

Time Series Anomaly Detection

for CERN Large-Scale Computing Infrastructure

Author: Matteo Paltenghi

Course of Study

Computer Science and Engineering (Politecnico di Milano)
ICT Innovation (Technische Universität Berlin)



CERN IT Department - Compute and Monitoring -
Resource Provisioning Services (IT-CM-RPS Team)

Academic Year 2019-2020

Thesis submitted for the M.Sc. Degree of Matteo Paltenghi

CERN Supervisor:

Ph.D. Domenico Giordano

Politecnico di Milano Supervisor:

Prof. Ph.D. Giacomo Boracchi

Technische Universität Berlin Supervisors:

Prof. Dr. Klaus-Robert Müller

M.Sc. Lukas Ruff



*To my parents and all my educators, teachers and professors that
contributed with their passions to the person that I am today.*

Abstract

Anomaly Detection in the CERN Data Center is a challenging task due to the large scale of the computing infrastructure and the large volume of data to monitor. At CERN, the current solution to spot anomalous server machines in the computing infrastructure relies on threshold-based alarming systems carefully set by the system managers on time series of performance metrics of each infrastructure component. The goal of this work is to relieve the burden of this complex task and explore fully automated machine learning solutions based on anomaly detection. Moreover, in most real industrial scenarios, labeled data to train supervised machine learning methods are unavailable due to their high cost or difficulties in their collection. Therefore, our focus is on fully unsupervised Anomaly Detection methods and we explore the current state of the art including both traditional anomaly detection ones and also recent successful deep anomaly detection approaches.

In this work, we propose novel formulations of time series specific approaches (CNN Forecaster, VAR Forecaster) and adaptations to reuse traditional machine learning methods (LOF, OCSVM, IFOREST, KNN, PCA) and deep learning ones (Autoencoder Fully Connected, CNN Autoencoder, LSTM Autoencoder) with time series data. In addition we explore six ensemble strategies to combine the individual algorithm strengths. We then present a comparative study of these 10 individual methods and 6 ensemble strategies on the CERN use case for identifying the best approach for the specific problem characteristics of the CERN large-scale computing infrastructure. In addition, given the absence of labelled data we put in place an annotation system that enables to annotate time series efficiently. We use this to collect and create two new time series datasets for anomaly detection and that represent two different CERN user categories.

The results of this study in terms of ROC-AUC detection performance and training time makes a strong point in favour of the traditional methods that for the specific problem at hand work extremely well; on the other hand we also observe that they tend to be outperformed by deep methods whenever the time series patterns for normal instances become more complex. In parallel with our comparative evaluation study, we also publish an open source proof-of-concept anomaly detection system.

Zusammenfassung

Die Erkennung von Anomalien im CERN-Rechenzentrum ist aufgrund des großen Umfangs der Computerinfrastruktur und des großen zu überwachenden Datenvolumens eine herausfordernde Aufgabe. Am CERN basiert die aktuelle Lösung zum Erkennen anomaler Servermaschinen in der Computerinfrastruktur auf schwellenwertbasierten Alarmsystemen, die von den Systemmanagern sorgfältig anhand der Leistungszeitreihenmetriken jeder Infrastrukturkomponente festgelegt werden. Ziel dieser Arbeit ist es, diese komplexe Aufgabe zu entlasten und vollautomatische Lösungen für maschinelles Lernen im Bereich Anomalieerkennung zu untersuchen. Darüber hinaus sind in praktisch jedem realen industriellen Szenario beschriftete Daten zum Trainieren von überwachten maschinellen Lernmethoden aufgrund ihrer hohen Kosten oder Schwierigkeiten bei ihrer Erfassung nicht verfügbar. Daher konzentrieren wir uns auf vollständig unbeaufsichtigte Methoden zur Erkennung von Anomalien und untersuchen den aktuellen Stand der Technik, einschließlich traditioneller Methoden zur Erkennung von Anomalien und neuerer erfolgreicher Ansätze zur Erkennung tiefer Anomalien.

In dieser Arbeit schlugen wir neuartige Formulierungen von zeitreihenspezifischen Ansätzen (CNN Forecaster, VAR Forecaster) und Anpassungen vor, um traditionelle Methoden des maschinellen Lernens (LOF, OCSVM, IFOREST, KNN, PCA) und Deep Learning (Autoencoder Fully Connected, CNN Autoencoder, LSTM Autoencoder) mit Zeitreihendaten. Zusätzlich untersuchen wir sechs Ensemble-Strategien, um die Stärken der einzelnen Algorithmen zu kombinieren. Anschließend präsentieren wir eine vergleichende Studie dieser 10 Einzelmethoden und 6 Ensemble-Strategien zum CERN-Anwendungsfall, um den besten Ansatz für die spezifischen Problemmerkmale der CERN-Großcomputerinfrastruktur zu ermitteln. Da keine gekennzeichneten Daten vorhanden sind, haben wir außerdem ein Anmerkungs-system eingerichtet, mit dem zwei neue Anomalieerkennungs-Datensätze für Zeitreihen erfasst werden können, die zwei verschiedene CERN-Benutzerkategorien darstellen.

