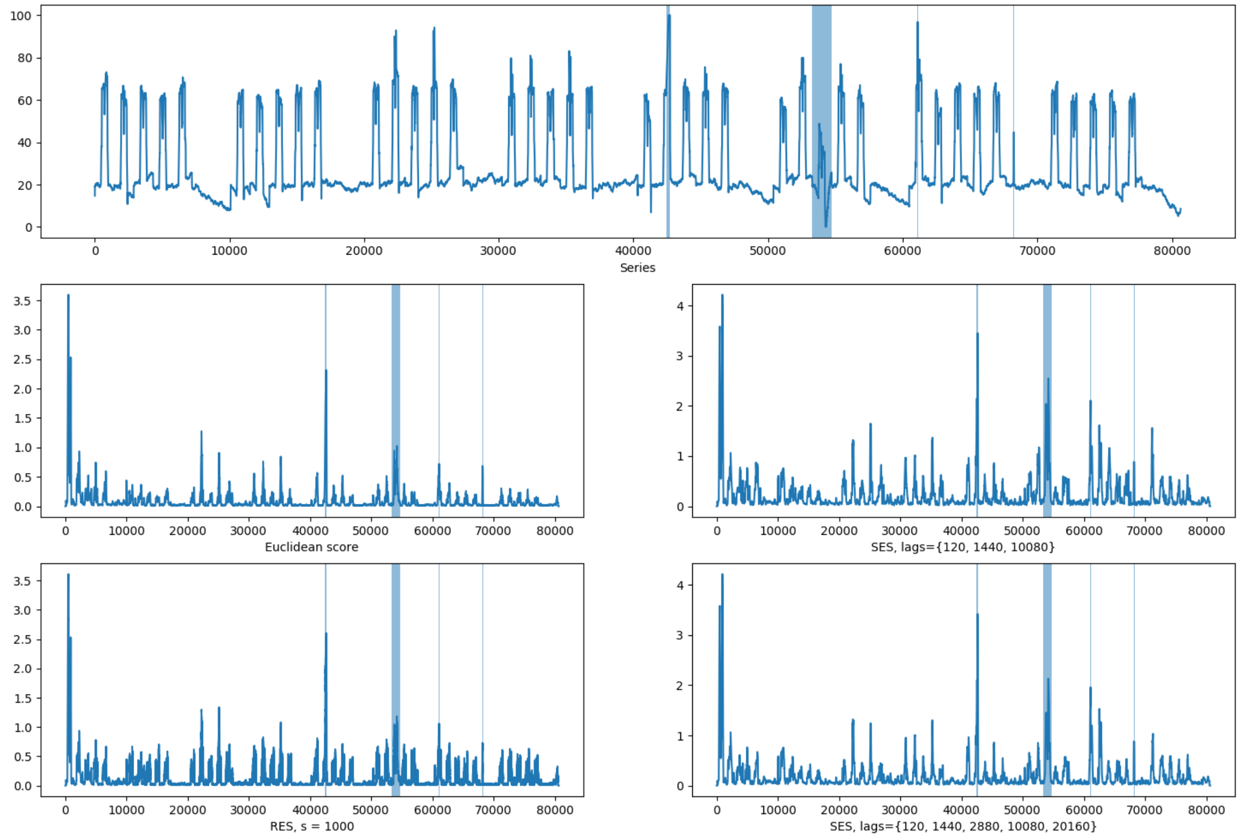


Figure 9.10 – Human activity-like series with 4 typical anomalies. SES, RES, PES and the euclidean score are demonstrated.

Real data Table 9.4 summarizes the performance of each approximation for anomaly detection on the real dataset. Figure 9.11 is given as an example of the typical shape of real time series and anomaly scores. Marks on the lowest time series correspond to detections, with green marks being real positives, red marks false positives and yellow marks detections in areas that have not been labelled as definitely normal or abnormal.

The euclidean score and ARIMA are included in the results as a reference. The metrics used are the area under the ROC curve (AUC), the False Discovery Rate (FDR), *i.e.* the proportion of false alarms, and the Missed Alarm Rate (MAR), *i.e.* the proportions of anomalies that did not result in an alarm. The AUC is computed using the Pareto front, *i.e.* using the optimal parameters for any given false positive rate. The actual ROC curves are depicted in Figure 9.12. The false discovery rate and missed alarm rate are computed using the eight sigma rule discussed in Section 9.2.3.

The results in Table 9.4 show that as far as anomaly detection is concerned, all three heuristics are considerably superior to the euclidean score or ARIMA, with PES showing the best results, followed by RES and SES.

The ROC curves of PES, RES, SES and the euclidean score, as depicted in Figure 9.12, are virtually identical: they first show a sharp increase up to a TPR of about 0.5 for an FPR below 0.1, which is a very beneficial trade-off. However, the trend quickly stops and the ROC curves slowly closes to the diagonal, indicating in increasingly unfavorable trade-off. This indicates a

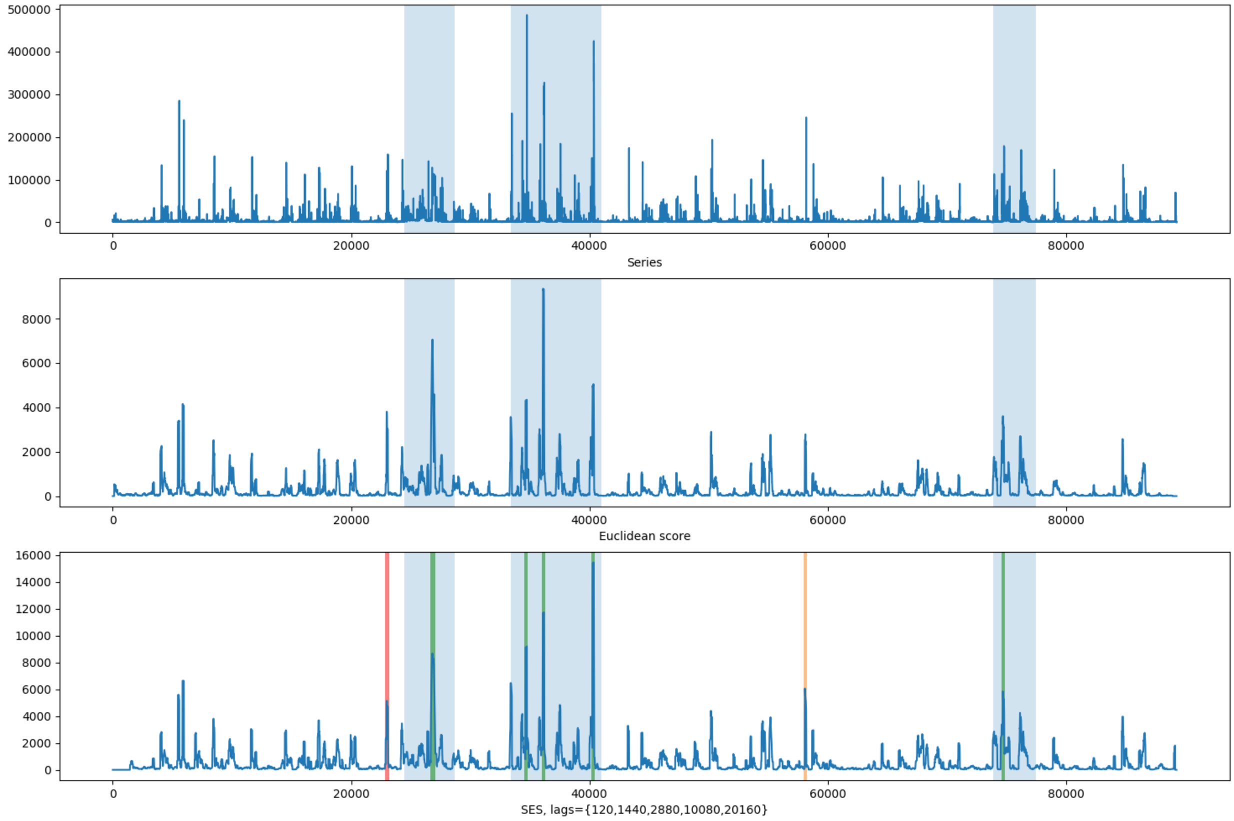


Figure 9.11 – Example real time series, euclidean anomaly score and SES score

Table 9.4 – Comparison of various euclidean approximations for anomaly detection on real data

Method	Complexity	AUC	FDR	MAR
Euclidean score	$O(N^2/2)$	0.69	0.14	0.41
ARIMA	$O(N)$	0.56	0.20	0.46
PES $(p = 60)$	$O(N^2/120)$	0.74	0.06	0.41
RES $(r = 100)$	$O(100N)$	0.71	0.07	0.38
SES $(l = 1440, 2880)$	$O(2N)$	0.71	0.10	0.46

Table 9.5 – Runtime cost of the SCHEDA heuristics

Method	Clock time	CPU time	RAM	CPU time/pt
Eucl. dist.	5 h 37 min.	67 hours	13.5 MB	166.4 ms
PES ($s = 100$)	212 s	42:15	13.2 MB	1.7 ms
RES ($s = 500$)	95 s	18:48	13.3 MB	$777 \mu s$
RES ($s = 100$)	25 s	4:41	13.0 MB	$194 \mu s$
SES (6 lags)	4 s	4 s	12.8 MB	$2.7 \mu s$

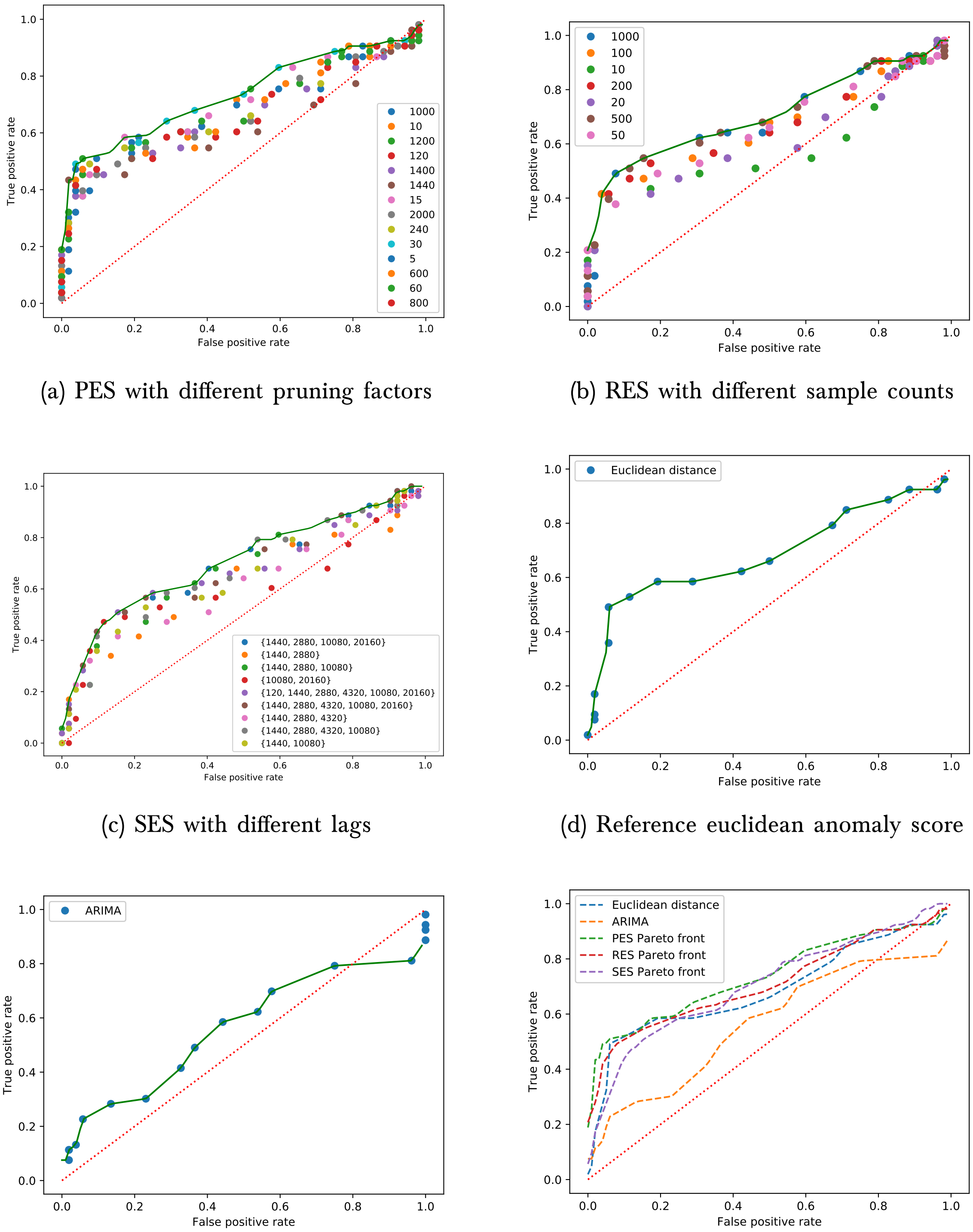
clear optimal operating point for these four models: aiming for the 0.5 TPR consistently gives the point furthest away from the diagonal, which corresponds to a random classifier. ARIMA, on the other hand, shows no such optimal point: its ROC curve is consistently slightly better than random, and becomes adversarial at about 0.8 TPR and FPR. This indicates that ARIMA is not suited for anomaly detection in the types of time series obtained by network monitoring. The Pareto fronts of all the models show the best operating point is achieved by PES, followed very closely by the euclidean score, then closely by RES and later by SES. These graphical results differ from those shown in Table 9.4 because the AUC considers the overall performance of the classifier across all possible operating point, *i.e.* alerting thresholds, while we interpreted the *best* operating point only in the ROC plot.

Computational footprint

The CPU and RAM costs are measured on a specially crafted artificial series running for 3 years, *i.e.* about 1.5 million points. All algorithms are implemented in C and read the full series in memory before carrying out the computations. The results are written in a file and not kept in memory. The CPU used is a 12 core Intel Xeon clocked at 2.2 GHz. The running times reported in Table 9.5 are both expressed in wall-clock time and CPU time, the latter being the most relevant here as it quantifies the actual cost of the algorithm, while the wall-clock time only reflects the degree of parallelism of one particular implementation on one particular CPU. Typically, the wall-clock time can be reduced by adding more cores to the machine, while CPU time can only be reduced by algorithmic of implementation improvements.

The results in Table 9.5 quite clearly demonstrate the primary quality of SES: its low computational cost. While all methods use about the same amount of memory (13 MB), their CPU cost varies wildly, from 2.7 microseconds per data point for SES to almost a second for the euclidean distance. Keep in mind, however, that both the euclidean score and PES run in quadratic time, therefore the CPU time per point increases with the length of the series. Obviously, if any real life application were to use one of these metrics, the search range would have to be limited somewhat (*e.g.* only compute the distance with respect to the data of the previous N months).

To assess the relative costs of RES and SES, let us imagine a scenario where a server would be running each one and receiving one point per minute for each series it monitors. Then the maximal number of series that could be monitored is given by $n = k * 60/t$, t being the processing time per point and k the number of cores on the server. This disregards the time required to actually *acquire* this point, which, if not optimized, may well be by far the most expensive operation. The results of this simulation are given in Table 9.6 for a 4-core server running at 2.2 GHz with 16 GB of RAM.



(a) PES with different pruning factors

(b) RES with different sample counts

(c) SES with different lags

(d) Reference euclidean anomaly score

(e) ARIMA (1, 1, 1)

(f) Pareto front for SES, RES, PES, the euclidean score and ARIMA

Figure 9.12 – ROC curves for PES, RES, SES, Euclidean score and ARIMA. For PES, RES and SES, the green line represents the Pareto front, *i.e.* the ROC curve using the optimal settings for each detection threshold

Method	Max. monitor count (CPU)	Max. monitor count (RAM)
Euclidean distance	1446	1185
PES ($s = 100$)	141,176	1212
RES ($s = 500$)	308,880	1203
RES ($s = 100$)	1,237,113	1230
SES (6 lags)	88,888,888	1250

Table 9.6 – Maximal number of SCHEDA monitors per server (4 cores, 2.2 GHz, 16 GB of RAM)

Number of points	RAM usage (kB)
1	1880
44639	2130
89279	2320
403200	4570
806400	7644
1451520	12800

Table 9.7 – RAM usage of SES *vs.* time series length

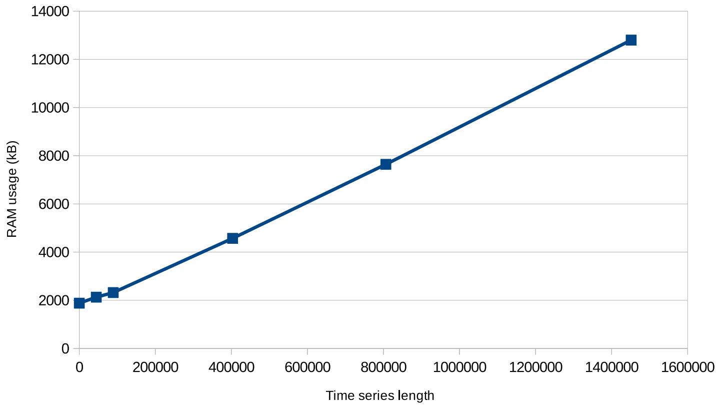


Figure 9.13 – RAM usage of SES *vs.* time series length

However, the RAM would be more limiting than the CPU. Considering a server with 16 GB of memory, and disregarding the memory used by the operating system and various services running on the machine, only about 1200 monitors could run in parallel, regardless of the method.

While this figure is somewhat disappointing, it fails to account for the fact that in a real scenario, the executable file and shared libraries would only be loaded *once*, and thus need to be factored out. To study the impact of the initial cost of loading the code and keeping the time series data in memory, we ran multiple SES instances and measured the total memory usage as a function of the size of the time series. The results are shown in Table 9.7 and Figure 9.13.

The results show a clear linear trend, with an essentially empty time series requiring 1880 kB and a 1000 day long time series using 12.5 MB[1]. A linear regression on these measurements

[1] 1 MB = 1024 kB

yields an initial cost of 1.66 MB, with a linear cost of 7.7 bytes per point of the series. While PES and RES are not bounded in time, *i.e.* can sample distances at any point in the time series, SES uses a set of predefined lags, thus only needs to keep as many points in memory as its maximal lag. If we take this lag to be two weeks, or 20,160 points, then applying the linear model gives a memory usage of an initial 1.66 MB, then 152 kB per series. With a 16 GB server, this means that 100,360 monitors could fit in memory.

9.3.6 Discussion

Let us now review the results of our experiments in the light of the requirements of monitoring systems, as defined in Section 1.3:

- *Low computational footprint:* Interestingly, the real limitation to the number of time series that can be processed is the available memory, and not the available CPU power. This can be mitigated by either using smaller float representations, compressing the data in memory or retrieving the relevant points from a time series database on disk. However, even without such optimizations, the SCHEDA heuristics, and especially SES, are scalable enough: it can process over 100,000 series on a single server, way over the required tens of thousands. The computational cost of SES is over 60,000 times lower than that of the euclidean anomaly score, yet its performance is generally superior when it comes to anomaly detection.
- *Real-time:* All three heuristics are real-time by design, as they process data points without buffering.
- *Configuration-free:* Under the 8-sigma rule, *i.e.* with a fixed threshold and no configuration at all, all three algorithms perform correctly.

The ROC curves themselves are not typical of what is usually considered “good”: an area barely above 0.7 is rarely considered a good outcome. Outlier detection in [Lazarevic and Kumar, 2005] typically exhibits AUC over 0.85. However, we have seen, by examining the ROC plots, that they are divided in two regions: one where the curves moves away from the diagonal, where the classifiers perform well, and one where it comes back towards the diagonal. The AUC averages the very efficient region of the heuristics with the diminishing returns region, where they should not be operated.

A better way to understand these results is to look at the MAR and FDR. In Table 9.4, we can see that, for instance, RES, using the eight-sigma rule, has a false alarm rate of 7%, thus one in 14 detections is a false alarm, while the missed alarm rate is 38%, meaning that nearly two thirds of the anomalies are detected. By comparison, SES, with the exact same AUC, has a 10% false alarm rate and a 46% missed alarm rate, both worse than RES.