Die Ergebnisse dieser Studie in Bezug auf ROC-AUC und Schulungszeit sprechen stark für die traditionellen Methoden, die für das jeweilige Problem sehr gut funktionieren. Andererseits haben wir auch festgestellt, dass sie durch tiefe Methoden immer dann überfordert werden, wenn die Zeitreihenmuster für normale Instanzen weniger trivial werden. Parallel zur Vergleichsstudie haben wir auch ein Open

Source Proof-of-Concept-Anomalieerkennungssystem entwickelt.

Sommario

Il rilevamento delle anomalie nel Data Center del CERN è un compito impegnativo a causa della vasta scala dell'infrastruttura di elaborazione e del grande volume di dati da monitorare. Al CERN, l'attuale soluzione per individuare macchine server anomale nell'infrastruttura informatica si basa su sistemi di allarme basati su soglie impostati con cura dai gestori di sistema sulle metriche delle serie temporali delle prestazioni di ciascun componente dell'infrastruttura. L'obiettivo di questo lavoro è alleviare il peso di questa complessa attività ed esplorare soluzioni di Machine Learning completamente automatizzate nel campo del rilevamento delle anomalie. Inoltre, praticamente in ogni scenario industriale reale, i dati etichettati per addestrare metodi di apprendimento automatico supervisionati non sono disponibili a causa del loro costo elevato o delle difficoltà nella loro raccolta. Pertanto, il nostro focus è sui metodi di rilevamento delle anomalie completamente non supervisionati ed esploriamo lo stato dell'arte attuale, inclusi quelli tradizionali di rilevamento delle anomalie e anche i recenti approcci di rilevamento delle anomalie basati su Reti Neurali. In questo lavoro abbiamo proposto nuove formulazioni di approcci specifici per serie temporali (CNN Forecaster, VAR Forecaster) e adattamenti per riutilizzare metodi di Machine Learning tradizionali (LOF, OCSVM, IFOREST, KNN, PCA) e Deep Learning (Autoencoder Fully Connected, CNN Autoencoder, LSTM Autoencoder) con dati di serie temporali. Inoltre, esploriamo sei strategie di insieme per combinare i punti di forza dei singoli algoritmi. Presentiamo quindi uno studio comparativo di questi 10 metodi individuali e 6 strategie di insieme sul caso d'uso del CERN per identificare l'approccio migliore per le caratteristiche del problema specifico dell'infrastruttura informatica su larga scala del CERN. Inoltre, data l'assenza di dati etichettati, abbiamo messo in atto un sistema di annotazione per rendere possibile la raccolta di due nuovi set di

dati di rilevamento di anomalie per serie temporali che rappresentano due diverse categorie di utenti del CERN. I risultati di questo studio in termini di ROC-AUC e tempo di allenamento costituiscono un punto di forza a favore dei metodi tradizionali che per lo specifico problema in esame funzionano molto bene; d'altra parte abbiamo anche notato che tendono ad essere sorpassati da metodi basati su Reti Neurali ogni volta che i modelli di serie temporali per istanze normali diventano meno banali. Parallelamente allo studio comparativo, abbiamo anche prodotto un sistema Open Source Proof of Concept di rilevamento delle anomalie.

Acknowledgements

I would like to thank my parents for having supported my studies and for their constant presence and thoughtfulness. In particular for this thesis and related experience at CERN I would like to thank very much Domenico for making this internship at CERN possible and for having introduced me to the Cloud Team and the CERN community in the best way possible despite the difficulties in the current situation. Thanks to Tim Bell that as group leader of the Compute and Monitoring team (IT-CM) believed in this project and sponsored this CERN Technical Student position, and to Jan van Eldik that as team leader welcomed me to the Cloud Infrastructure section (IT-CM-RPS) with his energy and positivity. A special thanks also to Prof. Giacomo Boracchi and Lukas for the valuable meetings and the continuous guidance along the entire thesis work. Thanks also to Prof. Dr. Klaus-Robert Müller for being my official supervisor at Technische Universität Berlin. Thanks to EIT Digital for having sponsored my Double Degree between Politecnico di Milano and Technische Universität Berlin, and awarded me with an Excellence Scholarship that gave me independence and made me meet many amazing people part of the program. Among those a special thanks to Snehal for having always believed in me as "Man of Science".

Contents

Abstract	I
Zusammenfassung	III
Sommario	V
Acknowledgements	VII
1 Introduction	1
1.1 CERN Context	1
1.2 Challenges and Contributions	3
1.3 Thesis Structure	4
2 Problem Formulation	7
2.1 Monitoring Infrastructure at CERN	7
2.1.1 The Big Picture: Pipeline Architecture	8
2.2 Modelling Assumptions	10
2.2.1 Two Main User-Categories: “Batch” and “Shared”	10
2.2.2 Homogeneous Group: same Hardware and Software	11
2.3 Concept of Anomaly	12
2.3.1 Anomalous Time Series at CERN	13
2.4 Anomaly Detection Problem	14
2.5 Time Series Anomaly Detection Problem at CERN . .	14
2.5.1 Anomaly Detection in the CERN Scenario . . .	16
2.6 Assumptions	17
3 Background	19
3.1 Anomaly Detection for Computing Infrastructures . . .	19
3.1.1 Challenges	20
3.2 Algorithms for Anomaly Detection	21

3.3	Traditional Anomaly Detection Methods	21
3.3.1	K-Nearest Neighbour	21
3.3.2	Local Outlier Factor	22
3.3.3	Principal Component Analysis for Anomaly De- tection	23
3.3.4	Isolation Forest	24
3.3.5	One-Class Support Vector Machine	25
3.4	Deep Anomaly Detection Methods	25
3.4.1	AutoEncoder	26
3.5	Time Series specific Methods	27
3.5.1	Forecasting Error: Anomaly Detection on Time Series	28
3.5.2	Traditional Time Series Models	28
3.5.3	Deep Time Series Models	30
3.6	Ensemble Methods in Anomaly Detection	30
3.6.1	How to practically combine models in an ensemble	31
4	Dataset Acquisition and Annotation	35
4.1	Need for a Dataset: Research Field Perspective	36
4.2	Extension of Grafana Annotation	36
4.2.1	“Connecting the Dots”	37
4.3	Dataset Creation	40
4.4	Datasets Description	40
4.4.1	Batch Group	40
4.4.2	Shared Group	42
4.5	Artificial Setting Creation	42
5	Anomaly Detection Solution: System Engineering	45
5.1	System Design Challenges	45
5.2	Core Technologies and their Scope	46
5.3	Anomaly Detection Pipeline	47
5.4	Data Preparation	48
5.4.1	Discretization of Time Series	49
5.4.2	Input Normalization	50
5.4.3	Minimal Unit of Analysis	50
5.4.4	Greymap for TimeSeries AD Algorithms	51
5.4.5	Vectorization for Static AD Algorithms	52
5.5	Production Implementation with Apache Airflow	52

6	Anomaly Detection Solution: Data Analytics Core	55
6.1	Train and Test Set	56
6.2	Anomaly Score Normalization	56
6.3	Novel Time Series Methods	57
6.3.1	Forecasting with VAR	58
6.3.2	Forecasting with CNN	59
6.4	Autoencoders for Time Series Modelling	60
6.4.1	Fully Connected Autoencoder	61
6.4.2	CNN Autoencoder	61
6.4.3	LSTM Autoencoder	62
6.5	Ensemble	62
6.5.1	Logistic Regression Ensemble	63
7	Experimental Evaluation	65
7.1	Performance measures	65
7.1.1	Receiver Operating Characteristic	66
7.1.2	Precision, Recall, F1 Score	67
7.1.3	Distribution of Anomaly Scores	68
7.1.4	Computation performance	68
7.1.5	Similarities among Methods	68
7.2	Experiment A - CERN curated Dataset	69
7.2.1	Formalized Training and Testing Procedure	69
7.2.2	Algorithms Training Time	70
7.2.3	Detection Performance	72
7.3	Experiment B - CERN Artificial Setting	73
7.4	Algorithm Implementation Details	76
7.5	Discussion	77
7.5.1	Performance and Task Complexity	77
7.5.2	Input Data Quantity	79
7.5.3	Novel Methods Discussion	80
8	Conclusion	83
	References	87
A	Dataset	95
A.1	Annotations Overview	95

B	Complete Experimental Results	99
B.1	Performance Metrics: Complete Analysis	99
B.2	Correlation of Anomaly Scores	99
B.3	Ensemble Performance	99
C	Appendix	109
C.1	Code Modification for Grafana Extension	109
C.2	ElasticSearch Document Format	109
C.3	Apache Spark - Big Data Preparation	112
	C.3.1 Data Reading	112
	C.3.2 Time Series/Metrics Aggregation	114
	C.3.3 Time Series/Metric Normalization	116
C.4	Deep Learning Architecture Details	116
C.5	Image Credits	117
C.6	Disclaimer: Connection with previous work	119