Now we can put these results in the context of monitoring: the state of the industry is such that current monitoring systems are unable to detect anything but point anomalies; and, even then, generate more than 10% of false alarms[2]. Therefore, SCHEDA offers a significant improvement over traditional monitoring solutions, while keeping the operational cost (the false alarms sent to the support teams) acceptably low.

[2] Personal experience of the author

In the context of Cloud-Monitor and the AIE, we have demonstrated that any of the SCHEDA heuristics can replace the euclidean score for anomaly detection. In particular, SES is significantly less demanding in computational power and memory than RES and PES. Overall, all three heuristics verify the core properties for which they were designed: low computational cost, support for long-term dependencies, which is even the basic principle on which SES operates, real-time analysis and absence of configuration – as shown Table 9.4 where the thresholds were automatically computed.

9.4 Conclusion

We have shown that the proposed heuristics are competitive with the euclidean anomaly score in terms of anomaly detection and, in some cases, can even perform better. The heuristics only require a small computational power and can operate in real time, at scale.

The purpose of SCHEDA is to provide a performant anomaly detection module for Cloud-Monitor. The initial implementation used the euclidean anomaly score as its metric for anomaly detection (see Section 7.4), and thus could only use the full mode on a few series at a time. The introduction of SCHEDA, with its extremely low computational cost and better anomaly detection abilities, makes it possible to enable Cloud-Monitor's full mode for each and every time series under monitoring. This, in turn, makes the AIE scalable to thousands of time series per node, as we demonstrate in the next chapter.

Chapter 10

Evaluation of the AIE

In Chapter 6, we have presented the Artificial Immune Ecosystem (AIE) as an integration of Cloud-Monitor, ALPAGA and SCHEDA, where Cloud-Monitor provides a detection pipeline, including memory, SCHEDA accelerates anomaly detection to make it scalable and ALPAGA interacts with the expert and filters the false alerts. This model of the AIE was illustrated in Figure 6.4.

The AIE still uses a thresholding model as a form of innate immunity, performing a broad-spectrum detection. The anomaly detection studied in Cloud-Monitor (Chapter 7) and made scalable by the work on SCHEDA (Chapter 9) models the acquired immunity. Finally, the immune memories of Cloud-Monitor and ALPAGA (Chapter 8) provide the secondary immune reaction and tolerance.

In the following, we first propose an alternative, more efficient model where the three main components of the AIE – Cloud-Monitor, ALPAGA and SCHEDA – are more tightly integrated. This new model does not change the visible behaviour of the AIE, *i.e.* the alerts it raises, only its internal structure. We present the this model, along with the inefficiencies of the first one, in Section 10.1. In Section 10.2, we benchmark the AIE on our standard test dataset and compare its performance with Cloud-Monitor as an immune monitoring system. In Section 10.3, we discuss how well the AIE provides the full spectrum of immune properties: anomaly detection, memory and tolerance. Section 10.4 concludes this chapter.

10.1 Final model

Let us consider again the basic priciple of the AIE, as shown in Figure 6.4. This model is based around Cloud-Monitor and, as such, has some of the same properties, in particular anomaly detection and memory. The use of SCHEDA for anomaly detection means it is also scalable. The main difference between the complete AIE and Cloud-Monitor is the presence of ALPAGA to interact with the expert and filter the false alerts generated by Cloud-Monitor, which were shown to be problematic (see Section 7.5.3). This full model should verify the three main immune properties of anomaly detection, memory and tolerance, and also fulfil the requirements of monitoring systems (see Section 1.3), in particular a low false positive rate.

The AIE, as we just described it, shows a major point of inefficiency: it uses *two* databases – one for Cloud-Monitor and one for ALPAGA. The order of operations also becomes illogical because of this duplicated database lookup. If the thresholding step generates an alert, then

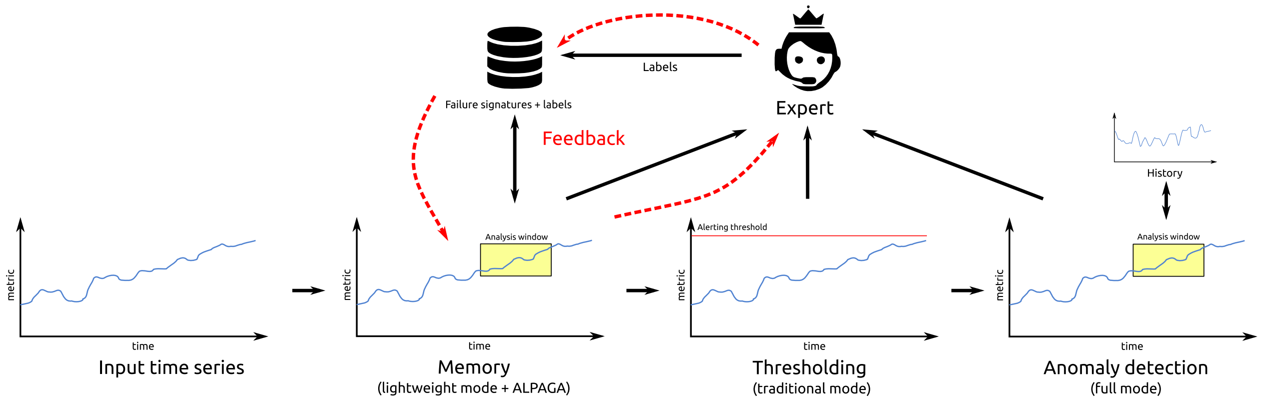


Figure 10.1 – Integrated AIE data flow with single database/classifier

ALPAGA follows with a lookup to confirm it. If it does not, then the lightweight mode generates a lookup; worse, if this lookup results in a match and an alert, ALPAGA looks up its own database to confirm. Two things become obvious here:

- Every sample generates a database lookup, and
- Separate databases lead to duplication of work.

We concluded Chapter 8 by showing that nearest neighbour classifiers are perfectly suitable for false alert filtering. Thus the memories of both the lightweight mode of Cloud-Monitor and the classifier of ALPAGA are essentially the same: time series subsequences placed in an accelerating structure. Moreover, since ALPAGA records all the alerts from its parent system, all the samples contained in the memory of Cloud-Monitor should also be present in the ALPAGA database. In other words, these databases are nearly identical, with only one difference: ALPAGA's database contains *labels* along with the samples. This difficulty can be solved by the introduction of three-state logic [Holland, 1983]: the labels can be either 0 (false alert), 1 (true alert) or # (no label), which allows labelled and unlabelled samples to be merged in the same database.

Thus we propose a new data flow where the databases are merged and the lookup is moved at the beginning of the detection pipeline, so as to provide an early exit for the process and remove lookup duplication. This data flow is depicted in Figure 10.1. It shows three main differences:

- The memory matching step is moved to the first position in the analysis pipeline, providing an early exit point if ALPAGA matches the analysis window with the signature of a false alarm; thus the alarm itself, instead of being filtered, is never generated.
- There is no longer a clear division between Cloud-Monitor and ALPAGA, as they share the same database. The classifier within ALPAGA is able to stop the detection process if it detects the potential for a false alarm as well as send the alarm if the window matches a known failure mode.
- The feedback loop propagates to the very beginning of the analysis: instead of a simple expert/filter interaction, the integrated model moves the expert's locus of control to the head of the pipeline.

Note that this new model is only an optimization of the AIE and does not change its response in any way. Hence the AIE remains the result of the integration of Cloud-Monitor, SCHEDA and ALPAGA.

10.2 Evaluation

Using this new model, we can now study the performance of the AIE, both in terms of computational footprint and incident detection, including false positive rate.

10.2.1 Cost of nearest neighbour lookups

The anomaly detection step of the AIE has been optimized by the SCHEDA heuristics. Since the immune memory is implemented by a nearest neighbour classifier, and this classifier is run for every data point (first step depicted in Figure 10.1), it is also necessary to optimize the speed of the nearest neighbour lookup that provides memory and tolerance. To this end, accelerating structures have been developed, notably the k-d tree [Bentley, 1975] and the ball tree [Omohundro, 1989]. A general review of instance-based classifiers is proposed in Section 4.2. These structures allow neighbours to be retrieved without computing the distances between each possible pair of samples.

Table 10.1 shows the 1-NN lookup time for both structures and using only brute-force search. The test vectors are typical failure modes, as recorded by the AIE: time series subsequences of 120 points corresponding to alerts raised by the system. The measured times are for 100,000 lookups against a database of 10,000 instances.

This experiment clearly shows that for high-dimensional data such as time series, a ball tree is the best accelerating structure. We can now study the scalability of this approach. Assuming that:

- the failure signatures are not shared between time series,
- ten new signatures are generated per month and per series (mostly by false positives),
- signatures are kept for three years,

then a server monitoring 100,000 time series would have to check 100,000 points every minute against a database of 360 signatures. Using a ball tree, simulations show that these 100,000 lookups would require 430 milliseconds, or just 0.72% of a minute. Note that monitoring one million time series requires one million lookups, but the size of the database does not change, since the signatures are recorded *per series.* Thus the three main properties of immune systems, arnomaly detection, memory and tolerance, can all be implemented in such way as to

Table 10.1 – Lookup time for 1-NN search

Index structure	Lookup time (s)
None	16.38
k-d tree	6.79
Ball tree	3.59

Table 10.2 – Performance figures of the AIE

CPU time (total)	27.6s
CPU time (per sample)	192μs
Memory used (total)	147MB
Memory used (Python + libraries)	134MB
Memory used (AIE)	13MB
Number of signatures	8

validate the low computational cost requirement of monitoring systems: they can effectively be added to a monitoring system.

10.2.2 Total runtime cost

Keeping the assumption that the processing of different time series is completely independant, we can measure the overall performance of the AIE by running it on a single time series. To this end, we use the "dummy" time series used to demonstrate Cloud-Monitor, as shown in Figure 7.2. This series is 144,000 samples long, equivalent to 100 days of data sampled once per minute. We evaluate a Python/NumPy implementation of the AIE with a single ball-tree data structure for the immune memory. Alerts are displayed to the user as graphics plots accompanied by a dialog asking to classify it as a true or false alarm. We obtain the following performance figures displayed in Table 10.2.

If we contrast these results with the performance figures of SCHEDA in Section 9.3, we find the exact same memory usage, though the Python implementation and additional libraries have a higher initial cost. This indicates that the added footprint of the ball tree used to store signatures remains negligible. Using the same computation as for SCHEDA, the CPU runtime cost remains sufficiently low to fit 1,250,000 monitors on a 4-core server clocked at 3.20 MHz. The relative benefits of specific instruction sets such as SSE ans AVX has not been studied.

Thus the memory and CPU usage of the full AIE fulfil the requirement for monitoring systems; as is the case with SCHEDA, memory usage can be reduced by only keeping a buffer as long as the maximum SES lag. In this case, moving to a C implementation would also allow for a fine-grained memory management.

10.2.3 Detection performance

Let us first consider the Cloud-Monitor test time series. Since Cloud-Monitor does not generate false positives on this particular example, we need to change our labels to demonstrate immune tolerance. Thus we consider that the dips (short drops) are part of the normal activity of the monitored system.

Active learning allows the user to decide, for each detected event, to flag it as a condition requiring alerting or not. In our proof-of-concept implementation, this is simply a matter of clicking "Yes" or "No" in a dialog box.

As expected, the first dip is detected by SCHEDA and generates a user request in the form of a plot showing the alerting threshold and the past 2 hours of data, and the aforementioned dialog; the following three dips are automatically classified as false alerts, generating no false

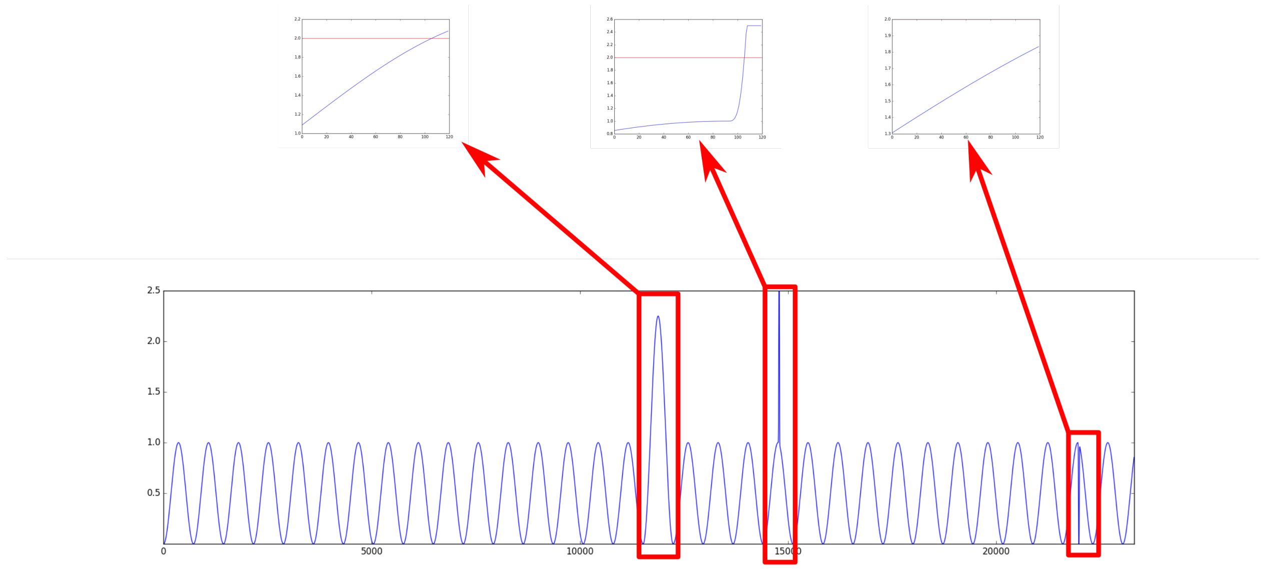


Figure 10.2 – User requests generated by the AIE

Table 10.3 – Overall performance of the AIE compared to Cloud-Monitor

Algorithm	Learning strategy	TPR	FPR	AUC	Time
Thresholding	None	0.37	0.29	0.53	1.6 μs
Cloud-Monitor	Local	0.85	0.62	0.62	20.9 μs
Cloud-Monitor	Global	0.98	0.95	0.51	27.7 μs
AIE	Local	0.75	0.44	0.65	20.9 μs
AIE	Global	0.91	0.81	0.55	28.1 μs

positive. Figure 10.2 shows the user requests graphs generated by the AIE, with the corresponding time series fragments and the alerting threshold. The beginning of the test time series is depicted at the bottom, with red squares representing the detected anomalies.

We can now focus on more realistic data. Using the same dataset and labels used to evaluate SCHEDA, we use a grid search to find the best settings for memory sensitivity and anomaly sensitivity, resulting in a 3-sigma rule for both. Note that this means that distances *superior* to the anomaly threshold are considered as anomalies, while distances *inferior* to the memory threshold are considered as matches. In this instance, the grid search validates the well-known 3-sigma rule [Pukelsheim, 1994].

The performance of the AIE on a real data set are summarized in Table 10.3. They can be contrasted with the benchmarks of Cloud-Monitor in Table 7.2. The time is measured per data point, *i.e.* reflects the time required to process a new sample in a single time series. As with the Cloud-Monitor benchmark, the True Positive Rate (TPR), False Positive Rate (FPR) and running time are measured, and the Area Under the ROC Curve (AUC) is computed from the TPR and FPR.

As with Cloud-Monitor, we find that the AUC, which is an overall performance indicator measuring the trade-off between true and false positives, is higher when using the local learning strategy, *i.e.* learning failure modes on each separately rather than globally. This point has been discussed in Section 7.5.3 for Cloud-Monitor. Though the AIE has a 12% lower TPR than

Cloud-Monitor, which is expected because, contrary to Cloud-Monitor, it filters the alerts, it has a higher AUC, meaning it is better at trading false positives for true positives. The TPR, *i.e.* the ratio of detected anomalies to the total number of anomalies, still remains above twice (103%) that of the monitoring system alone using only thresholding. By contrast, Cloud-Monitor provides a 130% increase in TPR over the monitoring system.

The FPR is 29% lower in the AIE than in Cloud-Monitor, but still 51% higher than a thresholding system. In Cloud-Monitor, however, it is 114% higher than the thresholding model.

The running time of the AIE and Cloud-Monitor are remarkably similar, both clocking at $20.9\mu s$ per point using the local learning strategy. This is 13 times slower than the thresholding model, which can be surprising given that thresholding entails literaly only this: comparing two values, while the AIE also performs a nearest-neighbour search and computes an anomaly score, both of which are significantly more complex operations.

10.3 Requirements evaluation

These benchmarks allow us to evaluate the AIE in light of our initial requirements, as defined in Section 1.3, p.14:

- *Low CPU and memory footprint:* the final evaluation shows that the CPU resources required to run the algorithm is sufficiently low, while the memory footprint may require optimizations. The best configuration, the full AIE with local learning, requires 20.9 μs to process a single data point. On a 4-core server, this scales to 11.5 million data points per minute, which exceeds by far the requirements of monitoring of tens of thousands of pints per minute. The AIE, because it uses the same database as the Cloud-Monitor memory, has no overhead with respect to Cloud-Monitor itself.
- *Real-time analysis:* the design of the AIE is inherently real-time.
- *Anomaly detection:* SCHEDA provides reliable anomaly detection and allows the full mode of Cloud-Monitor to be enabled on all series. When compared to the thresholding model, the AIE provides a 103% increase in true positive rate, meaning, by definition, a twofold increase in detected anomalies. Cloud-Monitor, without false positive filtering, provides a 130% increase.
- *Failure prediction:* the immune memory of Cloud-Monitor's lightweight mode allows failure modes to be remembered and alerts to be triggered even before the failures are detected by anomaly detection or threshold crossing.
- *Low configuration overhead (autonomy):* the default parameters (sensitivity of the signature matching, anomaly threshold and SES lags) can be used with satisfactory results on all series. In particular, the well-known three-sigma rule of thumb is confirmed experimentally and requires no further tuning.
- *Low false alarm rate:* the results show an 29% decrease in false positive rate with respect to Cloud-Monitor, at the cost of a 12% decrease in true positive rate. However, the effect should be measured relative to the base model: thresholding. Cloud-Monitor increases the false positive rate by 114%, while the increase with the AIE is only 51%. The actual usability of a system with this FPR has to be tested in a real deployment.

10.4 Conclusion

The AIE, in essence, is an immune-like overlay on a traditional monitoring system. It uses a repertoire of signatures to add a fast reaction mechanism, similar to the result of vaccination or prior exposure to a pathogen, and a self-tolerance property inspired by the negative selection reaction in the thymus. Furthermore, it introduces anomaly detection which, while not exactly an immune process, bears similarities with the naïve “self/non-self” understanding of immunity.

The expected properties of this type of immune overlay, as stated in the introduction of this thesis (Section 1.4, p.18), were the following:

- *Anomaly detection:* unknown patterns should trigger an alarm,
- *Memory:* known failure signals should trigger an alarm,
- *Tolerance:* known rare patterns should not trigger an alarm,
- *Adaptation:* obsolete signals should not trigger false alarms.

Only the first three were retained as objectives for this work.

Cloud-Monitor has demonstrated and experimentally validated anomaly detection and memory. It has shown, on artificial data, that memory provides a faster secondary immune reaction. The difficulty of labelling production data accurately, and in particular to mark the exact start of an event, prevents the failure prediction property to be accurately tested on real data; however, memory has been shown to lead to higher true positive rates on real data. The difference in false positive rate between Cloud-Monitor and the AIE, as well as the benchmarks of ALPAGA, validate the tolerance property: the AIE has a lower false positive rate than Cloud-Monitor, and overall a better classification performance, as measured by the area under its ROC curve.