Chapter 1

Introduction

Industry devices such as server machines, engines, robots, etc. are the basis of the correct functioning of virtually any complex industrial system (e.g. data centres, car production factory, etc.). Also the healthcare sector is increasingly using sensors to monitor the patient's status and robotic environments to improve the quality of surgery. The common denominator of all these complex systems is that they rely on the underlying hardware and software to work efficiently and in a safe manner. Unfortunately, the more complex a system is, the more likely it is to suffer from system failures during its lifetime. As a consequence, its failures will inevitably degrade the performance, which eventually can lead to a cascade effect on the whole system. Depending on the nature of the system this cascade failure can be translated in a loss of money [1] or a bad outcome for the patient that can also lead to death in the worst-case scenario [2]. Therefore, in these contexts where we heavily rely on machines, it is crucial to monitor those complex systems and to be informed on the status of their components in order to react promptly and avoid bigger losses from late intervention. Detecting anomalies in the data here can provide useful information to reveal concrete hardware or software failures.

1.1 CERN Context

This thesis was created during an internship I conducted at CERN under the supervision of PhD. Domenico Giordano. As part of the *Openstack Cloud team* I had the opportunity to work closely with colleagues responsible for the maintenance (i.e. *operations*) of the cloud

infrastructure of the CERN Data Centre. Every experiment for High Energy Physics (HEP) at CERN requires millions, if not billions of CHF of investment [3], therefore every step in the chain that lead to new results for HEP is fundamental. In particular, the elaboration and computation of results is crucial and it happens in the CERN Data Centre, which, with a large number of server machines, renders itself as a perfect case of those complex systems just mentioned. Consequently, a continuous monitoring of the data center is required to ensure its correct functioning. The primary goal of a monitoring infrastructure is to give a comprehensive and complete view of the system via monitoring dashboards; this is already in place at CERN with an excellent monitoring infrastructure. A second and even more critical quality of an effective monitoring infrastructure is to promptly inform the user in case of anomalies in the functioning of the complex system at hand. Being able to observe the system in real-time is not sufficient when the number of entities becomes considerably large. In particular we cannot expect to allocate a proportionate number of human resources to manually monitor every entity. For this reason, there is a need for an anomaly detection (AD) system that can automatically and reliably spot the problems of the system ideally on par or even better than the human expert.

Here, the notion of anomaly is central and is often paired with the concept of outlier. One of the first definitions of an outlier dates back to 1969: “An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs” [4]. At that time the goal of outlier detection was closely linked to the problem of data quality, since for a good data mining application the presence of outliers can be often problematic therefore the identification of outlier patterns is functional to their removal from the dataset for a better data mining result. On the other hand, the concept of anomaly detection puts more focus on spotting outliers because they represent interesting events that we want to be informed of (e.g. failures). Depending on the meaning behind the outlier in the literature, another synonym is novelty detection.

In a modern data center structure where the data are in the form of time series the AD task has to consider the discoveries of anomalies across time [5]. Therefore for CERN Data Centre, an outlier intuitively corresponds to an unexpected event, or set of events, that is

rare and differs in various ways from the majority of usual events.

1.2 Challenges and Contributions

Given the size of the CERN context with more than 7'000 physical servers on premise and a monitoring data rate production in the order of terabytes, the CERN Data Centre resembles a Big Data scenario. In this context, as part of my thesis and internship, I was responsible for the designing and implementation of a fully-functioning proof of concept for an anomaly detection system for the CERN Monitoring Infrastructure. To achieve this paramount goal, my thesis work involved two main components in both Data Engineering and Machine Learning Engineering.

In terms of Data Engineering, I faced and overcome the following challenges:

1. **Data Preparation:** I prepared and processed the huge volume of data produced by the monitoring infrastructure in a scalable way with Apache Spark.
2. **Infrastructure Design:** I designed, implemented, and integrated a pipeline to publish the result candidate anomalies in the monitoring platform for visualization into the monitoring infrastructure, together with a procedure to let the CERN experts label the time series data as anomalous or normal.
3. **Dataset Collection:** I facilitated the collection of two new CERN anomaly detection datasets on time series that will possibly be shared with the scientific community allowing scientists to benchmark their methods on the industrially relevant CERN use case.

From the algorithmic perspective the initial lack of a labelled dataset for anomaly detection at CERN and also the limited human resources available for annotating this complex data excluded the direct application of supervised techniques in a scalable way. Therefore we found it natural to investigate fully unsupervised Anomaly Detection methods since the very beginning. For the data analytics core of the anomaly detection system, I worked on the following tasks:

1. **Modelling:** I modelled the Anomaly Detection on Time Series problem at CERN and adapted traditional as well as deep anomaly detection methods that are designed to detect static outliers to work on our time series data (LOF, OCSVM, IFOREST, KNN, PCA, Autoencoder, CNN Autoencoder, LSTM Autoencoder).
2. **Novel usage of Existing Models:** I designed variants of existing models for our specific Anomaly Detection formulation (CNN Forecaster, VAR Forecaster).
3. **Comparative Study:** I compared a pool of selected unsupervised Anomaly Detection methods on the two newly collected CERN dataset on time series anomaly detection.