Adaptation, however, has not been studied for two reasons. Firstly, it is partly covered by the self-tolerance property: signatures of failures that are now considered normal should simply be relabelled as such using the same process used to label new data. The ability for the expert to correct labels on the fly is an implementation point, not a research topic: as long as the samples are correctly labelled, the self-tolerance mechanism prevents false alarms. Secondly, the lack of long running time series in our datasets, due to the limitations of the monitoring system from which they were extracted, would have made it impossible to test on real data, and any conclusion on adaptation would have been speculation.

However, the three main properties of immune systems – anomaly detection, memory and tolerance – have been successfully injected into monitoring systems through the AIE overlay, thus confirming our initial hypotheses. The fact that it does not hinder the scalability of monitoring servers is one of the essential findings of this thesis: it shows that state-of-the-art neural networks trained on terabytes of data on GPU clusters are not the only way to tackle the challenges of a domain, like network monitoring, where machine learning is still in its infancy.

Chapter 11

Conclusion and roadmap

After this final evaluation, the time has come to reflect on the overall results of this work, on the Artificial Immune Ecosystem model and on its potential impact in the artificial immune system and the monitoring communities. The cross-disciplinary approach to network monitoring developed throughout this thesis reflects, in our opinion, the necessity of bringing together different research communities, in particular when one's focus is on building models and the other faces more down-to-earth challenges and has to adapt existing algorithms. The AIS and monitoring communities, being in this exact configuration, created the opportunity to bridge the gap between AIS and monitoring systems with the AIE model.

The initial goal of the AIE was to apply the concepts of immune computing to network monitoring. This basic concept drew in the notions of active learning as a path to immune-like tolerance, and the requirement of lightweight algorithms. Far from the scale of GPU clusters crunching millions of images, the AIE modestly processes thousands of time series on commodity servers, relying on extremely simple classifiers to improve the well-known and poorly tooled process of network monitoring.

Section 11.1 reflects on the contributions of the models described in this thesis – not only the AIE, but also its building blocks, Cloud-Monitor, ALPAGA and SCHEDA. It highlights the contributions of these elements, the precisions they bring on algorithm choice and design under constrained resources and the trade-offs involved in this kind of design.

Section 11.2 proposes a roadmap of future research on the AIE, possible improvements on the architecture and refinements in the models. While the AIE, in our opinion, should remain focussed on areas of application where embarrassingly parallel problems must be processed in constrained time with little computational power and low label availability for training samples. Many steps could be taken, however, to extend the abilities of the model for a low computational overhead. In fact, joint learning, *i.e.* learning the same model for multiple time series, could even *reduce* the current cost of the AIE. A rule processor, be it an expert system or a Learning Classifier System, could also replace the thresholding model and give a much more accurate initial classifier on which the AIE could improve. Finally, monitoring the activation level of the AIE, as a true immune system does, would bring the additional property of *self-regulation*, which could potentially make the false positive rate, if not *lower*, at least *predictable* and even *tunable*.

11.1 Contributions of the AIE

We can now summarize the contributions made by the AIE in the monitoring, machine learning and time series analysis communities. We first position the AIE with respect to the AIS, then each of its component with respect to its related field. Naturally, Cloud-Monitor, as a detection system or a "proto-AIE", and the full AIE as a meta-algorithm verifying all the immune properties, are targeted towards monitoring, while ALPAGA is a machine learning framework and SCHEDA a time serie analysis algorithm.

11.1.1 AIE and AIS

Emma Hart, in [Hart et al., 2003], proposed a classification of AIS models into three categories based on their relationship to biological immune systems: *literal* models aiming to replicate the *purpose* of an immune system, *e.g.* in computer security, *metaphorical* models drawing inspiration from some of the processes found in immune systems, *e.g.* the clonal expansion of lymphocytes, to design new algorithms, and *simulation* models trying to accurately replicate the biological immune system *in silico* to better understand it.

Our approach with the AIE is somewhat the literal and the metaphorical; however, the AIE does not pretend to bring *immunity* to a target system – only to use an immune-like model to detect and classify events in streaming data. Moreover, it is not based on immune *processes* but on *properties*. We suggest that the AIE belongs to a *functional* approach to artificial immunity: it considers what specific properties the immune system displays while performing its task and attempts to provide them. In particular:

- Cloud-Monitor provides memory and anomaly detection.
- ALPAGA provides memory and tolerance.
- SCHEDA provides anomaly detection and scalability.

These properties are summarized in Table 5.1. Note that the final AIE, as depicted in Figure 10.1, takes full advantage of the overlap of these properties by using SCHEDA's anomaly detection ability in place of Cloud-Monitor's, thus achieving scalability, and by combining the memories of Cloud-Monitor and ALPAGA, thus building a single immune memory – or a single repertoire of memory T-cells.

11.1.2 Cloud-Monitor and the AIE

Cloud-Monitor and the AIE address the intersection of three problems that arise in monitoring: the scale of the domain, the need for advanced analytics such as anomaly detection, and the low resources allocated to it. Any two of these three can be picked at the expense of the third: many modern implementations presented in research [Lin et al., 2016, Zhou et al., 2018, Wang et al., 2016b, Macit et al., 2016, Gupta et al., 2016] get analytics at scale by using stream computing frameworks such as Apache Spark, which use high power dedicated compute nodes. Traditional, commercial monitoring suites such as Nagios sacrifice time series analysis completely to achieve scalability on commodity hardware, which completely prevents them from detecting any failure mode that does not involve a metric crossing a threshold, *e.g.* a server becoming active at an odd time, or a highly loaded one suddenly dropping in CPU usage. Finally, methods based on neural networks can achieve good results, but are generally trained on a single time series [Janssens et al., 2016, Bahmanyar and Karami, 2014].

Cloud-Monitor, by applying anomaly detection to the collected time series and using a special memory to keep track of specific events, provides analytics at essentially the same scale as any simpler monitoring system, and does not require any specific compute infrastructure. The full AIE can detect new classes of anomalies – mainly context, or structural anomalies – and tolerate specific deviations from the norm when instructed to do so by a human operator, at a computational cost low enough to have no effect on scalability.

From the immune system perspective, the AIE verifies the expected core properties of anomaly detection, memory and tolerance, the former two provided by Cloud-Monitor and SCHEDA and the latter by ALPAGA. From a monitoring perspective, we have shown that the memory and anomaly detection properties lead to satisfactory failure prediction. Configuration overhead is non-existent, as the thresholds of Cloud-Monitor can simply be set to follow the three-sigma rule: any subsequence more than three standard deviations away from the average distance is considered an anomaly. The additional computational cost is well below what would reduce the scalability of a monitoring system.

The accuracy of the AIE can still be challenged. While it detects behavioural anomalies and threshold crossings correctly, it still generates a higher false positive rate than a basic monitoring system (about 50% higher). We expect further work on filtering and joint learning, as described in Section 11.2, to help tackle this limitation.

Finally, let us highlight the differences between the AIE and a typical AIS. Firstly, the AIE does not attempt to simulate an immune system with any level of accuracy; like Watkins' and Timmis' AIRS [Watkins et al., 2004], we only draw inspiration from immunity. Secondly, the influence of immunity on the design of the AIE is visible at the level of *properties*, not *processes*. Thirdly, the AIE is not a single algorithm *per se;* rather, it is a framework within which other algorithms operate. In this regard, it is more similar to a Learning Classifier System, where the choice of evolutionary algorithm and reward function is largely left to the implementor, than to an AIS.

11.1.3 ALPAGA

ALPAGA solves the immune tolerance problem by using the expert to train a filter. The algorithm can be seen in two ways. In one regard it is an overlay on top of a binary classifier that adds a second classification step when *one* of the classes is predicted. In the context of the AIE, it is used to filter the false positives generated by Cloud-Monitor and, as such, can be likened to a very minimalist form of bootstrap aggregator [Freund and Schapire, 1995].

In another regard, more akin to the way it is described in Chapter 8, it is a way to transform a dataset and correct class imbalance by introducing a heuristic between the raw data and the classifier. This is still a relatively unexplored technique in active learning, where *internal* heuristics, *e.g.* uncertainty in a classifier's reported prediction, are way more prevalent than *external* heuristics, *i.e.* domain-driven metrics, as used by ALPAGA.

This dual view on ALGAPA is also a reflection on active learning as a whole, and how it integrates in the machine learning landscape. In particular, it shows that the performance of a classifier can be significantly improved by allowing it to communicate with a human expert, without any considerable computational overhead and without any modification to the original classifier.

ALPAGA also occupies a specific niche in that it does not *replace* a classifier; instead, it adds a layer between a system and its output. This approach of active learning, combining the use of external heuristics and the complementary role of the active classifier, is still novel and has only been developed by a handful of authors [Veeramachaneni et al., 2016]. The emphasis of minimalist, lightweight algorithms in this context is also new and, to the best of our knowledge, has not been explored. Through the benchmarks of various algorithms in extreme and unusual configurations, we have gained insights on the behaviour of well-known systems when pushed to their limits in terms of available training data and computational power. It is our hope that more focus be put on minimalistic methods, and that the trade-off between on the one hand data and computational complexity, and on the other hand accuracy improvement, be explored more thoroughly.

11.1.4 SCHEDA

SCHEDA is an attempt to bridge the gap between the limitations of lightweight anomaly detection algorithms and costly distance-based methods. It does so by becoming domain-specific, *i.e.* by introducing a clear boundary in its domain of correct operation; namely, it requires that the time series be periodic, and of a known periodicity. Contrary to seasonal extensions of auto-regressive models, however, it can tolerate any number of nested patterns, *e.g.* monthly, weekly, daily, hourly...

When compared to SAX-based algorithms, which are its closest contenders, SCHEDA reaches the same level of performance in terms of CPU and memory costs, but also functions in real time and does not exhibit some of the failure modes that result from the assumptions of this family of models. While the assumptions it makes on the data are not directly transferable to all possible applications, mainly because the period of the signal is not necessarily known, workarounds are easy to derive, for instance using peak detection and autocorrelation. Moreover, time series of a *known* period are common, as we saw earlier, in signals generated by human activity.

11.2 Roadmap and perspectives

This section presents the improvements that should, given the performance of the current model, be added to the AIE. It also highlights the contributions connected to the AIE model, though not directly part of this thesis.

11.2.1 Improvement of the Cloud-Monitor model

While Cloud-Monitor, especially when embedded in the AIE framework, is able to achieve detection performance comparable to distance-based methods and much higher than simple thresholding allows, the model has a number of limitations. The primary issue is the sharing of signatures. The benchmarks show that sharing the signature database leads to extremely high false positive rates, but also to the detection of more anomalies.

From a domain perspective, this result is not surprising: the time series obtained in network monitoring are extremely diverse and there is no reason to expect much learning transferrability between different *types* of series. However, the benchmarks of ALPAGA also show that, between series of the same type, learning transfer can occur (see Section 8.4.6 for details).

This opens the perspective of using time series clustering, with a lightweight algorithm such as the SAX/Chaos Game bitmap representation [Wei et al., 2005], to partition the times into similar groups between which learning transfer can occur. Then the signature database, or immune memory, could be shared across series of the same cluster, but not across different clusters. It is worth poiting out that methods for decentralized clustering such as DeCENTREx [Pastor et al., 2017] allow the cluster definitions to be maintained in a distributed system.

11.2.2 Refinement of the AIE

Another research topic would be the further reduction of false positives, even to a lower point than the thresholding model. One way to tackle this issue would be to use the notion of self-regulation from the immune system, and to automatically adjust the detection thresholds of the AIE to match a target "activation level", *i.e.* a reaction, or alerting, rate. Beside the immune metaphor, this approach is similar to the Constant False Alarm Rate (CFAR) detection algorithm, notably used in radars [Bouchelaghem et al., 2019] but also extended to detect faults in industrial machinery [Trachi et al., 2016].

The notion of adaptation should also be explored, as it has been left out of this thesis. The implementation could be as simple as the deletion of the oldest entries in the immune memory, or much more advanced; the lack of real-world data to demonstrate and validate the concept makes it hard to benchmark in non-trivial scenarios. A study of aging in artificial immune systems, including the AIE, could be the starting point of such endeavour.

Another improvement would be achieved by replacing the thresholding model by a better rule engine. If the expert systems of the 1980s no longer the trending AI model, it still has a tremendous impact, in particular in medicine [Abu-Nasser, 2017], as a tool to extract and apply domain knowledge. Capturing expert knowledge without a tedious process of rule writing can also be accomplished by Learning Classifier Systems and their extensions, such as the XCS [Arif et al., 2018]. In any case, replacing a simple thresholding rule by a more complete description of the expected behaviour of a metric, which is generally known to the domain experts, would undoubtedly improve the performance of the AIE.

11.2.3 Perspectives for the AIE

The AIE is particular in the machine learning landscape in that it imposes a very specific architecture but leaves the actual choice of algorithms open. In its essence, it requires a rule-based detector, a memory and an anomaly detector. An adaptation of this architecture for smart building monitoring was published as the Artificial aWareness Architecture (AWA) in [Parrend et al., 2017]. The memory should be able to accept labels from an external user and classify inputs even with small training sets. The input data flow through the memory, rule-based detector and anomaly detector, with "reactive" samples fed back to the memory module along with a label.

Interestingly, time series are not necessarily the only application of the AIE. We can see how it would be relevant in other areas where real-time stream processing can add significant value. One obvious domain would be event analysis, *e.g.* log processing. Contrary to time series data, event streams are not regularly sampled and generally not real-valued: a typical log entry contains text that identifies the context of the event and then some more to describe the event itself. The role of the AIE in expert-guided analysis systems is explored in [Parrend et al., 2018a].

An argument could therefore be made that the AIE is not a machine learning algorithm *per se*, but rather a form of blueprint for such algorithms. This is often referred to in the software community as a *design pattern* [Vlissides et al., 1995, Fowler, 2002]. This brings the notion of the AIE as a design pattern for stream analysis systems with a human in the loop. This approached was published in [Parrend et al., 2018b].

11.3 Conclusion

The AIE was the first step towards a new understanding of the immune metaphor in computing, not as an inspiration for an algorithm but as a set of properties that stem from particular interactions between its various components. The model and the implementation presented and benchmarked in this thesis are not the final form of the AIE; rather, they constitute a proof-of-concept of the blueprint, or design pattern, applied to the problem of network monitoring time series analysis.

The AIE demonstrates that the immune system can be an inspiration not only for algorithms, but also for software architecture. The success of this first model shows that the desirable properties of the biological immune system can be implemented by *interactions* instead of the algorithms themselves. We feel this new angle on AIS, as well as the design pattern approach, are a way of creating a new bridge between immunologists and computer scientists, as the AIS research of the 1990s did.

The obvious next step for the AIE is a real-world deployment in an ISP-scale network, with actual users populating the memory and labelling the incidents. This is outside the research scope of the AIE, but very much a planned step. Remaining research topics mostly entail making the AIE "more immune": focussing on secondary properties of the immune system to achieve better accuracy.

The other essential work to do around the AIE is to use the same blueprints and interactions on a different data type. We feel log analysis would be a great use case and demonstrate the universality of the approach. To this end, other distance measures, heuristics and classifiers would have to be used, but if their interactions remain the same, *i.e.* the data passes through the immune memory, the rule-based detector and the anomaly detector and alerts are filtered by ALPAGA based on the same memory, then the concept of an Artificial Immune Ecosystem would have proven to be applicable to scenarios other than time series analysis.

Bibliography

E. Abbasi and N. Naghavi. Offline auto-tuning of a pid controller using extended classifier system (xcs) algorithm. *Journal of Advances in Computer Engineering and Technology*, 3(1):41–44, 2017.

B. Abu-Nasser. Medical expert systems survey. *International Journal of Engineering and Information Systems (IJEAIS)*, 1(7):218–224, 2017.

D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine learning*, 6(1): 37–66, 1991.

S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 2017.

U. Aickelin and S. Cayzer. The danger theory and its application to artificial immune systems. *arXiv preprint arXiv:0801.3549*, 2008.

U. Aickelin, J. Greensmith, and J. Twycross. Immune system approaches to intrusion detection–a review. In *International Conference on Artificial Immune Systems*, pages 316–329. Springer, 2004.

H. Aidos, R. P. Duin, and A. L. Fred. The area under the roc curve as a criterion for clustering evaluation. In *ICPRAM*, pages 276–280, 2013.

R. Akbani, S. Kwek, and N. Japkowicz. Applying support vector machines to imbalanced datasets. In *European conference on machine learning*, pages 39–50. Springer, 2004.

A. Akusok, E. Eirola, Y. Miche, A. Gritsenko, and A. Lendasse. Advanced query strategies for active learning with extreme learning machine. In *The European Symposium on Artificial Neural Networks (ESANN), upcoming*, 2017.

K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, and V. Bhatnagar. An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. *Computing*, 97(4):357–377, 2015.

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

I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. D. Spyropoulos, and P. Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *arXiv preprint cs/0009009*, 2000.

M. H. Arif, J. Li, M. Iqbal, and K. Liu. Sentiment analysis and spam detection in short informal text using learning classifier systems. *Soft Computing*, 22(21):7281–7291, 2018.

K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

A. Bahmanyar and A. Karami. Power system voltage stability monitoring using artificial neural networks with a reduced set of inputs. *International Journal of Electrical Power & Energy Systems*, 58:246–256, 2014.

M. Barnsley. Fractals everywhere, 1993.

B. Barz, E. Rodner, Y. G. Garcia, and J. Denzler. Detecting regions of maximal divergence for spatio-temporal anomaly detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

K. Begnum and M. Burgess. Improving anomaly detection event analysis using the eventrank algorithm. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 145–155. Springer, 2007.

J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. URL `http://doi.acm.org/10.1145/361002.361007`.

E. Bernadó-Mansilla and J. M. Garrell-Guiu. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary computation*, 11(3):209–238, 2003.

H. Bersini. Self-assertion versus self-recognition: A tribute to francisco varela. In *Published in the proceedings of 1st International Conference on Artificial Immune Systems (ICARIS), University of Kent at Canterbury, UK*, 2002.

H. Bersini and F. J. Varela. Hints for adaptive problem solving gleaned from immune networks. In *International Conference on Parallel Problem Solving from Nature*, pages 343–354. Springer, 1990.

M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: methods, systems and tools. *Ieee communications surveys & tutorials*, 16(1):303–336, 2014.

M. Bloodgood. Support vector machine active learning algorithms with query-by-committee versus closest-to-hyperplane selection. In *Semantic Computing (ICSC), 2018 IEEE 12th International Conference on*, pages 148–155. IEEE, 2018.

H. E. Bouchelaghem, M. Hamadouche, F. Soltani, and K. Baddari. Distributed clutter-map constant false alarm rate detection using fuzzy fusion rules. *Radioelectronics and Communications Systems*, 62(1):1–5, 2019.

C. Boufenar, M. Batouche, and M. Schoenauer. An artificial immune system for offline isolated handwritten arabic character recognition. *Evolving Systems*, 9(1):25–41, 2018.

L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

F. Breitinger and H. Baier. Similarity preserving hashing: Eligible properties and a new algorithm mrsh-v2. In *International conference on digital forensics and cyber crime*, pages 167–182. Springer, 2012.

T. M. Breuel. Benchmarking of lstm networks. *arXiv preprint arXiv:1508.02774*, 2015.

P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting.* springer, 2016.

J. Brownlee. Clonal selection theory & clonalg-the clonal selection classification algorithm (csca). *Swinburne University of Technology*, 2005.

J. D. Brutlag. Aberrant behavior detection in time series for network monitoring. In *LISA*, volume 14, pages 139–146, 2000.

M. Bucurica, R. Dogaru, and I. Dogaru. A comparison of extreme learning machine and support vector machine classifiers. In *Intelligent Computer Communication and Processing (ICCP), 2015 IEEE International Conference on*, pages 471–474. IEEE, 2015.

L. Bull and T. O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 905–911. Morgan Kaufmann Publishers Inc., 2002.

M. Burgess. Two dimensional time-series for anomaly detection and regulation in adaptive systems. In *International Workshop on Distributed Systems: Operations and Management*, pages 169–180. Springer, 2002.

M. Burgess. Probabilistic anomaly detection in distributed computer networks. *Science of Computer Programming*, 60(1):1–26, 2006.

M. V. Butz and S. W. Wilson. An algorithmic description of xcs. *Soft Computing*, 6(3-4):144–153, 2002.

A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. isax 2.0: Indexing and mining one billion time series. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 58–67. IEEE, 2010.

A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh. Beyond one billion time series: indexing and mining very large time series collections with i sax2+. *Knowledge and information systems*, 39(1):123–151, 2014.

M. Celenk, T. Conley, J. Graham, and J. Willis. Anomaly prediction in network traffic using adaptive wiener filtering and arma modeling. In *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, pages 3548–3553. IEEE, 2008.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

P. Chaovalit and L. Zhou. Movie review mining: A comparison between supervised and unsupervised classification approaches. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 112c–112c. IEEE, 2005.

M.-H. Chen, C.-H. Teng, and P.-C. Chang. Applying artificial immune systems to collaborative filtering for movie recommendation. *Advanced Engineering Informatics*, 2015.

N. Chinchor and B. Sundheim. Muc-5 evaluation metrics. In *Proceedings of the 5th conference on Message understanding*, pages 69–78. Association for Computational Linguistics, 1993.

K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

M. C. Chuah and F. Fu. Ecg anomaly detection via time series analysis. In *International Symposium on Parallel and Distributed Processing and Applications*, pages 123–135. Springer, 2007.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

J. J. Commandeur and S. J. Koopman. *An introduction to state space time series analysis.* Oxford University Press, 2007.

J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks*, 5(2):240–254, 1994.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967. ISSN 0018-9448. doi: 10.1109/TIT.1967.1053964.

C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 647–651. ACM, 2003.

V. Cutello, G. Nicosia, and M. Pavone. Exploring the capability of immune algorithms: A characterization of hypermutation operators. In *International Conference on Artificial Immune Systems*, pages 263–276. Springer, 2004.

A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho. Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 27–35. IEEE, 2016.

K. Das, I. Avrekh, B. Matthews, M. Sharma, and N. Oza. Ask-the-expert: Active learning based knowledge discovery using the expert. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 395–399. Springer, 2017.

D. Dasgupta and S. Forrest. Novelty detection in time series data using ideas from immunology. In *In Proceedings of The International Conference on Intelligent Systems*, 1995.

D. Dasgupta and S. Forrest. Artificial immune systems in industrial applications. In *Intelligent Processing and Manufacturing of Materials, 1999. IPMM'99. Proceedings of the Second International Conference on*, volume 1, pages 257–267. IEEE, 1999.

A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya. A dependency-aware ontology-based approach for deploying service level agreement monitoring services in cloud. *Software: Practice and Experience*, 42 (4):501–518, 2012.

L. N. De Castro and F. J. Von Zuben. Artificial immune systems: Part i–basic theory and applications. *Universidade Estadual de Campinas, Dezembro de, Tech. Rep*, 210(1), 1999.

L. N. de Castro and F. J. Von Zuben. ainet: an artificial immune network for data analysis. *Data mining: a heuristic approach*, 1:231–259, 2001.

L. N. De Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE transactions on evolutionary computation*, 6(3):239–251, 2002.

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

A. F. de la Cruz, J. P. Muñoz-Gea, P. Manzanares-Lopez, and J. Malgosa-Sanahuja. Network failures support for traffic monitoring mechanisms in software-defined networks. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 691–694. IEEE, 2016.

A. M. De Livera, R. J. Hyndman, and R. D. Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.

L. Deng and D. Yu. Deep convex net: A scalable architecture for speech pattern classification. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

S. Dieleman, A. v. d. Oord, and K. Simonyan. The challenge of realistic music generation: modelling raw audio at scale. *arXiv preprint arXiv:1806.10474*, 2018.

T. Dillon, C. Wu, and E. Chang. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010.

T.-N. Do, P. Lenca, S. Lallich, and N.-K. Pham. Classifying very-high-dimensional data with random forests of oblique decision trees. In *Advances in knowledge discovery and management*, pages 39–55. Springer, 2010.

M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

J. Durbin. The fitting of time-series models. *Revue de l'Institut International de Statistique*, pages 233–244, 1960.

E. Earl and R. E. Ladner. Enhanced sequitur for finding structure in data. In *Data Compression Conference, 2003. Proceedings. DCC 2003*, page 425. IEEE, 2003.

J. Ehrensperger and J. Conradt. Growing neural gas for time series evaluation. 2015.

J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

L. Elsen, F. Kohn, C. Decker, and R. Wattenhofer. goprobe: a scalable distributed network monitoring solution. In *Peer-to-Peer Computing (P2P), 2015 IEEE International Conference on*, pages 1–10. IEEE, 2015.

M. Espinoza, C. Joye, R. Belmans, and B. De Moor. Short-term load forecasting, profile identification, and customer segmentation: a methodology based on periodic time series. *IEEE Transactions on Power Systems*, 20(3):1622–1630, 2005.

P. G. Ferreira, P. J. Azevedo, C. G. Silva, and R. M. Brito. Mining approximate motifs in time series. In *Discovery Science*, pages 89–101. Springer, 2006.

G. Forman and I. Cohen. Learning from little: Comparison of classifiers given little training. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 161–172. Springer, 2004.

S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *null*, page 202. Ieee, 1994.

S. Forrest, S. A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40(10): 88–96, 1997.

M. Fowler. *Patterns of enterprise application architecture.* Addison-Wesley Longman Publishing Co., Inc., 2002.

A. A. Freitas and J. Timmis. Revisiting the foundations of artificial immune systems: A problem-oriented perspective. In *International Conference on Artificial Immune Systems*, pages 229–241. Springer, 2003.

A. A. Freitas and J. Timmis. Revisiting the foundations of artificial immune systems for data mining. *IEEE Transactions on Evolutionary Computation*, 11(4):521–540, 2007.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. *arXiv preprint arXiv:1703.02910*, 2017.

P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009.

J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93, 1993.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

M. Gong, L. Jiao, L. Zhang, and H. Du. Immune secondary response and clonal selection inspired optimizers. *Progress in Natural Science*, 19(2):237–253, 2009.

J. Greensmith and U. Aickelin. Artificial dendritic cells: multi-faceted perspectives. In *Human-Centric Information Processing Through Granular Modelling*, pages 375–395. Springer, 2009.

J. Greensmith, U. Aickelin, and S. Cayzer. Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. In *International Conference on Artificial Immune Systems*, pages 153–167. Springer, 2005.

J. Greensmith, A. Whitbrook, and U. Aickelin. Artificial immune systems. In *Handbook of Metaheuristics*, pages 421–448. Springer, 2010.

K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

F. Guigou, P. Parrend, and P. Collet. An artificial immune ecosystem model for hybrid cloud supervision. In *First Complex Systems Digital Campus World E-Conference 2015*, pages 71–84. Springer, 2015.

F. Guigou, P. Collet, and P. Parrend. Anomaly detection and motif discovery in symbolic representations of time series (rapport de recherche). *arXiv preprint arXiv:1704.05325*, 2017a.

F. Guigou, P. Collet, and P. Parrend. The artificial immune ecosystem: a bio-inspired meta-algorithm for boosting time series anomaly detection with expert input. In *European Conference on the Applications of Evolutionary Computation*, pages 573–588. Springer, 2017b.

A. Gupta, R. Birkner, M. Canini, N. Feamster, C. Mac-Stoker, and W. Willinger. Network monitoring as a streaming analytics problem. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 106–112. ACM, 2016.

K. Gurney. *An introduction to neural networks.* CRC press, 2014.

R. Haertel, E. Ringger, K. Seppi, J. Carroll, and P. McClanahan. Assessing the costs of sampling methods in active learning for annotation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 65–68. Association for Computational Linguistics, 2008.

A. A. Haidar. *An adaptive document classifier inspired by t-cell cross-regulation in the immune system.* PhD thesis, Citeseer, 2011.

A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Łapacz, D. M. Swany, S. Trocha, and J. Zurawski. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *International Conference on Service-Oriented Computing*, pages 241–254. Springer, 2005.

E. Hart and J. Timmis. Application areas of ais: The past, the present and the future. *Applied soft computing*, 8(1):191–201, 2008.

E. Hart, P. Ross, A. Webb, and A. Lawson. A role for immunology in "next generation" robot controllers. In J. Timmis, P. J. Bentley, and E. Hart, editors, *Artificial Immune Systems*, pages 46–56, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45192-1.

A. E. Hauser and U. E. Höpken. B cell localization and migration in health and disease. In *Molecular Biology of B Cells (Second Edition)*, pages 187–214. Elsevier, 2015.

J. He and J. G. Carbonell. Nearest-neighbor-based active learning for rare category detection. In *Advances in neural information processing systems*, pages 633–640, 2008.

J. Heck and F. M. Salem. Simplified minimal gated unit variations for recurrent neural networks. *arXiv preprint arXiv:1701.03452*, 2017.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

S. A. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, pages 1289–1296. Morgan Kaufmann Publishers Inc., 1999.

J. H. Holland. Escaping brittleness. In *Proceedings Second International Workshop on Machine Learning*, pages 92–95. Citeseer, 1983.

G. Holmes, A. Donkin, and I. H. Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.

T. Hothorn, K. Hornik, and A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15(3):651–674, 2006.

O. Janssens, V. Slavkovikj, B. Vervisch, K. Stockman, M. Loccufier, S. Verstockt, R. Van de Walle, and S. Van Hoecke. Convolutional neural network based fault detection for rotating machinery. *Journal of Sound and Vibration*, 377:331–345, 2016.

N. K. Jerne. Towards a network theory of the immune system. *Ann. Immunol.*, 125:373–389, 1974.

R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

E. Keogh and J. Lin. Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and information systems*, 8(2):154–177, 2005.

E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286, 2001.

E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM, 2004.

E. Keogh, J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Data mining, fifth IEEE international conference on*, pages 8–pp. IEEE, 2005.

S. Khan and A. Ahmad. Relationship between variants of one-class nearest neighbours and creating their accurate ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

J. Kim and P. J. Bentley. Towards an artificial immune system for network intrusion detection: An investigation of dynamic clonal selection. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1015–1020. IEEE, 2002.

N. Kumar, V. N. Lolla, E. J. Keogh, S. Lonardi, and C. A. Ratanamahatana. Time-series bitmaps: a practical visualization tool for working with large time series databases. In *SDM*, pages 531–535. SIAM, 2005.

N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1939–1947. ACM, 2015.

A. Lazarevic and V. Kumar. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166. ACM, 2005.

Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

D.-T. Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE transactions on computers*, 100(6): 478–487, 1982.

T. Lee, Y. Xiao, X. Meng, and D. Duling. Clustering time series based on forecast distributions using kullback-leibler divergence. 2014.

D. D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning Proceedings 1994*, pages 148–156. Elsevier, 1994.

J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM, 2003.

J. Lin, M. Vlachos, E. Keogh, and D. Gunopulos. Iterative incremental clustering of time series. In *Advances in Database Technology-EDBT 2004*, pages 106–122. Springer, 2004.

J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

J. Lin, Q. Zhang, H. Bannazadeh, and A. Leon-Garcia. Automated anomaly detection and root cause analysis in virtualized cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 550–556. IEEE, 2016.

R.-S. Lin, D. A. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 848–854. IEEE, 2010.

Y. Lin, Y. Zhang, Q. Li, and J. Yang. Supporting efficient query processing on compressed xml files. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 660–665. ACM, 2005.

A. Liotta, G. Pavlou, and G. Knight. Exploiting agent mobility for large-scale network monitoring. *IEEE network*, 16(3):7–15, 2002.

Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

X.-Y. Liu, J. Wu, and Z.-H. Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2009.

M. Lopes, F. Melo, and L. Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer, 2009.

M. Macit, E. Delibaş, B. Karanlık, A. İnal, and T. Aytekin. Real time distributed analysis of mpls network logs for anomaly detection. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 750–753. IEEE, 2016.

V. Malbasa, C. Zheng, P.-C. Chen, T. Popovic, and M. Kezunovic. Voltage stability prediction using active machine learning. *IEEE Transactions on Smart Grid*, 8(6):3117–3124, 2017.

P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

S. Malinowski, T. Guyet, R. Quiniou, and R. Tavenard. 1d-sax: A novel symbolic representation for time series. In *International Symposium on Intelligent Data Analysis*, pages 273–284. Springer, 2013.

J. Masci, A. M. Bronstein, M. M. Bronstein, P. Sprechmann, and G. Sapiro. Sparse similarity-preserving hashing. *arXiv preprint arXiv:1312.5479*, 2013.

P. Matzinger. Tolerance, danger, and the extended family. *Annual review of immunology*, 12(1):991–1045, 1994.

P. Matzinger. The danger model: a renewed sense of self. *Science*, 296(5566):301–305, 2002.

A. Mayzaud, A. Sehgal, R. Badonnel, I. Chrisment, and J. Schönwälder. Using the rpl protocol for supporting passive monitoring in the internet of things. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 366–374. IEEE, 2016.

M. Mdini, A. Blanc, G. Simon, J. Barotin, and J. Lecoeuvre. Monitoring the network monitoring system: Anomaly detection using pattern recognition. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 983–986. IEEE, 2017.

P. Mell, T. Grance, et al. The nist definition of cloud computing. *National institute of standards and technology*, 53(6):50, 2009.

Y. Mi. Imbalanced classification based on active learning smote. *Research Journal of Applied Science Engineering and Technology*, 5:944–949, 2013.

M. L. Minsky. Some methods of artificial intelligence and heuristic programming. In *Proc. Symposium on the Mechanization of Thought Processes, Teddington*, 1958.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

C. Molina-Jimenez, S. Shrivastava, J. Crowcroft, and P. Gevros. On the monitoring of contractual service level agreements. In *Electronic Contracting, 2004. Proceedings. First IEEE International Workshop on*, pages 1–8, July 2004. doi: 10.1109/WEC.2004.1319502.

L. Montechiesi, M. Cocconcelli, and R. Rubini. Artificial immune system via euclidean distance minimization for anomaly detection in bearings. *Mechanical Systems and Signal Processing*, 2015.

F. Mörchen, A. Ultsch, and O. Hoos. Extracting interpretable muscle activation patterns with time series knowledge mining. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 9(3): 197–208, 2005.

A. P. Muniyandi, R. Rajeswari, and R. Rajaram. Network anomaly detection by cascading k-means clustering and c4. 5 decision tree algorithm. *Procedia Engineering*, 30:174–182, 2012.

M. Mustafa, M. Taib, Z. Murat, and N. Sulaiman. Comparison between knn and ann classification in brain balancing application via spectrogram image. *Journal of Computer Science & Computational Mathematics*, 2(4):17–22, 2012.

C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical strcture in sequences: A linear-time algorithm. *J. Artif. Intell. Res.(JAIR)*, 7:67–82, 1997.

A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

C. Oksel, D. A. Winkler, C. Y. Ma, T. Wilkins, and X. Z. Wang. Accurate and interpretable nanosar models from genetic programming-based decision tree construction approaches. *Nanotoxicology*, 10(7): 1001–1012, 2016.

F. Olsson. A literature survey of active machine learning in the context of natural language processing. 2009.

K. D. Omilusik and A. W. Goldrath. The origins of memory t cells. *Nature*, 552:337–339, 2017.

S. M. Omohundro. Five balltree construction algorithms. Technical report, 1989.

G. H. Orcutt and J. Irwin. A study of the autoregressive nature of the time series used for tinbergen's model of the economic system of the united states, 1919-1932. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(1):1–53, 1948.

T. Osugi, D. Kim, and S. Scott. Balancing exploration and exploitation: A new algorithm for active machine learning. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

D. Palmisano, P. L. Ventre, A. Caponi, G. Siracusano, S. Salsano, M. Bonola, and G. Bianchi. D-streamon—nfv-capable distributed framework for network monitoring. In *Teletraffic Congress (ITC 29), 2017 29th International*, volume 2, pages 30–35. IEEE, 2017.

P. Parrend, P. David, F. Guigou, C. Pupka, and P. Collet. The awa artificial emergent awareness architecture model for artificial immune ecosystems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 403–410. IEEE, 2017.

P. Parrend, F. Guigou, J. Navarro, A. Deruyver, and P. Collet. Artificial immune ecosystems: the role of expert-based learning in artificial cognition. *Journal of Robotics, Networking and Artificial Life*, 4(4): 303–307, 2018a.

P. Parrend, F. Guigou, J. Navarro, A. Deruyver, and P. Collet. For a refoundation of artificial immune system research: Ais is a design pattern. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1122–1129. IEEE, 2018b.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

D. Pastor, E. Dupraz, and F.-X. Socheleau. Decentralized clustering based on robust estimation and hypothesis testing. *arXiv preprint arXiv:1706.03537*, 2017.

P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 370–377. IEEE, 2002.

D. Pelleg and A. W. Moore. Active learning for anomaly and rare-category detection. In *Advances in neural information processing systems*, pages 1073–1080, 2005.

W. Perrizo, Q. Ding, and A. Denton. Lazy classifiers using p-trees. In *CAINE*, pages 176–179, 2002.

D. T. Pham and M. Castellani. The bees algorithm: modelling foraging behaviour to solve continuous optimization problems. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 223(12):2919–2938, 2009.

P. Polewski, W. Yao, M. Heurich, P. Krzystek, and U. Stilla. Active learning approach to detecting standing dead trees from als point clouds combined with aerial infrared imagery. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 10–18, 2015.

F. Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, 1994.

K. Qian, Z. Zhang, A. Baird, and B. Schuller. Active learning for bird sound classification via a kernel-based extreme learning machine. *The Journal of the Acoustical Society of America*, 142(4): 1796–1804, 2017.

J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description lenght principle. *Information and computation*, 80(3):227–248, 1989.

V. Radhakrishna, P. V. Kumar, and V. Janaki. A novel similar temporal system call pattern mining for efficient intrusion detection. *J. UCS*, 22(4):475–493, 2016.

M. Raghuram, K. Akshay, and K. Chandrasekaran. Efficient user profiling in twitter social network using traditional classifiers. In *Intelligent Systems Technologies and Applications*, pages 399–411. Springer, 2016.

T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

D. Reker, P. Schneider, and G. Schneider. Multi-objective active machine learning rapidly improves structure–activity models and reveals new protein–protein interaction inhibitors. *Chemical Science*, 7(6): 3919–3927, 2016.

C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH computer graphics*, volume 21, pages 25–34. ACM, 1987.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, and F. Casati. Automated sla monitoring for web services. In M. Feridun, P. Kropf, and G. Babin, editors, *Management Technologies for E-Commerce and E-Business Applications*, volume 2506 of *Lecture Notes in Computer Science*, pages 28–41. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-00080-8. doi: 10.1007/3-540-36110-3_6. URL `http://dx.doi.org/10.1007/3-540-36110-3_6`.

T. Saito and M. Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):10, 2010.

S. Salvador and P. Chan. Learning states and rules for detecting anomalies in time series. *Applied Intelligence*, 23(3):241–255, 2005.

A. Sant'Anna and N. Wickström. Symbolization of time-series: An evaluation of sax, persist, and aca. In *Image and Signal Processing (CISP), 2011 4th International Congress on*, volume 4, pages 2223–2228. IEEE, 2011.

R. Sathya and A. Abraham. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2):34–38, 2013.

N. Saunier, S. Midenet, and A. Grumbach. Stream-based learning through data selection in a road safety application. In *STAIRS 2004, Proceedings of the Second Starting AI Researchers' Symposium*, volume 109, pages 107–117, 2004.

O. Sener and S. Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

P. Senin and S. Malinchik. Sax-vsm: Interpretable time series classification using sax and vector space model. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1175–1180. IEEE, 2013.

P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, S. Frankenstein, and M. Lerner. Grammarviz 2.0: a tool for grammar-based pattern discovery in time series. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 468–472. Springer, 2014.

P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein. Time series anomaly discovery with grammar-based compression. In *EDBT*, pages 481–492, 2015.

K. Shafi, T. Kovacs, H. A. Abbass, and W. Zhu. Intrusion detection with evolutionary learning classifier systems. *Natural Computing*, 8(1):3–27, 2009.