The major contributions of this work to the research field include: a comparative evaluation study of some of the most used Anomaly Detection methods, both traditional and Deep Learning ones, on the industrially relevant CERN use case, a collection of two new CERN anomaly detection for time series datasets and finally an open source software implementation of a proof of concept for an anomaly detection system produced for CERN in the scope of the internship [6].

1.3 Thesis Structure

The thesis is organized as follows:

- In Chapter 2, we first give the details about the CERN cloud computing architecture and formally define the problem we address.
- In Chapter 3, we introduce background concepts necessary to understand this research work; we then review literature on traditional and Deep Learning Anomaly Detection methods.
- In Chapter 4, we describe how we use and adapt the CERN infrastructure to annotate and collect the two CERN Anomaly Detection datasets.
- In Chapter 5, we present details about the design of our Anomaly Detection System and how we prepare the data with it.

- In Chapter 6, we focus on the Data Analytics core of the System describing the novel adaptation of previous methods to the time series scenario.
- In Chapter 7, we assess the performance of the evaluated methods, describing the evaluation procedure and commenting on the experimental results.
- In Chapter 8, we summarize the contributions of this work, present our conclusions and suggest possible future research directions.
- In Appendix A, we present further details about the collected dataset.
- In Appendix B, we collect an extensive version of the evaluation results.
- In Appendix C, we present other complementary details about the system in terms of software contribution.

Chapter 2

Problem Formulation

In this chapter we give the details about the specific CERN cloud computing architecture and formally define the Anomaly Detection problem we address. Finally we present the main assumptions underlying the proposed formulation.

2.1 Monitoring Infrastructure at CERN

To accomplish its research goals on High Energy Physics, CERN relies heavily on the computational power of its Data Centre. The computational resources form a Private Cloud and are managed via components of Openstack, the free open standard cloud computing platform. OpenStack provides an Infrastructure-as-a-Service solution integrated with CERN’s computing facilities. CERN users can request on-demand virtual machines resources for production, test and development purposes. At the time of writing, the CERN Data Centre contains 7 487 hypervisors running 45 362 Virtual Machines (VMs) in total.

Virtual Machine: A compute resource that runs in a software environment instead of directly on a physical computer.

Hypervisor: Software that creates and runs virtual machines (VMs) [7]. It is sometimes also called virtual machine monitor (VMM). In this work we also equally refer to an hypervisor with the terms: *server*, *host*, *machine*.

Those machines run either computations for the various physics experiments or applications and services developed by CERN users to maintain CERN internal IT services. In both cases a malfunctioning of those servers can degrade the quality of service for its users. As a consequence CERN invested considerable amount of resources in making sure that its Data Centre can run as smoothly as possible. The main Business Unit created for this specific task in 2016 is the Monitoring Team from the IT Department whose mission is to “collect, transport, store and process metrics and logs for applications and infrastructure” [8]. To provide an effective Monitoring Infrastructure, the Monitoring Team installed a system statistics collection daemon named Collectd on every single hypervisor.

Collectd: System statistics collection daemon that periodically checks some parameters of the hypervisors and store those values in the Monitoring infrastructure acting as a sensor. This daemon is modular thanks to a plugin system that let the system admin decide which parameters to collect in every hypervisor. Some example of plugins are: CPU, Memory and ContextSwitch. Each of this plugins can collect more than one parameters.

Metric: Time series data representing the evolution of single parameters collected by one Collectd Plugin installed on the hypervisor. In this work we also equally refer to a single metric with the terms: *metric, time series, plugin*.

Definition 2.1.1. Time Series: is an ordered sequence of numerical data points $\vec{a} = \{a_0, a_1, a_2, a_3, \dots, a_t\}$ where the order is determined by the timestamp $\{t_0, t_1, t_2, t_3, \dots\}$ in which those data points are observed. It is naturally unbounded in continuous data streams and the most recent data point is referred to as a_t .

2.1.1 The Big Picture: Pipeline Architecture

The current configuration of the Monitoring Infrastructure is composed by five consecutive layers or stages, each of them relies on well known open-source technologies:

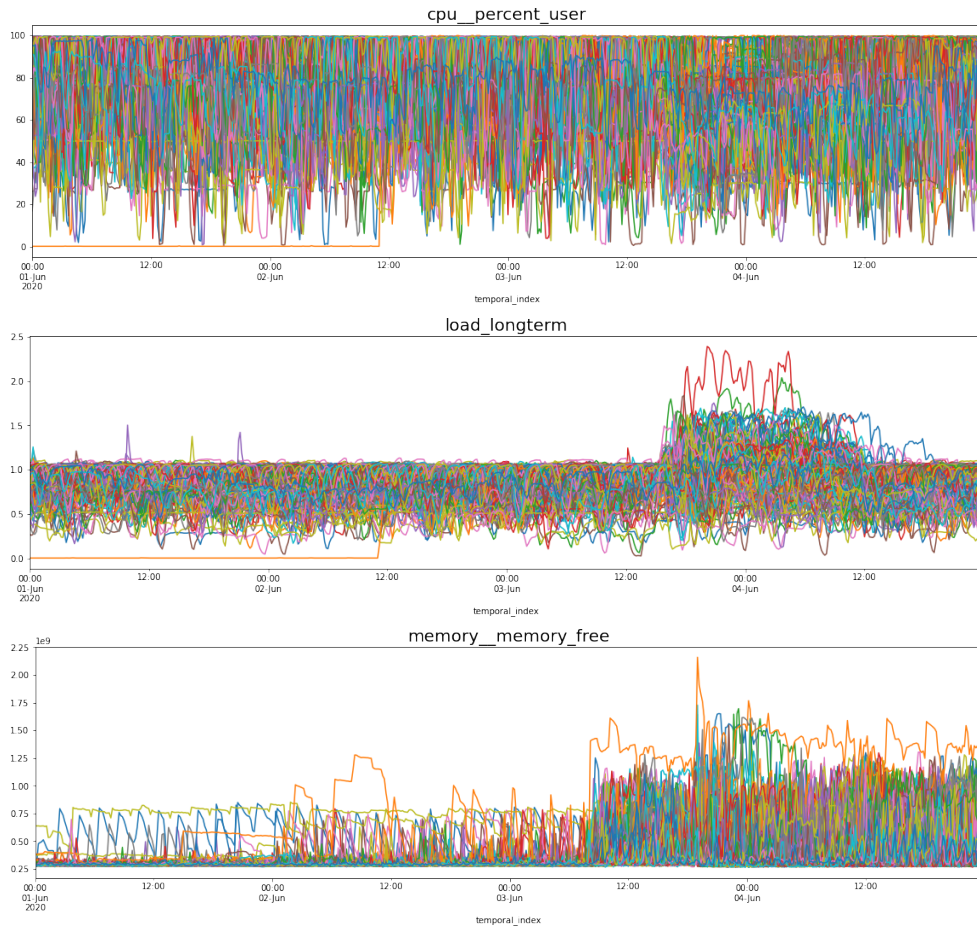


Figure 2.1: An example of three hypervisor's metrics: cpu user percentage, cpu load, memory usage. Data representing the first 4 days of June. Every time series represents a different hypervisor. The hypervisors shown are part of an Openstack Cell devoted to Batch usage (named: Geneva Project 013)

1. Sources: Where the raw data (e.g. metrics, logs, etc) are produced;
2. Transport: That serves as a connector from the heterogeneous data sources to a unified preprocessing engine;
3. Processing: Where data are aggregated and preprocessed thanks to technologies like Spark;
4. Storage: Where the processed data are persistently saved for future access;
5. Access: Where the data are available to the user for interactive exploration or systematic analysis.

Note that a detailed description of the components and technologies that were used will be given in Section 5.2.

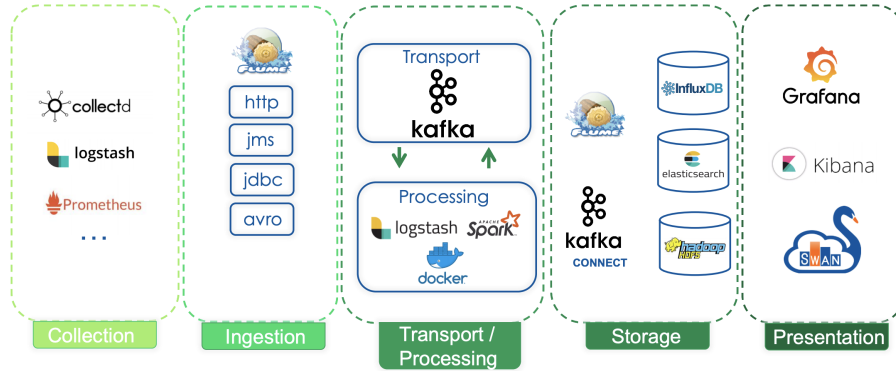


Figure 2.2: Monitoring Infrastructure at CERN. [9]

2.2 Modelling Assumptions

In this section we present the two principal assumptions that let us split the analysis into two main user categories, also called Groups.