F. Shen, Y. Xu, L. Liu, Y. Yang, Z. Huang, and H. T. Shen. Unsupervised deep hashing with similarity-adaptive and discrete optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

J. Shieh and E. Keogh. i sax: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631. ACM, 2008.

M. Shokoohi-Yekta, Y. Chen, B. Campana, B. Hu, J. Zakaria, and E. Keogh. Discovery of meaningful rules in time series. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1085–1094. ACM, 2015.

K. Shvachko, H. Kuang, S. Radia, R. Chansler, et al. The hadoop distributed file system. In *MSST*, volume 10, pages 1–10, 2010.

C. Silva-Santos, P. Goulart, F. Bertelli, A. Garcia, and N. Cheung. An artificial immune system algorithm applied to the solution of an inverse problem in unsteady inward solidification. *Advances in Engineering Software*, 121:178–187, 2018.

B. Skinner, H. Nguyen, and D. Liu. Classification of eeg signals using a genetic-based machine learning classifier. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 3120–3123. IEEE, 2007.

T. Stibor, P. Mohr, J. Timmis, and C. Eckert. Is negative selection appropriate for anomaly detection? In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 321–328. ACM, 2005.

C. Stone and L. Bull. For real! xcs with continuous-valued inputs. *Evolutionary Computation*, 11(3): 299–336, 2003.

J. Strackeljan and K. Leiviskä. Artificial immune system approach for the fault detection in rotating machinery. *Proceedings of CM*, pages 1365–1375, 2008.

R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

H. Tan. A brief history and technical review of the expert system research. In *IOP Conference Series: Materials Science and Engineering*, volume 242, page 012111. IOP Publishing, 2017.

M. Tan, L. Wang, and I. W. Tsang. Learning sparse svm for feature selection on very high dimensional datasets. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 1047–1054. Citeseer, 2010.

P. Thiam, S. Meudt, G. Palm, and F. Schwenker. A temporal dependency based multi-modal active learning approach for audiovisual event detection. *Neural Processing Letters*, pages 1–24, 2017.

A. L. Thomaz, C. Breazeal, et al. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Aaai*, volume 6, pages 1000–1005. Boston, MA, 2006.

M. Thottan and C. Ji. Anomaly detection in ip networks. *IEEE Transactions on signal processing*, 51(8): 2191–2204, 2003.

J. Timmis, P. Andrews, and E. Hart. On artificial immune systems and swarm intelligence. *Swarm Intelligence*, 4(4):247–273, 2010.

S. Tong and E. Chang. Support vector machine active learning for image retrieval. In *Proceedings of the ninth ACM international conference on Multimedia*, pages 107–118. ACM, 2001.

S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

Y. Trachi, E. Elbouchikhi, V. Choqueuse, T. Wang, and M. Benbouzid. Induction machine faults detection based on a constant false alarm rate detector. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 6359–6363. IEEE, 2016.

J. Twycross and U. Aickelin. Towards a conceptual framework for innate immunity. In *International Conference on Artificial Immune Systems*, pages 112–125. Springer, 2005.

J. Twycross and U. Aickelin. Biological inspiration for artificial immune systems. In *Artificial immune systems*, pages 300–311. Springer, 2007.

H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

P. A. Vargas, L. N. de Castro, and F. J. Von Zuben. Mapping artificial immune systems into learning classifier systems. In *International Workshop on Learning Classifier Systems*, pages 163–186. Springer, 2002.

K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li. Ai 2: training a big data machine to defend. In *Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016 IEEE 2nd International Conference on*, pages 49–54. IEEE, 2016.

J. Vlissides, R. Helm, R. Johnson, and E. Gamma. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 49(120):11, 1995.

N. Walkinshaw, S. Afshan, and P. McMinn. Using compression algorithms to support the comprehension of program traces. In *Proceedings of the Eighth International Workshop on Dynamic Analysis*, pages 8–13. ACM, 2010.

K. Wang, D. Zhang, Y. Li, R. Zhang, and L. Lin. Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, 2017a.

X. Wang, J. Lin, P. Senin, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, and S. Frankenstein. Rpm: Representative pattern mining for efficient time series classification. In *EDBT*, pages 185–196, 2016a.

Z. Wang, J. Yang, H. Zhang, C. Li, S. Zhang, and H. Wang. Towards online anomaly detection by combining multiple detection methods and storm. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 804–807. IEEE, 2016b.

Z. Wang, W. Yan, and T. Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017b.

J. S. Ward and A. Barker. Observing the clouds: a survey and taxonomy of cloud monitoring. *Journal of Cloud Computing*, 3(1):24, 2014.

A. Watkins and J. Timmis. Exploiting parallelism inherent in airs, an artificial immune classifier. In *International Conference on Artificial Immune Systems*, pages 427–438. Springer, 2004.

A. Watkins, J. Timmis, and L. Boggess. Artificial immune recognition system (airs): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, 5(3):291–317, 2004.

L. Wei, N. Kumar, V. N. Lolla, E. J. Keogh, S. Lonardi, and C. A. Ratanamahatana. Assumption-free anomaly detection in time series. In *SSDBM*, volume 5, pages 237–242, 2005.

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78 (10):1550–1560, 1990.

D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286, 2000.

S. W. Wilson. Zcs: A zeroth level classifier system. *Evolutionary computation*, 2(1):1–18, 1994.

S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary computation*, 3(2):149–175, 1995.

W. Wilson, P. Birkin, and U. Aickelin. The motif tracking algorithm. *International Journal of Automation and Computing*, 5(1):32–44, 2008.

C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh. Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 1317–1322. IEEE, 2016.

Y. Yuan, D. Lin, A. Mishra, S. Marwaha, R. Alur, and B. T. Loo. Quantitative network monitoring with netqre. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 99–112. ACM, 2017.

A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou. Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

T. Zhang and M. Nakamura. Neural network-based hybrid human-in-the-loop control for meal assistance orthosis. *IEEE transactions on neural systems and rehabilitation engineering*, 14(1):64–75, 2006.

Y. Zhang and V. Paxson. Detecting stepping stones. In *USENIX Security Symposium*, volume 171, page 184, 2000.

B. Zhou, J. Li, X. Wang, Y. Gu, L. Xu, Y. Hu, and L. Zhu. Online internet traffic monitoring system using spark streaming. *Big Data Mining and Analytics*, 1(1):47–56, 2018.

F. Zhou, F. De la Torre, and J. K. Hodgins. Aligned cluster analysis for temporal segmentation of human motion. In *Automatic Face & Gesture Recognition, 2008. FG'08. 8th IEEE International Conference on*, pages 1–7. IEEE, 2008.

G.-B. Zhou, J. Wu, C.-L. Zhang, and Z.-H. Zhou. Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3):226–234, 2016.

F. Zhu, W. Chen, H. Yang, T. Li, T. Yang, and F. Zhang. A quick negative selection algorithm for one-class classification in big data era. *Mathematical Problems in Engineering*, 2017, 2017.

J. Zhu, H. Wang, T. Yao, and B. K. Tsou. Active learning with sampling by uncertainty and density for word sense disambiguation and text classification. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 1137–1144. Association for Computational Linguistics, 2008.

X. Zhu. Semi-supervised learning. In *Encyclopedia of machine learning*, pages 892–897. Springer, 2011.

X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, volume 3, 2003.

Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh. Exploiting a novel algorithm and gpus to break the ten quadrillion pairwise comparisons barrier for time series motifs and joins. *Knowledge and Information Systems*, 54(1):203–236, 2018.

Annexe : Résumé

Introduction

Émergence et supervision du Cloud

Nous vivons dans l'ère du Cloud. Nos données et nos applications quittent progressivement l'ordinateur personnel des années 1990 et 2000 pour s'établir dans les Clouds publics. Les exemples de telles migrations abondent, y compris dans le monde professionnel : en 2018, Office 365, la version en ligne de la suite bureautique de Microsoft, atteignait 56% de parts de marché, dépassant la version native. Le nouveau modèle de développement d'applications en mode SaaS[1], avec des données stockées dans les Clouds publics plutôt que des disques locaux, implique un nouveau paradigme en termes de connectivité et d'accès au réseau. Les utilisateurs et les entreprises sont à présent connectés 24 heures par jour, et une panne d'accès Internet peut signifier le chômage technique instantané. Le succès du modèle Cloud a permis l'émergence d'une révolution mobile grâce à l'explosion des services hébergés.

Si la notion d'applications et de données "dans le Cloud" peut encore paraître confuse à certains [Dillon et al., 2010], le terme de *Cloud computing* a été précisément défini par le NIST (National Institute of Standards and Technology, bureau des standards américain) : "Le *Cloud computing* est un modèle permettant un accès réseau universel, pratique, à la demande, à un ensemble de ressources configurables (réseaux, serveurs, stockage, applications et services) pouvant être provisionnées et déprovisionnées rapidement, avec un effort minimal d'administration et d'interaction avec le fournisseur de service" [Mell et al., 2009]. Le même document définit les trois niveaux du *Cloud computing :*

- IaaS, Infrastructure-as-a-Service : déploiement à la demande d'infrastructure virtuelle (segments réseau et machines virtuelles),
- PaaS, Platform-as-a-Service : déploiement à la demande d'environnements d'exécution pour des applications,
- SaaS, Software-as-a-Service : accès à la demande à des services et des applications, en général à travers un navigateur web.

En d'autres termes, l'IaaS fournit à ses utilisateurs des machines et des réseaux virtuels, le PaaS des serveurs d'application et le SaaS des applications. Ces trois modèles nécessitent un accès par le réseau et, par leur nature même, génèrent une charge conséquente sur l'infrastructure Internet mondiale, devenue plus critique que jamais. Cette nouvelle criticité renforce le besoin de supervision réseau afin de détecter toute éventuelle panne, congestion, surcharge ou attaque dès ses premiers signes [Alhamazani et al., 2015, Mdini et al., 2017, Ward and Barker, 2014].

La supervision est le processus par lequel les opérateurs réseau et fournisseurs d'accès Internet sont informés en temps réel de tout problème survenant dans leur réseau ou les équipements qui le

[1] Software-as-a-Service, la nouvelle vague d'applications Web fournissant des fonctionnalités équivalentes aux applications de bureau et fonctionnant par abonnement mensuel.

composent, voire les applications qu'ils hébergent. Il s'agit d'un élément crucial dans l'opération de toute infrastructure informatique de taille non triviale. Toutefois, malgré l'importance de la supervision, particulièrement dans le contexte de la révolution du Cloud, les solutions mises en place par les entreprises et les hébergeurs sont encore rudimentaires ; ce domaine soulève encore des défis, en partie adressés par une large communauté de chercheurs. Avant de nous pencher sur ces défis et sur les solutions proposées, définissons précisément en quoi consiste la supervision.

La supervision de réseau peut être réalisée par capture et analyse de trafic [Zhou et al., 2018] ou des métadonnées associées [Yuan et al., 2017] ou par requêtage actif des équipements [Alhamazani et al., 2015, Gupta et al., 2016]. Si la première méthode est plus avancée, la seconde domine encore dans le monde des entreprises, avec des logiciels tels que Nagios, Cacti, PRTG ou Zabbix, certains libres, d'autres commerciaux, et tous basés sur la supervision active, c'est-à-dire sur le requêtage des équipements réseau, généralement en SNMP. À l'inverse, la supervision passive consiste à laisser les équipements signaler les incidents qu'ils rencontrent ou à capturer le trafic généré. Ces différents types de supervision servent des objectifs différents : la capture de trafic permet de détecter des attaques ou des tentatives d'intrusion [Gupta et al., 2016]. La supervision passive en général est adaptée aux serveurs capables d'exécuter des agents locaux [Anand, 2012]. La supervision de la plupart des équipements réseau reste dominée par le *polling* actif [Chowdhury et al., 2014].

Pour les founisseurs d'accès Internet et les hébergeurs IaaS, la supervision active des équipements réseau, généralement appelée simplement “supervision réseau”, est la plus critique. Ce domaine pose un certain nombre de défis, en particulier pour les raisons suivantes [Ward and Barker, 2014] :

- La plupart des équipements dans un réseau de FAI[2] ne supportent que la supervision active.
- Alors que les applications visant à la sécurité du réseau sont connues pour leur coût calculatoire important, la supervision de réseau est considérée comme scalable sur du matériel générique et très peu coûteuse en temps de calcul.
- Certains outils commerciaux dans le domaine de la sécurité appliquent les méthodes issues de l'apprentissage automatique et de l'intelligence artificielle sur des captures de trafic[3]. Les mêmes mécanismes n'ont pas été développés pour les séries temporelles obtenues par la supervision active.

Ce dernier point – l'application d'analyses de haut niveau sur les données de supervision à l'échelle d'un réseau de FAI complet – est le sujet auquel cette thèse est consacrée.

Défis posés par la supervision à l'échelle du Cloud

Des articles récents citent un certain nombre de défis associés à la supervision réseau [Alhamazani et al., 2015, Mdini et al., 2017, Ward and Barker, 2014, Celenk et al., 2008] :

- Application d'algorithmes d'apprentissage sur les données de supervision,
- Supervision prédictive,
- Détection d'anomalies,
- Scalabilité des analyses.

Essentiellement, le principal obstacle rencontré par la supervision réseau est la nécessité d'analyser en temps réel un volume important de données, qui rend toute analyse non triviale extrêmement difficile.

[2] Fournisseur d'Accès Internet

[3] Voir par exemple Darktrace, un détecteur d'intrusion basé sur une modélisation bayesienne du trafic réseau (https://www.darktrace.com)

Les solutions de supervision généralement déployées ne peuvent que comparer la dernière valeur d'une métrique particulière à un seuil statique défini manuellement. La supervision prédictive, c'est-à-dire la prédiction de valeurs [Alhamazani et al., 2015], la détection d'anomalies [Mdini et al., 2017] et l'apprentissage [Ward and Barker, 2014], sont toujours considérés comme des défis importants par la communauté en raison des coûts calculatoires élevés, pouvant aller jusqu'à nécessiter une infrastructure dédiée, qui lui sont associés [Zhou et al., 2018, Gupta et al., 2016]. Les analyses devant avoir lieu à l'échelle de centaines de milliers, voire de millions de métriques, échantillonnées toutes les cinq minutes[4], même pour un FAI de taille modeste, la puissance de calcul totale pouvant être allouée à *une* série temporelle est extrêmement limitée.

Le premier défi, l'utilisation d'algorithmes d'apprentissage, est lié à la problématique de scalabilité : les modèles d'apprentissage légers sont rares, et les modèles typiquement utilisés pour l'analyse de séries temporelles, comme les réseaux de neurones récurrents [Greff et al., 2017], sont connus pour leurs coûts calculatoires extrêmes nécessitant du matériel spécialisé. De même, la supervision prédictive devient rapidement coûteuse lorsque les dépendances dans le temps sont longues (c'est-à-dire qu'une valeur est fortement corrélée à une autre valeur située plusieurs centaines, voire plusieurs milliers de points dans le passé).

Le défi posé par la scalabilité des analyses implique qu'un système de supervision capable d'analyser des millions de séries en temps réel est fondamentalement incompatible avec la direction actuelle prise par les modèles d'apprentissage, en particulier les modèles d'apprentissage profond, qui nécessitent des clusters de calcul pour l'entraînement et dont l'exécution, si elle peut avoir lieu sur du matériel conventionnel, demeure coûteuse.

Propriétés des systèmes de supervision

En dehors de toute problématique de recherche, un système de supervision doit vérifier un certain nombre de propriétés pour être considéré comme fonctionnel. Un système ne vérifiant pas ces propriétés, quel que soit le niveau de sophistication des modèles mis en œuvre et la précision de ses résultats, devra être considéré comme essentiellement inutile. Ces propriétés ne sont pas nécessairement des défis en elles-mêmes ; en revanche, elles doivent influencer les solutions répondant aux problématiques de recherche soulevées par la supervision.

La première propriété est l'analyse en temps réel [Elsen et al., 2015, Zhou et al., 2018, Lin et al., 2016], qui ne se limite pas à la supervision *réseau :* cette même exigence se retrouve par exemple dans le monde de la sécurité [Zhang and Paxson, 2000] pour la détection d'intrusion. Le temps réel implique l'utilisation d'algorithmes *en ligne* [Wang et al., 2016b] : les valeurs échantillonnées doivent être traitées au fur et à mesure de leur acquisition, en flux, plutôt qu'au repos lorsque toutes les données sont disponibles[Gupta et al., 2016]. L'opération en ligne limite les algorithmes pouvant être utilisés pour l'analyse, car tous les modèles ne la supportent pas [Zhou et al., 2018].

La seconde propriété recherchée est l'autonomie, c'est-à-dire la capacité à opérer avec un contrôle humain minimal [Liotta et al., 2002]. De toute évidence, un système supervisant des millions de séries temporelles ne peut pas bénéficier d'un réglage fin pour chaque série : un seuil de sensibilité global peut être pertinent, mais pas davantage. Encore une fois, cette contrainte élimine un certain nombre de modèles nécessitant l'ajustement de paramètres multiples pour obtenir des résultats pertinents [Keogh et al., 2004].

La troisième propriété requise est un taux de faux positifs bas [Dastjerdi et al., 2012, Zhang and Paxson, 2000]. Les fausses alertes génèrent un surcroît de travail aux opérateurs du réseau et augmentent la probabilité qu'une véritable alerte soit prise pour un faux positif et traitée trop tard. Le but d'une

[4] Les paramètres par défault de Nagios stipulent qu'une métrique est mesurée toutes les cinq minutes et, si un incident est détecté, toutes les minutes.

supervision “avancée”, c’est-à-dire mettant en œuvre des analyses plus poussées que de simples seuils d’alerte, est d’obtenir une meilleure détection des incidents tout en maintenant un taux de fausses alertes limité [Wang et al., 2016b].

Ces propriétés sont de nature fonctionnelle : elle s’appliquent à tout système de supervision, qu’il soit commercial ou issu de la recherche académique. Toutefois, les défis listés plus haut (utilisation d’algorithmes d’apprentissage, détection d’anomalies, supervision prédictive et scalabilité) donnent lieu à un nouvel ensemble de propriétés pour un système de supervision *avancé*. Si l’apprentissage peut être vu comme un outil plutôt qu’un prérequis, les trois propriétés suivantes demeurent :

- *Détection d’anomalies :* une métrique changean brutalement de comportement peut constituer un meilleur indicateur d’un incident en cours qu’une valeur dépassant un seuil [Ward and Barker, 2014].
- *Supervision prédictive :* avertir les opérateurs du réseau dès les premiers signes d’une anomalie, avant qu’elle ne constitue un danger, peut prévenir des incidents et éviter leurs impacts éventuels [Celenk et al., 2008].
- *Scalabilité :* l’analyse en temps réel de millions de métriques, même sur des systèmes distribués, implique des algorithmes adaptés [Gupta et al., 2016].

En résumé, les propriétés que doit valider un système de supervision sont :

- Faible coût calculatoire, ou scalabilité,
- Analyse en temps réel,
- Détection d’anomalies,
- Supervision prédictive,
- Faible coût de configuration, ou autonomie,
- Faible taux de faux positifs.

Supervision et immunité

Le point de départ de cette thèse est qu’un sous-système en temps réel, distribué, scalable et capable d’autorégulation, destiné à superviser le système dans lequel il évolue, réagir à d’éventuelles anomalies ainsi qu’à des menaces connues aussi rapidement que possible, tout ceci en maintenant un taux de faux positifs acceptablement bas, existe déjà dans la nature. De fait, cette description correspond fidèlement au système immunitaire tel qu’il existe chez l’humain et beaucoup d’autres organismes multicellulaires.

Depuis ses premiers pas, l’informatique a produit des modèles inspirés de processus naturels pour résoudre des problèmes complexes – modèles qui ont à leur tour engendré des centaines de variations. Les années 1950 avaient déjà introduit la brique de base des réseaux de neurones profonds tels que nous les connaissons avec le *perceptron artificiel* [Rosenblatt, 1958]. Les algorithmes évolutionnaires, eux aussi, ont été introduits à cette époque [Minsky, 1958]. Au fil des années, de plus en plus de phénomènes ont été copiés, simulés et transformés en algorithmes et en métaheuristiques : les nuées d’oiseaux [Reynolds, 1987], la manière dont les fourmis découvrent le plus court chemin vers une source de nourriture [Dorigo and Gambardella, 1997] ou encore la danse des abeilles [Pham and Castellani, 2009]. Parmi ces modèles, les systèmes immunitaires artificiels (SIA) ont été construits à partir des découvertes des immunologistes telles que le modèle de réseau immunitaire [Jerne, 1974] ou la maturation des lymphocytes T dans le thymus [Forrest et al., 1994].

De même que beaucoup d’autres métaheuristiques inspirées par des phénomènes naturels, les systèmes immunitaires artificiels comportent beaucoup de modèles : tout algorithme construit autour de la réaction entre anticorps et antigènes, la maturation des lymphocytes T ou B, par suppression ou par hypermutation, ou n'importe quel autre processus immunitaire, peut prétendre à l'appellation de SIA [Hart and Timmis, 2008]. Cette variété rend le terme lui-même générique.

Même si la communauté des systèmes immunitaires artificiels n'est jamais parvenue à créer une famille d'algorithmes unifiée, à la manière par exemple des réseaux de neurones artificiels qui, s'ils peuvent varier en géométrie et en activation – classiques ou profonds, à propagation avant ou récurrents, à neurones linéaires ou sigmoïdes... –, elle a en revanche établi un certain nombre de ***propriétés*** désirables partagées entre tous les SIA. Ces propriétés correspondent largement à celles des systèmes immunitaires biologiques. Dans [Aickelin and Cayzer, 2008], elles sont résumées comme suit :

- *Tolérance aux erreurs :* une défaillance unique ne doit pas impacter le comportement du système à long terme.
- *Distribution :* tout comme les lymphocytes sont distribués dans tout le corps par l'intermédiaire du système lymphatique, un SIA doit pouvoir être distribué entre un nombre arbitraire de nœuds de calcul.
- *Adaptation :* des changements dans le système "immunisé" doivent être appris par le système immunitaire plutôt que déclencher une réaction auto-immune.
- *Auto-supervision :* le système immunitaire doit être capable de détecter des anomalies dans son propre comportement et de les corriger.

Dans [Twycross and Aickelin, 2005], les auteurs formalisent ces propriétés dans le modèle ODISS, initiales de :

- *Ouverture :* le système immunitaire et son hôte interagissent dans un état d'équilibre dynamique.
- *Diversité :* un vaste répertoire de cellules immunitaires couvre un large ensemble d'antigènes possibles.
- *Interaction :* le système immunitaire peut être modélisé comme un réseau de cellules interagissant par échanges de signaux.
- *Structure :* le système immunitaire est fonctionnellement découpé en sous-systèmes remplissants différents rôles.
- *Scalabilité :* le système immunitaire est composé d'une large population d'agents simples plutôt que d'une petite population d'agents complexes.

Dans un article plus récent [Twycross and Aickelin, 2007], les mêmes auteurs décrivent un ensemble de propriétés légèrement différent :

- *Spécificité :* un anticorps ciblant un antigène particulier n'est sensible qu'à cet *exact* antigène.
- *Diversité :* un vaste répertoire de cellules immunitaires couvre un large éventail de possibles antigènes.
- *Mémoire :* des antigènes spécifiques, après une première exposition, laissent une trace dans le système immunitaire et déclenchent une réponse plus rapide et plus intense lors d'une exposition secondaire.
- *Tolérance :* le système immunitaire, tout en restant capable de réagir à presque tout type d'antigène, ne réagit pas contre le corps lui-même.

Dans [Greensmith et al., 2010], Greensmith propose la cellule dendritique comme base pour les systèmes immunitaires artificiels, citant les propriétés suivantes :

- *Décentralisation :* le système immunitaire n'a pas de point central de décision ou de coordination : les cellules qui le composent interagissent localement.
- *Robustesse/tolérance aux erreurs :* des erreurs commises par un seul individu, c'est-à-dire une seule cellule du système immunitaire, ne se propagent pas au reste du système et n'influent pas sur son comportement global.

Bien entendu, les processus formant le cœur du système immunitaire, tels que la détection d'anomalies – parfois formulée comme une distinction entre le *soi* et le *non-soi* – ou la mémoire, ne sont pas toujours cités comme des *propriétés* mais comme *cas d'usage* pour les SIA, ou comme la fonction principale du système immunitaire. De la même manière que nous avons pu extraire les propriétés essentielles des systèmes de supervision, nous pouvons proposer les propriétés suivantes pour les SIA :

- *Décentralisation/scalabilité :* le système immunitaire, du fait qu'il fonctionne par interactions locales, peut passer à l'échelle presque indéfiniment, horizontalement et sans point individuel de défaillance.
- *Détection d'anomalies :* des objets inconnus, qu'ils soient modélisés par des vecteurs, des antigènes ou des motifs dans une chaîne de caractères ou une série temporelle, peuvent déclencher une réaction immunitaire.
- *Tolérance :* des objets qui devraient en d'autres circonstances déclencher une réaction immunitaire peuvent être tolérés : la réaction immunitaire peut être supprimée.
- *Adaptation :* lorsque l'hôte du système immunitaire change, le système est capable de supprimer la réaction auto-immune globale que ce changement pourrait déclencher. En d'autre termes, l'adaptation est identique à la tolérance appliquée *dans le temps*.
- *Mémoire :* des pathogènes ne causant normalement pas de réaction immunitaire peuvent être découverts et une réaction immunitaire peut être forcée. Cette propriété est l'inverse de la tolérance ; elle est également la base de fonctionnement de la vaccination et de la réaction plus rapide et plus intense lors d'une exposition secondaire à un pathogène connu.

Un point rarement cité dans la littérature est que, pour simuler un système immunitaire biologique, ces propriétés adaptées en SIA doivent être appliquées en temps réel : l'adaptation, par exemple, n'a pas de sens si l'intégralité de l'historique de l'hôte est connue en avance de phase. De même, la mémoire doit évoluer au fil du cycle de vie de l'hôte : il ne s'agit pas simplement d'un ensemble de détecteurs appris durant une hypothétique phase d'apprentissage.

La communauté des SIA est unique dans le sens qu'elle est extrêmement proche des immunologistes : suivant l'exemple de Jerne [Jerne, 1974], des chercheurs comme Varela et Bersini [Bersini and Varela, 1990, Bersini, 2002, Cutello et al., 2004] ont exploré les deux aspects de ce domaine, d'un côté les modèles issus de l'informatique et de l'autre ceux utilisés en biologie. Leurs recherches ont contribué non seulement au développement des systèmes immunitaires artificiels, mais également à la compréhension des systèmes immunitaires naturels. Les travaux de Greensmith [Greensmith and Aickelin, 2009] sur les cellules dentritiques constituent autant une proposition d'algorithme qu'un modèle réaliste d'un processus biologique. De même, la théorie du danger développée par Matzinger [Matzinger, 1994, 2002], un immunologiste, a été étudiée du point de vue des SIA par Aickelin [Aickelin and Cayzer, 2008] rapidement après son introduction.

Malheureusement, cette communauté tend à adopter une vision pessimiste de son champ d'étude, soulignant le décalage entre les problèmes réels auxquels les SIA peuvent être appliqués [Freitas and Timmis, 2003, 2007] et les modèles parfois naïfs du système immunitaire utilisés les traduire [Matzinger, 2002, Bersini, 2002]. Les questions vitales pour la communauté sont : les SIA peuvent-ils réellement vérifier les propriétés des systèmes immunitaires biologiques ? Et peuvent-ils passer à l'échelle sur des problèmes réels comme le font les réseaux de neurones ou les algorithmes évolutionnaires ?

Une vision pragmatique du problème, introduite dans deux articles de Freitas et Timmis [Freitas and Timmis, 2003, 2007], propose d'orienter la conception de nouveaux modèles vers les *propriétés* que les SIA tentent d'atteindre plutôt que sur les *processus* biologiques dont ils s'inspirent. Le but est de rapprocher les modèles des problèmes qu'ils cherchent à résoudre, avec des contraintes et des évaluations issues du domaine d'application.

L'un des principes démontrés au fil des années par la communauté des SIA est que différentes facettes du processus biologique d'immunité doivent être modélisées par différents algorithmes [de Castro and Von Zuben, 2001, Forrest et al., 1997, De Castro and Von Zuben, 2002, Aickelin and Cayzer, 2008]. Ces modèles sont discutés plus en détail à la section 4.1. La conclusion logique est que, pour vérifier un ensemble de propriétés tirées des systèmes immunitaires, plusieurs algorithmes doivent fonctionner ensemble. En ce sens, plutôt que des *algorithmes*, les SIA devraient être vus comme des méta-algorithmes intégrant un certain nombre de modèles pour simuler, par exemple, les systèmes immunitaires inné et acquis, et les interactions entre les deux [Greensmith and Aickelin, 2009].

Nous introduisons dans cette thèse le terme d'*Écosystème Immunitaire Artificiel* (EIA), considérant qu'un ensemble de systèmes en interaction constitue un écosystème. Cette approche est développée en détail au chapitre 6. Essentiellement, nous démontrons que, une fois admise la prémisse que différents composants – systèmes ou algorithmes – sont nécessaires pour valider toutes les propriétés des systèmes immunitaires, les SIA des années 1990 et 2000 apparaissent comme des outils imparfaits, et qu'un ensemble de modèles spécialisés est préférable, à la fois en termes de performances – précision ou coût calculatoire – et de compréhension de l'écosystème résultant. Ainsi, l'EIA constitue une structure dans laquelle des algorithmes adaptés au problème peuvent être insérés ; son rôle est de spécifier les interactions entre ces algorithmes afin d'assurer les propriétés recherchées.

L'argument central de cette thèse est que les propriétés des systèmes immunitaires sont compatibles avec celles des systèmes de supervision, et qu'un modèle vérifiant les deux ensembles de propriétés peut constituer à la fois un système de supervision efficace et un SIA valide. Le cas de test de l'EIA est la supervision du réseau d'un opérateur Internet et Cloud, qui peut bénéficier des propriétés particulières des SIA. Cette application consiste à superviser en temps réel des centaines de milliers de séries temporelles *(scalabilité)* afin de détecter de potentielles pannes *(détection d'anomalies)*, éviter les fausses alertes *(tolérance)*, enregistrer les modes de défaillance connus afin de les détecter plus rapidement *(mémoire)* et alerter les équipes de support lorsqu'une panne est relevée.

Ces propriétés peuvent être assurées par différents algorithmes ou ensembles d'algorithmes. L'EIA ne définit pas directement quel algorithme doit être utilisé pour quelle fonction, seulement comment les modèles remplissant ces fonctions interagissent. Le choix des algorithmes dépend du domaine d'application et du type de données considéré. Dans cette thèse, il sera guidé par les propriétés spécifiques des systèmes de supervision :

- Coût calculatoire et mémoriel limité,
- Analyse en temps réel,
- Faible coût de configuration,
- Faible taux de faux positifs.

Rappelons à ce sujet les sages paroles d'Andrew Watkins et Jon Timmis [Watkins and Timmis, 2004] concernant leur Système de Reconnaissance Immunitaire Artificielle (AIRS, Artificial Immune Recognition System) et applicables à tout modèle bio-inspiré : *"**Inspiré par l'immunité est le mot-clef.** Nous ne prétendons pas que l'AIRS modélise directement un quelconque processus immunitaire, mais plutôt que l'AIRS emploie certains composants qui peuvent être métaphoriquement reliés à des composants du système immunitaire."*

De même, nous ne prétendons pas que l'EIA est un modèle biologiquement plausible. Il est toutefois plausible dans le monde des SIA, basé sur les mêmes métaphores et vérifiant les mêmes propriétés.

Contribution

La ressemblance frappante entre les systèmes immunitaires et de supervision, en termes de finalité et de propriétés – ou de propriétés *souhaitées* – constitue le point de départ de cette thèse. Notre hypothèse est que l'ajout des briques algorithmiques nécessaires pour introduire les propriétés spécifiques des SIA dans un système de supervision peut adresser au moins une partie des problématiques soulevées par la communauté. Nous posons les questions suivantes : une conception inspirée des systèmes immunitaires peut-elle résoudre certains des problèmes de systèmes de supervision ? En particulier, peut-elle apporter un niveau de précision, de prédiction des pannes et d'autonomie compétitif avec les approches de l'état de l'art tout en maintenant un coût calculatoire et mémoriel limité ?

Ces questions doivent être lues dans le contexte des propriétés souhaitées pour les systèmes de supervision :

- *Précision :* mesurée par le taux de faux positifs et les nouveaux modes de défaillance pouvant être détectés par rapport à une approche basée sur des seuils d'alertes.
- *Prédiction de pannes :* définie comme la durée entre la détection d'un incident et le moment où il aurait été détecté par franchissement de seuil.
- *Autonomie :* capacité à opérer sans contrôle humain, et en particulier sans paramétrage fin.

Nous montrons au chapitre 2 que ces problématiques correspondent aux axes de recherche de la communauté de la supervision. Les communautés des statistiques et de l'apprentissage fournissent des modèles qui peuvent constituer des solutions appropriées à ces problèmes, quoique les exemples d'applications et d'évaluations manquent. Le modèle immunitaire apporte également une notion de *mémoire* qui, au mieux de notre connaissance, peut être modélisée par de nombreux algorithmes mais n'a jamais été formalisée en tant que telle. De même, la notion de *tolérance* n'a pas été explorée en détail par la communauté de l'apprentissage.

Notre objectif est d'adapter ou de concevoir les algorithmes adaptés pour modéliser les propriétés immunitaires décrites à la section précédente. Nous espérons voir le système résultant vérifier les propriétés des systèmes de supervision mieux que les méthodes traditionnelles, principalement à cause des ressources très limitées disponibles pour chaque série temporelle analysée. Cette entreprise soulève un certain nombre de questions qui n'ont pas été totalement résolues dans la littérature :

- La détection d'anomalies dans des séries temporelles est-elle réalisable à l'échelle de la supervision d'un réseau en temps réel avec un budget calculatoire réduit ?
- Une forme de mémoire immunitaire peut-elle fournir une meilleure prédiction de pannes et, de la même manière que dans les systèmes immunitaires biologiques, accélérer la réaction à une défaillance connue ?
- Comment un système conçu pour détecter des anomalies peut-il être rendu tolérant à certains comportements rares, de la même manière que le système immunitaire tolère les bactéries composant la flore intestinale ?

L'objectif de l'EIA est de répondre à ces questions : il sera mis à l'épreuve sur ces points précis. Notons que ceux-ci reflètent les propriétés clefs des systèmes immunitaires telles que décrites précédemment ; ces propriétés seront la cible des évaluations menées à bien sur l'EIA.

Cette thèse apporte quatre contributions à l'état de l'art, qui sont reprises à la section 11.1 du dernier chapitre. Elles couvrent les domaines suivants :

- *Processus et architecture pour la supervision :* le modèle *Cloud-Monitor* ajoute un niveau de détection d'anomalies et de prédiction de panne aux systèmes de supervision traditionnels.
- *Apprentissage actif :* le modèle *ALPAGA* utilise la sortie d'un classifieur comme heuristique pour le choix d'échantillons à étiqueter par un humain, permettant en particulier la réduction des faux positifs.
- *Détection d'anomalies :* le modèle *SCHEDA* fournit une approximation du score d'anomalie euclidien pour des séries temporelles périodiques à un coût calculatoire extrêmement réduit. La fonction obtenue fournit une meilleure précision que le score euclidien lui-même.
- *Modélisation immunitaire :* l'*EIA* lui-même reprend l'ensemble des propriétés des SIA dans un méta-algorithme très différent des SIA classiques.

SCHEDA permet le passage à l'échelle de Cloud-Monitor, tandis qu'ALPAGA filtre ses faux positifs sur la base des retours d'un expert. Le système construit grâce à l'interaction de ces trois blocs forme l'EIA.

Cette thèse est divisée en deux parties. La première examine la littérature actuelle afin de déterminer les modèles les plus à même de fournir les constituants d'un EIA. Elle identifie également les limitations de ces modèles et les questions nécessitant la conception de nouveaux algorithmes. Cette partie est organisée comme suit :

Le chapitre 2 examine le domaine de la supervision de réseaux. Il présente les différentes communautés impliquées dans la supervision et la recherche associée à chaque domaine spécifique : la sécurité des réseaux, les machines virtuelles et la supervision des niveaux de service. Cet examen élargit les questions déjà évoquées, en particulier les analyses avancées à grande échelle, et présente les progrès actuels dans la résolution de ces questions. La littérature montre une tendance à intégrer des outils Big Data comme Apache Spark Streaming [5] dans les nouveaux outils de supervision; toutefois, ces techniques et outils ne sont pas encore appliqués aux séries temporelles obtenues par la supervision *active*. Ils sont plutôt utilisés sur les métadonnées de flux, qui représentent des instantanés du système à un moment donné, sans contexte. L'apprentissage automatique reste également largement inutilisé.

Le chapitre 3 couvre les modèles de détection d'anomalies pour les séries temporelles. En particulier, il met l'accent sur les données acquises en flux, les algorithmes en ligne et la scalabilité. La littérature est extrêmement riche et couvre une grande variété de méthodes, mais montre également un certain nombre de limites. Les problèmes posés par les algorithmes scalables qui peuvent fonctionner en ligne, en temps réel, sur le type de séries temporelles obtenues par les systèmes de supervision, ne sont résolus de manière satisfaisante par aucun modèle actuel. En particulier, les dépendances à long terme observées dans les séries temporelles de supervision rendent les modèles auto-régressifs et les réseaux de neurones récurrents largement inopérants. À partir de cette étude, nous concluons qu'un nouvel algorithme est nécessaire pour adresser les caractéristiques spécifiques des séries temporelles obtenues en supervision de réseau, à savoir les motifs longs qu'ils présentent à différentes échelles et le risque d'anomalies à l'une ou l'autre de ces échelles.

Le chapitre 4 relie les domaines de la supervision et des systèmes immunitaires artificiels. Nous avançons que les exigences relatives à la prédiction de pannes dans la supervision et la mémoire dans les systèmes immunitaires sont des problèmes largement identiques : une défaillance d'un système peut être prédite soit en détectant une anomalie dans un paramètre de ce système, soit en remarquant que l'indicateur considéré montre un comportement qui a, par le passé, conduit à une défaillance; les deux approches sont connues sous le nom de *supervision des symptômes de défaillance* [Salfner et al., 2010]. La deuxième définition, cependant, est extrêmement similaire à la notion de mémoire dans les systèmes immunitaires. Dans ce chapitre, nous passons en revue les modèles qui peuvent être

[5] `https://spark.apache.org`

appliqués au problème de la création d'un répertoire immunitaire. En particulier, la littérature montre que les classifieurs basés sur des *instances* (IBC, Instance-Based Classifier), c'est-à-dire des algorithmes entièrement axés sur les données, ne construisant pas de modèles, sont les plus adaptés à une telle approche. Les plus efficaces d'entre eux sont les systèmes de plus proches voisins, qui sont plus efficaces et plus précis que les systèmes immunitaires artificiels traditionnels ou les systèmes basés sur des règles tels que les systèmes de classeurs (LCS, Learning Classifier Systems).

Un classifieur peut être non seulement supervisé ou non-supervisé, il existe également des familles de modèles semi-supervisés utilisant les données non étiquetées au même titre que les étiquetées pour prédire la classe d'un vecteur de test. La mémoire immunitaire étant le mécanisme permettant à la fois une réaction secondaire rapide et une tolérance aux comportements rares [Matzinger, 1994], le classifieur remplissant cette fonction doit être à la fois capable de reconnaître un symptôme de défaillance (mémoire) et une réaction auto-immune (tolérance). Toutefois, cette fonction nécessite que les réactions erronées (faux positifs) soient identifiées comme telles. Dans le contexte de la supervision réseau, nous proposons que les utilisateurs, c'est-à-dire les opérateurs du réseau et les équipes de support technique, peuvent marquer une alerte comme vraie ou fausse. Le classifieur jouant le rôle de mémoire immunitaire peut ensuite, à partir de ces étiquettes reçues en temps réel, ajuster son comportement. Cette approche d'apprentissage est appelée ***apprentissage actif (active learning)*** [Tong and Chang, 2001, Veeramachaneni et al., 2016]. Le chapitre 5 examine les modèles d'apprentissage actif ainsi que leurs scénarios d'application. La littérature montre cependant que la composante temps réel manque dans beaucoup d'évaluations ; la méthodologie active est uniquement comparée à l'apprentissage supervisé classique en termes de nombre d'étiquettes nécessaires pour atteindre un certain niveau de précision.