2.2.1 Two Main User-Categories: “Batch” and “Shared”

Depending on the computational needs, we have two main user categories. The first one is represented by CERN users or services that are independent from each other and require a relatively low number of hypervisors, whereas the second one consists in physicists that need

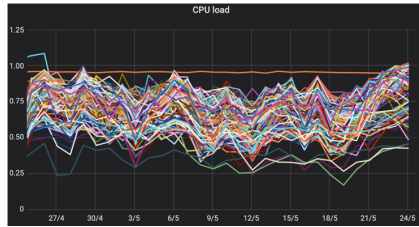
a large pool of computational resources to elaborate the data from an experiment in a compute intensive way. For the first user category we can assume to have independent and intermittent jobs running on the various hypervisors while in the second case we have jobs that work in a batch and continuous mode showing relatively regular patterns. In Figure 2.3 we can see the difference between the two categories on the *CPU Load* metric.

Definition 2.2.1. User categories: Every Hypervisor h_i is uniquely assigned only to one target usage:

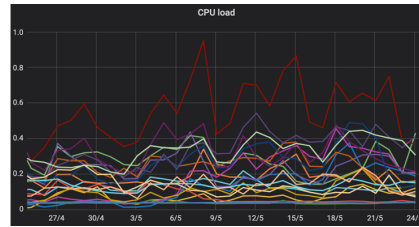
$$\forall h_i \in DC, ((h_i \in S) \vee (h_i \in B)) \wedge (S \cap B = \emptyset)$$

Where DC represent the set of machines in the Data Centre, whereas S represents the set of machines assigned to the *Shared* usage and B to the *Batch* usage.

Set name	User Category	Pattern Usage	Objective
S	Shared by User/Services	Intermittent / Irregular	Maximize the No. of independent jobs satisfied
B	Batch (Physics Experiments)	Regular	Maximize the utilization



(a) *Batch* user category



(b) *Shared* user category

Figure 2.3: Typical CPU load patterns in the two user categories. Each image shows a single group and every line represents the CPU load of different hypervisors.

2.2.2 Homogeneous Group: same Hardware and Software

To support the various user categories, the compute infrastructure is organized accordingly. The hypervisor dedicated for a common goal are grouped in Openstack Cells.

OpenStack Cell: functionality to ease the scaling of an OpenStack cloud in a more distributed way and support very large deployments. For the scope of this work we will treat it as an aggregate of hypervisors.

More formally we define it as:

Definition 2.2.2. Group (or Openstack Cell):

$$G = \{h_1, \dots, h_n\}$$

is a set of n hypervisors with homogeneous Software and Hardware configuration. Moreover they share also the same target usage: batch or shared.

$$(\forall h_i \in G, h_i \in S) \vee (\forall h_i \in G, h_i \in B)$$

2.3 Concept of Anomaly

Before diving into the details of our problem formulation, we informally introduce the concept of anomaly. In the spoken language we refer to an anomaly whenever we have something that “is unusual enough to be noticeable or seem strange” (Cambridge Dictionary). We find other more precise description if we refer to the data science literature on the concept of statistical outlier, that is intertwined with the notion of anomaly. According to Hawkins [10]:

“an outlier is an observation which deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism”.

Whereas Barnett and Lewis [11] defines it as:

“an observation [...] which appears to be inconsistent with the remainder of that set of data”.

These definitions add two important attributes to our concept of anomaly, respectively the fact that an anomalous observation is likely generated by a different mechanism and that to discover it we require a peer confrontation with other observations, thus the anomaly is distribution dependant.

2.3.1 Anomalous Time Series at CERN

In our scenario we are reasoning on time series data, therefore, to familiarize with our anomaly detection problem statement, we introduce and discuss some typical examples of time series anomaly at CERN.

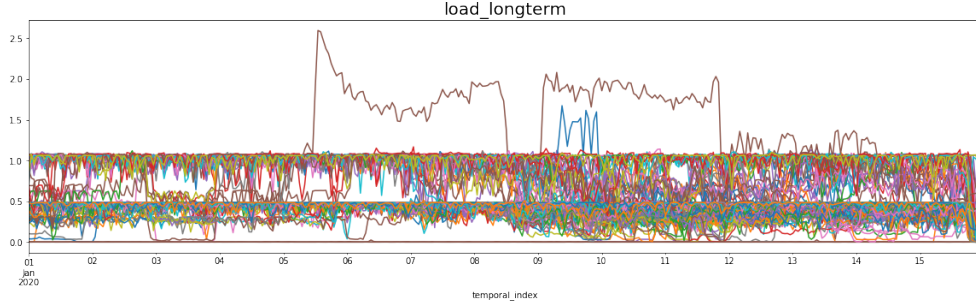


Figure 2.4: Example of anomalies in a Batch group. The brown hypervisor that peaks up to cpu load of 2.5 shows an anomalous behaviour because its load increases above the normal expected behaviour. Other three hypervisors have zero CPU load; that means they are inactive, potentially wasting resource. Ultimately another subset of machines is used only at half of its power with load equals to 0.5.