Le seconde partie de cette thèse présente le modèle l'EIA proposé. L'EIA est un classifieur immuno-inspiré remplissant les critères des systèmes de supervision. En tant que tel, il adresse certains des problèmes soulevés en première partie, en particulier concernant les systèmes de supervision. Nous proposons également de nouveaux algorithmes pour l'apprentissage actif et la détection d'anomalies dans des séries temporelles, nécessaires dans ce modèle. Ces contributions, de même que l'EIA au global, sont évaluées dans le contexte de la supervision de réseaux et en fonction des propriétés des systèmes immunitaires et de supervision. Cette partie est organisée comme suit :

Le chapitre refchap :global propose une vue de haut niveau de l'écosystème immunitaire artificiel. Il décrit les sous-systèmes individuels qui forment l'écosystème proprement dit et leurs interactions, ainsi que les propriétés de chacun d'entre eux. Il présente également la métaphore immunitaire sur laquelle repose l'EIA, en particulier la manière dont l'EIA modélise les systèmes immunitaires inné et adaptatif en trois processus différents qui correspondent au fonctionnement d'un système de supervision :

- L'immunité innée fournit des mécanismes de détection peu granulaires couvrant un large éventail de pathogènes. Elle correspond à la détection à base de seuils des systèmes de supervision : une connaissance statique, non évolutive, peu spécifique mais capable de réagir à des dysfonctionnements très apparents.
- L'immunité acquise fournit une détection de pathogènes à la fois connus et inconnus à une granularité très fine. Les pathogènes connus sont détectés plus rapidement que les inconnus (réponse immunitaire secondaire). La détection de pathogènes inconnus correspond au mécanisme de détection d'anomalies [Greensmith et al., 2005]. La réponse immunitaire secondaire, toutefois, n'a pas d'équivalent direct dans les systèmes de supervision, quoiqu'elle soit comparable à la base de données de signatures d'un antivirus ou d'un détecteur d'intrusion. Il s'agit de la *mémoire* immunitaire.

Le chapitre 7 décrit un modèle d'EIA simple qui vérifie *certaines* des principales propriétés immunitaires, à savoir la détection d'anomalies et la mémoire. Nous donnons à ce premier modèle le nom de *Cloud-Monitor.* Celui-ci est évalué dans le contexte d'un système de supervision. Nous montrons que sa capacité accrue à détecter à la fois les défaillances d'un type inconnu, c'est-à-dire les nouveaux comportements, et les défaillances *connues*, permet des temps de détection plus rapides. Bien que cette constatation soutienne notre hypothèse initiale selon laquelle les modèles inspirés par le système immunitaire peuvent être la base des systèmes de supervision, nous constatons également que, en l'absence d'une boucle de rétroaction, le taux de faux positifs devient si élevé qu'il menace de rendre le système inutilisable. Ce constat n'est pas surprenant, car Cloud-Monitor ne modélise pas la tolérance immunitaire. Il confirme l'importance de cette propriété.

Pour filtrer ces fausses alertes, un deuxième bloc est ajouté à l'EIA. Il se positionne entre le système de détection immunitaire et l'utilisateur et permet de détecter et de filtrer les fausses alertes. Ainsi, il offre une tolérance immunitaire. Le chapitre 8 propose *ALPAGA* (Actively Learnt Positive Alert GAteway), une méthodologie basée sur le paradigme de l'apprentissage actif pour utiliser les retours des experts afin de distinguer les fausses alarmes des vraies. ALPAGA est basé sur la notion qu'une alarme de supervision peut être étiquetée par les opérateurs du réseau comme correspondant à un incident réel ou à un simple faux positif, et que les associations de fragments de séries temporelles et d'étiquettes peuvent être utilisées pour entraîner un classifieur. Celui-ci peut alors filtrer les alarmes futures et rejeter celles qui sont détectées comme fausses.

Nous évaluons plusieurs familles d'algorithmes pour déterminer les meilleurs candidats au rôle de filtre d'alertes de supervision : réseaux de neurones, filtres bayésiens, arbres de décision, plus proches voisins et SVM. Les alarmes étant un phénomène rare, avec au plus une ou deux alarmes par mois dans la plupart des séries temporelles, ce classifieur doit être en mesure de travailler avec de très petits ensembles de données. Les expériences montrent que dans ces conditions, les classifieurs basés sur des instances, comme les modèles de plus proches voisins, obtiennent les meilleurs résultats en termes de précision et de coût calculatoire. En comparant la précision d'un système de sueprvision générant des alertes sans filtre et filtrées par ALPAGA, nous montrons que les modèles de plus proches voisins et les arbres de décision sont plus performants que le système de supervision seul. Les SVM, réseaux de neurones et classifieurs bayésiens, à l'inverse, ont des performances inférieures ou égales au système de supervision non filtré. Ceci indique que les algorithmes les mieux adaptés pour fournir la tolérance immunitaire sont également les meilleurs modèles de mémoire immunitaire. Ce fait ouvre la possibilité d'utiliser une mémoire *unique* pour la réaction secondaire et la tolérance immunitaire. Ceci non seulement simplifie le modèle, mais correspond également au fonctionnement réel du système immunitaire biologique par répertoires de cellules mémoire (lymphocytes T et B spécialisés).

Un processus crucial dans ce système de supervision immunitaire est la détection d'anomalies. Les séries temporelles obtenues par la supervision montrent un certain nombre de propriétés qui peuvent être observées par simple tracé et ont été étudiées dans d'autres domaines, notemment l'utilisation de puissance électrique [Espinoza et al., 2005, Keogh et al., 2004]. Typiquement, les motifs présents dans ces séries sont multi-échelles, imbriqués et non-linéaires [De Chaves et al., 2011], avec de fortes composantes périodiques, en particulier quotidiennes et hebdomadaires. Comme nous le montrons au chapitre 3, ce type de dépendance met en échec certains modèles d'apprentissage tels que les réseaux de neurones récurrents et rend la modélisation de processus stochastiques, par exemple auto-régressifs, pratiquement impossible.

Le chapitre 9 décrit un ensemble d'heuristiques qui permettent la détection d'anomalies en ligne, en temps réel, sur les séries temporelles issues de la supervision. Nous montrons que ces heuristiques, que nous appelons *SCHEDA (Sampled Causal heuristics for Euclidean Distance Approximation),* sont de bons prédicteurs du score d'anomalie euclidien, mais peuvent être calculés en temps linéaire plutôt que quadratique. Nous montrons que SCHEDA et le score d'anomalie euclidien, qui est une méthode naïve de recherche de nouveauté – naïve au sens que son coût calculatoire la rend essentiellement inutile pour des séries de plus de quelques milliers d'échantillons [Keogh et al., 2005] – donnent des performances au moins similaires sur des données réelles de supervision, avec un léger avantage pour SCHEDA. SCHEDA, cependant, peut passer à l'échelle de centaines de milliers de séries sur un seul serveur.

Ainsi Cloud-Monitor fournit *mémoire* nécessaire à la réaction immunitaire secondaire, ALPAGA y ajoute la *tolérance* et SCHEDA permet la *détection d'anomalies* et la *scalabilité.* Ces algorithmes fonctionnent tous *en ligne* et en *temps réel.* La dernière propriété, l'*autonomie,* est évaluée dans le contexte de l'EIA complet.

Le chapitre 10 décrit et évalue l'EIA complet intégrant le système de détection immunitaire Cloud-Monitor, en utilisant SCHEDA pour la détection d'anomalies, et un classifieur par plus proches voisins partagé entre la base de données de défaillances de Cloud-Monitor (mémoire immunitaire) et le filtre ALPAGA qui élimine les fausses alertes (tolérance immunitaire). Les évaluations sont réalisées sur un système de supervision simple, un système de supervision avec un filtre ALPAGA et Cloud-Monitor seul. Les paramètres pertinents sont le temps d'exécution, le taux de fausses alertes et le taux d'alertes manquées, avec divers réglages de sensibilité et une quantité variable de données étiquetées pour entraîner ALPAGA.

Nous montrons que l'EIA atteint de meilleures performances que n'importe quelle combinaison de ses sous-systèmes et maintient un coût d'exécution compatible avec les exigences des systèmes de supervision. En vérifiant la propriété d'*autonomie,* il répond également à toutes les exigences des systèmes immunitaires artificiels. Les résultats des expériences sur l'EIA valident l'hypothèse selon laquelle les systèmes immunitaires et de supervision sont compatibles et que les premiers peuvent résoudre certains des défis des seconds. Nous proposons en outre qu'en fournissant un cas d'utilisation concret pour des SIA et en introduisant dans la communauté de la supervision de nouveaux concepts issus de l'immunologie, l'EIA peut constituer la base de travaux en commun entre ces deux communautés.

Le chapitre 11 conclut cette thèse, résume ses contributions essentielles et ouvre des perspectives d'amélioration et d'exploration du modèle d'écosystème immunitaire artificiel.

Cloud-Monitor, un détecteur immunitaire

Cloud-Monitor constitue le cœur de l'EIA : son système de détection. En tant que tel, il fournit les propriétés de *mémoire* et de *détection d'anomalies.* Ce modèle a été publié pour la première fois dans [Guigou et al., 2015].

Un système de supervision typique contient un certain nombre de modules de *polling*, c'est-à-dire d'interrogation active, capables d'extraire les métriques de performance critiques des équipements réseau. Ces métriques sont comparées à des seuils d'alerte; une alerte est lancée par un mécanisme adéquat (e-mail, SMS, appel d'API...) lorsque l'un de ces seuils est dépassé. Ce fonctionnement rend mes systèmes de supervision très sensibles au paramétrage : pour chaque métrique, un seuil d'alerte précis doit être fixé. Un seuil trop bas cause des faux positifs tandis qu'un seuil trop haut ne détecte pas à temps les incidents.

Cloud-Monitor est conçu pour accélérer la réaction aux anomalies en introduisant une mémoire et un module de détection d'anomalies. Avec ce modèle, les alertes peuvent être générées non seulement par un dépassement de seuil, mais également par reconnaissance de la signature d'un mode de défaillance connu ou par un écart par rapport au comportement normal de la série temporelle considérée. Ces propriétés répondent aux propriétés de détection d'anomalies et de supervision prédictive citées à la section 11.3.

Ce modèle est comparable aux algorithmes de systèmes imunitaires artificiels. Un SIA typique encode ses données d'entrée en une représentation appropriée, généralement binaire [Bersini, 2002], pour calculer les appariements entre anticorps et antigènes, les antigènes étant générés aléatoirement au cours de la phase d'apprentissage. L'"antigène" est comparé à toute la population d'"anticorps", ce qui induit un coût calculatoire prohibitif en raison du nombre de détecteurs aléatoirement générés (anticorps)

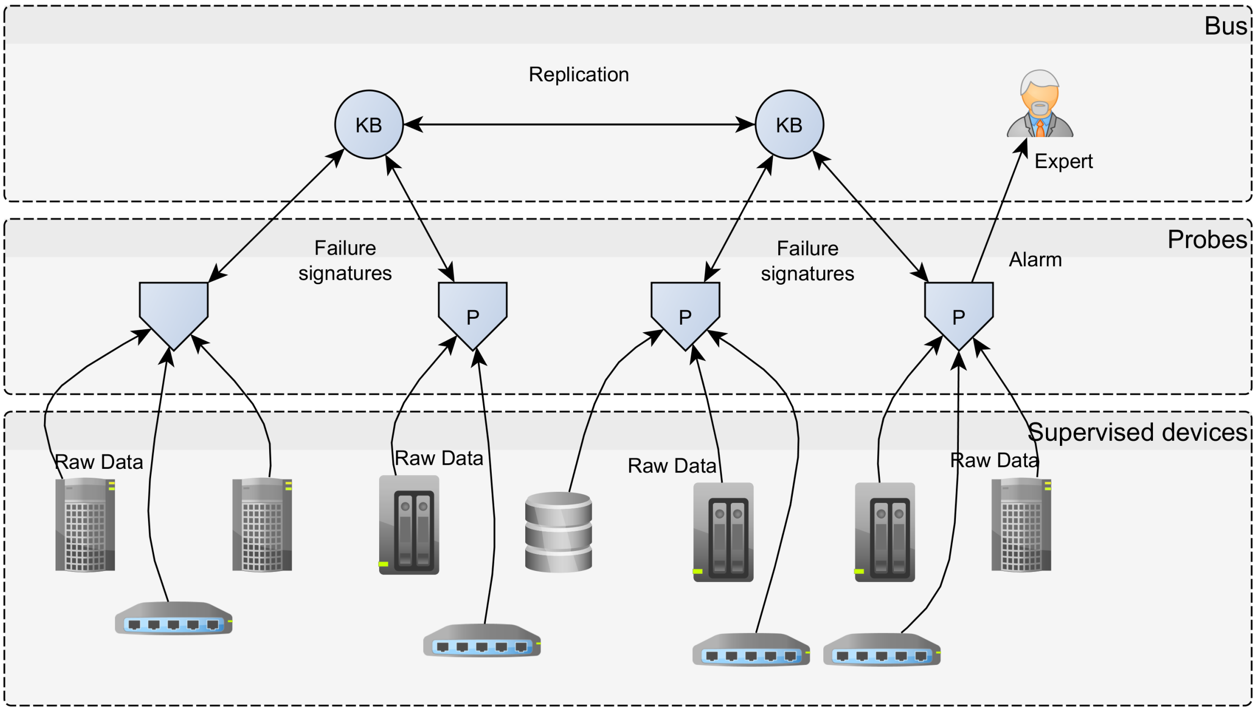


Figure 1 – Architecture de Cloud-Monitor

nécessaires pour correctement apprendre un jeu de données [Stibor et al., 2005]. Cloud-Monitor, en revanche, ne calcule les appariements qu'avec des anticorps correspondant à des antigènes connus, c'est-à-dire des signatures de modes de défaillance déjà rencontrés par le système. La taille de cette population est beaucoup plus réduite, d'où un coût calculatoire plus faible. Ce fonctionnement est possible parce que, contrairement à un SIA classique, Cloud-Monitor ne se repose pas uniquement sur ces anticorps pour détecter des menaces : les dépassements de seuils et la détection d'anomalie sont les modes principaux de détection, et la mémoire immunitaire est utilisée pour reconnaître les signes avants-coureurs d'une typologie d'incident connue, c'est-à-dire pour simuler la réaction immunitaire secondaire.

Cloud-Monitor opère sur les séries temporelles brutes et utilise une base de données de signatures pour enregistrer les modes de défaillance. Cette base de données est l'équivalent de la mémoire immunitaire. Le traitement d'une seule série temporelle étant indépendant des autres, il peut être parallélisé et distribué à volonté. L'architecture est représentée à la figure 6.1. Il est échelonné en trois couches : sur la première, les équipements supervisés, qui ne font pas partie du système lui-même, communiquent avec les sondes, sur la deuxième couche, relevant des métriques de performance. Les sondes (P) effectuent les analyses et partagent les informations communes, telles que les signatures, avec la troisième couche, un bus sur lequel une base de connaissances répliquée (KB) enregistre ces informations ; ce bus est également utilisé par les sondes pour remonter des alertes à l'expert.

Les sondes sont distribuées dans tout le réseau. Elles peuvent être placées à proximité des équipements supervisés, contournant les liaisons WAN et les pare-feu, ou installées en datacenter. Leur rôle est de recueillir les mesures et d'effectuer des analyses. En général, une sonde peut surveiller jusqu'à quelques milliers d'équipements, recueillant des dizaines de milliers de séries temporelles à une fréquence variant entre un échantillon par minute et un échantillon toutes les cinq minutes. Les comparaisons de seuils, à cette échelle, ne sont pas problématiques ; la détection d'anomalies, en revanche, peut devenir rapidement impossible [Keogh et al., 2005].

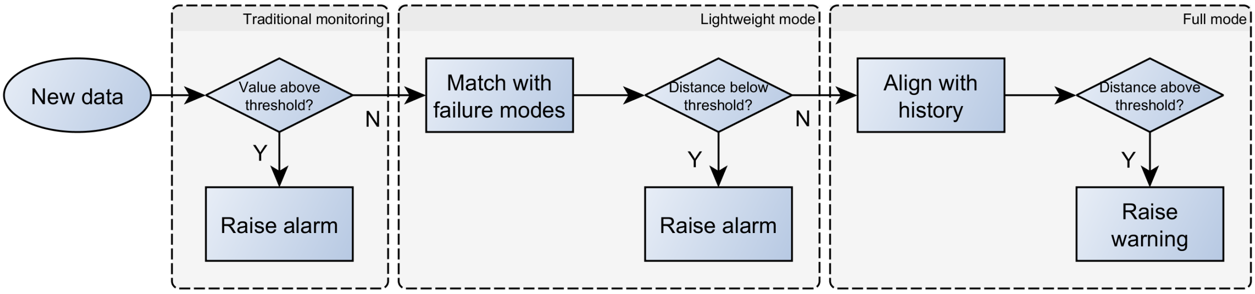


Figure 2 – Flux de données dans Cloud-Monitor

Cloud-Monitor utilise des dépassements de seuils comme les systèmes de supervision classiques, mais les actions qui s'ensuivent sont différentes. Non seulement une alerte est lancée, mais Cloud-Monitor enregistre également une courte fenêtre de la série temporelle avant l'incident et la stocke dans sa base de données de signatures, c'est-à-dire la base de connaissances de la figure 1. Par la suite, lorsque les valeurs de la série considérée correspondent à l'une de ces signatures, une alerte est déclenchée. Cette correspondance est, dans notre implémentation, un seuil appliqué à la distance euclidienne entre les fenêtres d'analyse considérées. Cela permet une réaction plus rapide aux anomalies ou aux modes de défaillance connus (en termes de SIA, après une *exposition secondaire* à un pathogène), et donc une supervision prédictive. Une troisième étape utilise la détection d'anomalies pour détecter les comportements inconnus. Dans notre implémentation, la détection d'anomalies est réalisée par seuillage de la distance euclidienne entre la fenêtre considérée et l'ensemble des fenêtres de même longueur extraites de l'historique de la série ; une heuristique optimisée pour la détection rapide d'anomalies est proposée au chapitre 9. Le flux de données complet est illustré dans la figure 2.

Notons que ce modèle ne tente pas de résoudre tous les problèmes associés aux systèmes de supervision. En particulier, il ne traite pas les problèmes des faux positifs. Sa cible est la détection d'anomalies et la réduction des délais de détection. Les trois étapes illustrées par la figure 2 peuvent être activées ou désactivées séparément. Le mode traditionnel (*traditional mode*) est semblable à tout système de supervision basé sur des seuils. L'ajout du mode léger (*lightweight mode*) fournit une mémoire immunitaire qui peut accélérer la détection des défaillances connues ; enfin, permettre le mode complet (*full mode*), qui ajoute la détection d'anomalies, fournit la capacité de réagir à des modes de défaillance inconnus.

Cloud-Monitor implémente un système de détection immunitaire distribué qui peut être superposé à un système de surveillance classique. Ce point est essentiel pour l'intégration dans les écosystèmes logiciels existants : le mode traditionnel peut être remplacé par tout système de supervision capable d'enregistrer les données de performance et d'effectuer des actions paramétrables lors d'une alerte – par exemple Nagios.

Si nous considérons les propriétés des systèmes de supervisions telles que décrites à le section 11.3, Cloud-Monitor vérifie les suivantes :

- Analyse en temps réel,
- Détection d'anomalies,
- Supervision prédictive.

De manière générale, il ne vérifie pas la propriété de faible taux de faux positifs. Ses capacités de détection accrues génèrent davantage d'alertes, donc nécessairement davantage de faux positifs, qu'un système de supervision classique.

La propriété d'autonomie peut être discutée : le mode traditionnel de Cloud-Monitor est basé sur des seuils et nécessite donc la même configuration qu'un système classique. Toutefois, les deux autres étapes, le mode léger et le mode complet, ne nécessitent qu'un paramètre de sensibilité *global*. Si le mode traditionnel est remplacé par un système de supervision déjà existant, la configuration *supplémentaire* est donc négligeable.

Les détails des modèles statistiques utilisés pour sélectionner les seuils, l'optimisation de l'appariement des signatures et l'évaluation du Cloud-Monitor sont présentés au chapitre 7.

ALPAGA, un système de tolérance immunitaire

Cloud-Monitor, en tant que modèle de mémoire immunitaire simple, est suffisant pour diminuer le temps de détection des défaillances et détecter les anomalies qui ne se manifestent pas lors de la seule application de seuils d'alerte ; cependant, il génère de faux positifs, ce qui viole une contrainte essentielle dans le domaine de la supervision. Pour réduire le taux de fausses alertes, ce système de détection doit être complété par un système de *tolérance*. L'approche choisie dans cette thèse est d'impliquer les utilisateurs finaux, les experts qui reçoivent les alertes de supervision, et de leur donner la possibilité de marquer les fausses alertes en tant que telles. À cette fin, la mémoire immunitaire doit être modifiée pour inclure des étiquettes, afin de pouvoir déterminer si la fenêtre d'une série temporelle qui a déclenché une alarme correspond à une vraie ou à une fausse alerte. Ce processus d'étiquetage transforme un classifieur *supervisé* en un classifieur *actif*. Le flux de données correspondant est représenté à la figure 3.

De la même manière que Cloud-Monitor, ce nouveau méta-algorithme que nous appelons ALPAGA (*Actively Learnt Positive Alerting GAteway*) fonctionne par-dessus un système d'alerting et utilise les alertes qu'il génère, couplées aux données de performance enregistrées, pour alimenter ses analyses. ALPAGA a été publié pour la première fois dans [Guigou et al., 2017b].

En fournissant une tolérance immunitaire, ALPAGA répond à l'exigence de faible taux de faux positifs tout en maintenant le fonctionnement en temps réel et le faible coût de configuration. En particulier, il peut être utilisé par-dessus Cloud-Monitor lui-même, en prenant ses alertes – émises par le système de supervision sous-jacent et par Cloud-Monitor lui-même – et les données de performance historiques qui ont précédé l'alerte comme entrées. C'est le modèle de base de l'EIA, comme nous le verrons à la section 11.3.

Le principe de base de l'apprentissage actif est de raffiner progressivement le modèle appris au lieu d'exiger un réglage à la main fine ou un ensemble de données étiquetées. La configuration totale requise est seulement l'ensemble des hyperparamètres du classifieur. Un faible coût CPU peut être atteint en exploitant la capacité de l'apprentissage actif d'atteindre une haute précision avec de petits ensembles d'apprentissage [Bucurica et al., 2015].

Les expériences au chapitre 8 montrent que les classifieurs les plus efficaces pour l'apprentissage actif sur des fenêtres de séries temporelles sont les modèles de plus proches voisins, qui ne nécessitent aucun paramétrage et restent très légers en termes de coût calculatoire.

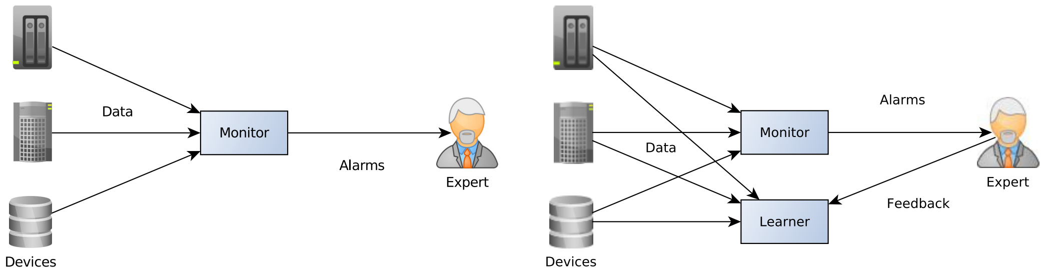


Figure 3 – Flux de données pour un système de supervision simple (gauche) et actif (droite)

Tout comme le mode léger de Cloud-Monitor, ALPAGA implémente une forme de mémoire immunitaire, mais dans un but différent : alors que Cloud-Monitor maintient un répertoire d'échantillons *réactifs*, c'est-à-dire des échantillons causant une réaction immunitaire, ALPAGA construit un répertoire d'échantillons *non réactifs*, fournissant ainsi une tolérance aux phénomènes normaux qui déclencheraient autrement une réponse immunitaire indésirable. Dans les deux cas, la connaissance des signatures qui devraient ou non causer une réaction est construite dans le temps, au fil des retours des experts ; toutefois, deux différences essentielles existent entre Cloud-Monitor et ALPAGA :

Premièrement, la mémoire de Cloud-Monitor est construite automatiquement, sans contrôle ni régulation externe. Le modèle enregistre les cas qui ont causé des alertes par le passé. La mémoire d'ALPAGA, en revanche, est construite en collaboration avec un opérateur humain : contrairement à Cloud-Monitor, elle utilise des étiquettes pour distinguer les vraies alertes des fausses.

L'intervention d'un agent externe s'écarte de la métaphore immunitaire telle qu'elle est généralement comprise, dans la mesure où le système immunitaire est généralement considéré, du moins dans la communauté des SIA, comme un système autonome. Les SIA ont surtout été utilisés sur des ensembles de données étiquetés ou pour des tâches telles que la détection d'anomalies qui entrent dans la catégorie de l'apprentissage non supervisé. À notre connaissance, les SIA n'ont jamais été considérés comme des classifieurs potentiels pour l'apprentissage actif. La participation d'un agent externe offre toutefois certains avantages :

- Contrairement à un modèle auto-régulé, ou tout modèle n'utilisant aucune entrée utilisateur après sa phase d'apprensissage, un classifieur interagissant avec un agent externe peut être contrôlé et corrigé lorsque son comportement dévie de l'attendu [Zhang and Nakamura, 2006].
- Les sorties d'un tel modèle ne dépendent pas de paramètres ajustés finement par un expert [Aickelin and Cayzer, 2008, Aickelin et al., 2004].
- Le modèle ne nécessite pas que des signaux complexes tels qu'un signal de danger ou un niveau d'agression, typiques des SIA, soient exprimés dans le domaine d'application cible [Montechiesi et al., 2015].
- Le modèle peut capturer de la connaissance à la volée, sans nécessiter de modélisation explicite de la part des experts [Tan, 2017].

Deuxièmement, Cloud-Monitor est un système à boucle ouverte : la détection d'un incident conduit à la génération d'une nouvelle signature, ce qui conduit toujours à plus de détections. ALPAGA, en revanche, est un système à boucle fermée : il utilise une boucle de rétroaction pour supprimer les fausses alertes. Une fausse alerte conduit à une nouvelle signature, ce qui conduit à la suppression d'alertes similaires.

L'évaluation de différents classifieurs dans le méta-algorithme d'ALPAGA et l'influence de la disponibilité d'un expert pour étiqueter les alertes sont présentées au chapitre 8, en même temps que les détails du méta-algorithme proprement dit. Ces évaluations montrent qu'un très petit jeu de données d'apprentissage (moins d'une centaine de points) peut être suffisant pour diminuer fortement le taux de fausses alertes d'un système de supervision. Dans le cadre de l'EIA, ALPAGA est utilisé pour filtrer les alertes générées par Cloud-Monitor et fournir une forme de tolérance immunitaire.

SCHEDA, un détecteur d'anomalies scalable

L'une des problématiques soulevées par Cloud-Monitor est le coût calculatoire de son étape de détection d'anomalie. Le Score Euclidien d'Anomalie (SEA) est défini au point i d'une série temporelle T de longueur N avec une taille de fenêtre n par :

$$A_i = min_{j=n}^{N} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{j-k})^2} \tag{11.1}$$

Pour chaque i, tout le passé de la série considéré est examiné : la complexité calculatoire de A_i pour une série de longueur N est $O(N)$. Par conséquent, le calcul de l'ensemble des N valeurs possibles de A pour tout i a une complexité de $O(N^2)$. Afin de limiter les temps de calcul résultants, nous proposons SCHEDA (Scalable Causal Heuristics for Euclidean Distance Approximation), un ensemble de trois heuristiques approximant A_i tout en limitant le coût calculatoire associé. Ces heuristiques sont présentées par ordre décroissant de complexité :

Échantillonnage Euclidien Périodique (E2P) La première heuristique de SCHEDA est l'Échantillonnage Euclidien Périodique (E2P). Le principe d'E2P est que la distance entre une fenêtre donnée d'une série temporelle et deux fenêtres voisines dans le temps (au point d'être fortement superposées) est sensiblement identique. Par conséquent, il peut être suffisant, pour obtenir une approximation du minimum de ces distances, d'en calculer seulement une sur cinq, ou dix, et ainsi de réduire d'autant le temps de calcul associé. E2P nécessite un paramètre : le facteur de sous-échantillonnage s. L'approximation de A_i donnée par E2P est alors :

$$A_i = min_{l=1}^{\frac{i}{s}} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{i-l*s-k})^2} \tag{11.2}$$

Les tests réalisés au chapitre 9 montrent que cette approximation est la meilleure des trois heuristiques. Toutefois, malgré l'accélération obtenue, cette méthode a toujours une complexité quadratique en $O(\frac{N}{2s})$.

Échantillonnage Euclidien Aléatoire (E2A) L'Échantillonnage Euclidien Aléatoire (E2A) calcule un ensemble de distances choisies aléatoirement. Son seul paramètre est le nombre de distances à calculer r. Du fait de son caractère aléatoire, il peut être utilisé sur tout type de signal, sans hypothèse de stationarité ou de périodicité. Il est défini par :

$$A_i = min_{l=1}^{r} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{i-R-k})^2} \tag{11.3}$$

où R est un entier aléatoire compris entre n et i généré à chaque itération. Le nombre de distances à calculer étant fixe, la complexité de l'évaluation d'un point A_i est $O(1)$ et la complexité du calcul de l'ensemble des valeurs de A pour une série temporelle complète est $O(rN)$. Les tests montrent que E2A est considérablement moins coûteux que E2P mais approxime correctement le SEA.

Échantillonage Euclidien Épars (E3) La dernière heuristique proposée est l'Échantillonage Euclidien Épars (E3). Cette heuristique est extrêmement efficiente pour des séries périodiques. E3 calcule uniquement un très petit ensemble de distances à certains intervalles prédéfinis dans le temps (par exemple 1440 points, soit un jour dans le passé, ou 2880 points, deux jours, ou 10,080 points pour une semaine). Les tests montrent que 6 intervalles seulement, soit six distances par point, sont suffisants pour atteindre des performances similaires aux autres heuristiques. E3 est paramétré par l'ensemble des intervalles $L_1, L_2, ..., L_m$ auxquels les calculs de distance ont lieu. E3 définit le score d'anomalie à un point i comme :

$$A_i = min_{l=1}^{m} \sqrt{\sum_{k=0}^{n-1} (T_{i-k} - T_{i-L_l-k})^2} \tag{11.4}$$

Ces intervalles peuvent être connus à l'avance, par exemple en remarquant la périodicité quotidienne et hebdomadaire de la plupart des séries temporelles associées à la supervision et à d'autres activités humaines [Espinoza et al., 2005], ou calculés par des méthodes spectrales ou de détection de pic. Si la période n'est pas connue et est susceptible de varier, il peut être nécessaire de répéter régulièrement une telle analyse, par exemple tous les mois.

Les évaluations au chapitre 9 montrent que ces trois heuristiques ont des performances équivalentes au SEA sur des problèmes de détection d'anomalies et, dans certains cas, donnent de meilleurs résultats, pour un temps de calcul jusqu'à 60,000 fois inférieur. Dans le modèle final de Cloud-Monitor et de l'EIA, la détection d'anomalies est implémentée par SCHEDA.

L'écosystème immunitaire artificiel

L'écosystème immunitaire artificiel est le système résultant de l'interaction entre Cloud-Monitor, ALPAGA et SCHEDA. Le premier apporte la *mémoire*, le second la *tolérance* et le troisième la *détection d'anomalies* – les trois propriétés principales des systèmes immunitaires sur lesquelles nous nous concentrons dans cette thèse. ALPAGA prendre en entrée les alertes de Cloud-Monitor, qui peut lui-même "emprunter" n'importe quel système de supervision pour son more traditionnel. Le principe de base est illustré dans la figure 4. La boucle de rétroaction dans laquelle l'expert est intégré est surlignée en rouge. Les encadrés pointillés montrent les rôles des différentes contributions présentées dans cette thèse : Cloud-Monitor pour la détection, ALPAGA pour le filtrage d'alerte, SCHEDA pour la détection rapide d'anomalies, et enfin l'EIA global.

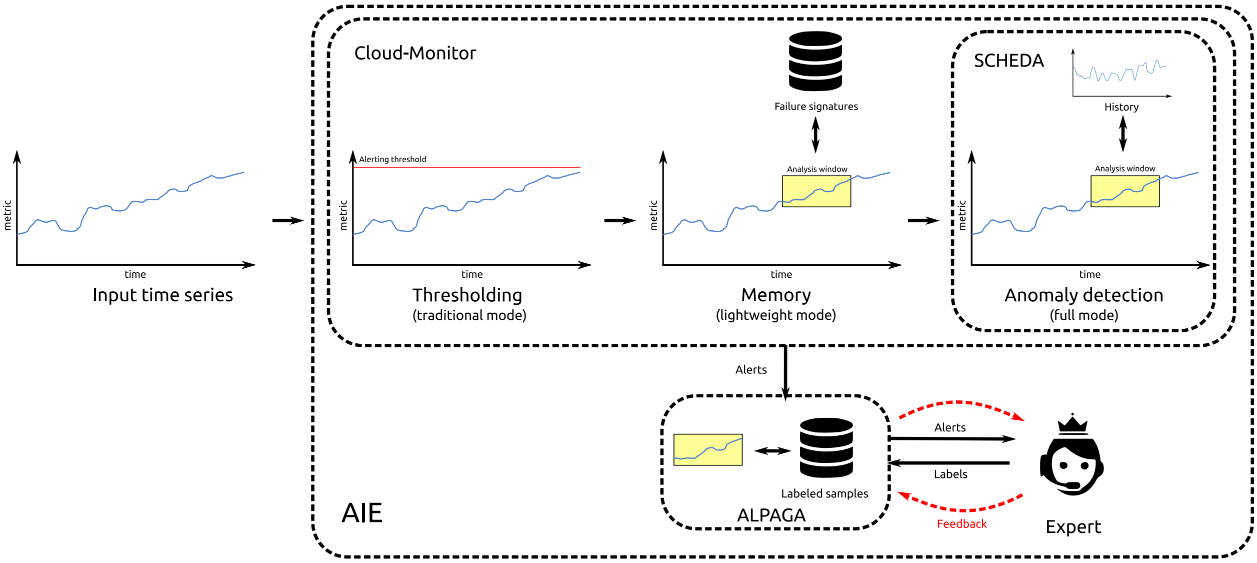


Figure 4 – L'EIA comme combinaison de Cloud-Monitor, ALPAGA et SCHEDA

Le modèle final de l'EAI, incluant les modifications et les optimisations rendues possibles par les résultats des évaluations de Cloud-Monitor, ALPAGA et SCHEDA, est présenté et évalué au chapitre 10. Ces résultats montrent notamment que l'EIA vérifie les propriétés des systèmes immunitaires et de systèmes de supervision : détection d'anomalies, supervision prédictive, analyse en temps réel, autonomie, faible coût calculatoire et faible taux de faux positifs.

Conclusion

L'EIA présenté dans cette thèse constitue le premier pas vers une nouvelle compréhension de la métaphore immunitaire pour les systèmes informatiques, non pas comme une inspiration pour un algorithme, mais comme un ensemble de propriétés qui découlent d'interactions particulières entre ses divers composants. Le modèle et l'implémentation présentés et évalués dans cette thèse ne sont pas la forme finale de l'EIA ; ils constituent plutôt une preuve de concept appliquée au problème de l'analyse de séries temporelles issues de la supervision de réseaux.

L'EIA démontre que le système immunitaire peut être une source d'inspiration non seulement pour les algorithmes, mais aussi pour l'architecture logicielle. Le succès de cette première version montre que les propriétés du système immunitaire biologique peuvent être mises en œuvre par les *interactions* entre des algorithmes plutôt que les algorithmes eux-mêmes. Nous espérons que cette nouvelle perspective sur les SIA constitue un moyen de créer un nouveau pont entre immunologistes et informaticiens, comme l'a fait la recherche sur les SIA dans les années 1990.

La prochaine étape dans l'étude de l'EIA est un déploiement réel dans un réseau à l'échelle d'un fournisseur de services Cloud et Internet, avec des utilisateurs réels remplissant la mémoire immunitaire et étiquetant les alertes. Les autres axes de recherche consistent à rendre l'EIA plus "immunitaire", c'est-à-dire à mettre l'accent sur les propriétés secondaires du système immunitaire pour obtenir une meilleure précision.

L'autre travail essentiel à réaliser autour de l'EAI consiste à utiliser le même schemé d'interaction, le même méta-algorithme sur un type de données différent. Nous pensons que l'analyse de logs serait un excellent cas d'utilisation et démontrerait l'universalité de notre approche. À cette fin, d'autres mesures de distance, heuristiques et classifieurs devraient être utilisés, mais si leurs interactions demeurent les mêmes, avec un moteur de règles générant des alertes (mode traditionnel) stockées dans une mémoire immunitaire (mode léger) et un module de détection d'anomalies (mode complet), l'ensemble des alertes générées étant classées par un filtre entraîné par apprentissage actif (ALPAGA), alors le concept d'un écosystème immunitaire artificiel se serait avéré applicable à d'autres scénarios que l'analyse de séries temporelles.

Fabio GUIGOU

The Artificial Immune Ecosystem

Résumé

Introduits au début des années 1990, les systèmes immunitaires artificiels visent à adapter les propriétés du système immunitaire biologique, telles que sa scalabilité et son adaptivité, à des problèmes informatiques : sécurité, mais également optimisation et classification. Cette thèse explore une nouvelle direction en se concentrant non sur les processus biologiques et les cellules elles-mêmes, mais sur les interactions entre les sous-systèmes. Ces modes d'interaction engendrent les propriétés reconnues du système immunitaire : détection d'anomalies, reconnaissance des pathogènes connus, réaction rapide après une exposition secondaire et tolérance à des organismes symbiotiques étrangers. Un ensemble de systèmes en interaction formant un écosystème, cette nouvelle approche porte le nom d'*Écosystème Immunitaire Artificiel.* Ce modèle est mis à l'épreuve dans un contexte particulièrement sensible à la scalabilité et à la performance : la supervision de réseaux, qui nécessite l'analyse de séries temporelles en temps réel avec un expert dans la boucle, c'est-à-dire en utilisant un apprentissage *actif* plutôt que *supervisé.*

Mots-clefs : *système immunitaire artificiel, supervision réseau, analyse de séries temporelles, apprentissage actif*

Résumé en anglais

Since the early 1990s, immune-inspired algorithms have tried to adapt the properties of the biological immune system to various computer science problems, not only in computer security but also in optimization and classification. This work explores a different direction for artificial immune systems, focussing on the interaction between subsystems rather than the biological processes involved in each one. These patterns of interaction in turn create the properties expected from immune systems, namely their ability to detect anomalies, memorize their signature to react quickly upon secondary exposure, and remain tolerant to symbiotic foreign organisms such as the intestinal fauna. We refer to a set of interacting systems as an ecosystem, thus this new approach is called the *Artificial Immune Ecosystem.* We demonstrate this model in the context of a real-world problem where scalability and performance are essential: network monitoring. This entails time series analysis in real time with an expert in the loop, *i.e.* active learning instead of supervised learning.

Keywords: *artificial immune system, network monitoring, time series analysis, active learning*