Figure 2.4 shows some typical anomalous behaviours regarding the metric *CPU LOAD* of the hypervisors in a *Batch* group. The *CPU LOAD* is a measure of the average system utilization during a time period of 5 min, 10 min, 15 min; in this example it is 15 min. *CPU Load* equal to 1.73 means that we have 1.73 runnable processes in the hypervisor, so that on average 0.73 processes had to wait in the queue for a CPU clock (if we assume to be on a single CPU system). The anomalous behaviours can be present with respect to different relative point of reference:

- **The past of the same entity.** A hypervisor is increasing its activity in a way that never happened in the past. As it can be seen in Figure 2.4 where some machines go from a CPU load of 1 to 2.
- **The behaviour of other entities.** Some hypervisors are having a CPU load of 0.5 and that is strange if compared to the majority of the other machines that instead are fully utilized with load equals to 1, as it is expected in the ideal case for the *Batch* group.
- **In absolute.** An hypervisor with load equals to 0 means an idle resource and therefore a potential waste of resources. In this case

it is not necessary to have particular information on the past or the other machines, an idle machine is always bad and we want to identify it.

2.4 Anomaly Detection Problem

Given a Dataset D composed of n samples $D = \{x_1, \dots, x_n\}$ where x_i is a multidimensional datapoint $x_i \in R^k$, the goal of Anomaly Detection is to model a binary prediction function $AD(x) : R^k \rightarrow 0, 1$ such that:

$$AD(x) = \begin{cases} 1 & \text{if } x \text{ is anomalous} \\ 0 & \text{if } x \text{ is normal} \end{cases} \quad (2.1)$$

This is usually achieved through a two step procedure: learning some scoring function followed by the application of a threshold. The scoring function $s(x) : R^k \rightarrow R$ assigns to each sample an anomaly score indicating its degree of anomalousness:

$$\forall x_a, x_b \in D, s(x_a) > s(x_b) \text{ if } x_a \text{ is more anomalous than } x_b \quad (2.2)$$

We then apply a thresholding function $T_c(s(x)) : R \rightarrow 0, 1$ to binarize the score, where c is the threshold to discern between normal and anomalous data. The value of the cut-off point c is chosen using a validation dataset so that the false alarms rate stays below a certain desired value.

$$T_c(s(x)) = \begin{cases} 1 & \text{if } s(x) \geq c \text{ when } x \text{ is anomalous} \\ 0 & \text{if } s(x) < c \text{ when } x \text{ is normal} \end{cases} \quad (2.3)$$

2.5 Time Series Anomaly Detection Problem at CERN

The first step, when working with time series data, is to determine what the minimal unit of analysis is. Since time series streams are naturally unbounded, we adopt the windowing approach and consider for each hypervisor a fixed length window of data.

Definition 2.5.1. Discretized Time Series: given a time series $\vec{a} = \{a_0, a_1, a_2, a_3, \dots, a_s\}$ representing the sensor reading of one sensor attached to one hypervisor, $\vec{d}$ is a derived time series where each data

point is a mean summary statistics of the values of $\vec{a}$ in the previous c minutes: $\vec{d} = \{d_0, d_1, d_2, d_3, \dots, d_t\}$ with $t \leq s$. Every element d_i summarizes the mean of a disjoint consecutive subset A_i of $\vec{a}$. A_i contains a variable number of elements that were recorded in x consecutive minutes. The start can be fixed arbitrary.

Definition 2.5.2. Univariate Temporal Window: given a discretized time series $\vec{d} = \{d_0, d_1, d_2, d_3, \dots, d_t\}$, a Univariate Temporal Window $\vec{w}$ is a vector that represent a subsequence of $\vec{d}$ made of w consecutive data points $\vec{w} = \{d_i, d_{i+1}, \dots, d_{i+w-1}\}$, where w is the window length.

We decide to analyse our time series in separate **non-overlapping** temporal windows because we want to contain the computational cost of the analysis. Since in our scenario every time series is produced by an hypervisor, we can see the windows as a representation of the health status of that hypervisor. Nonetheless this is only a partial view of our hypervisor because each hypervisor produces m metrics. Therefore we extend our window representation to include also the other time series produced by the same hypervisor.

Definition 2.5.3. Multivariate Temporal Window (or Window): given a hypervisor h and the set D of the m time series metrics it produces $D = \{\vec{d}^0, \dots, \vec{d}^m\}$, a Multivariate Temporal Window is a matrix W that has as rows the m Univariate Temporal Windows of h . Note that the univariate windows are *synchronized*, meaning that each column represents the view of the hypervisor h at the same time step for all the m metrics. We refer to it with $W(h)$.

$$W_{m,w}(h) = \begin{pmatrix} d_i^1 & d_{i+1}^1 & \cdots & d_{i+w-1}^1 \\ d_i^2 & d_{i+1}^2 & \cdots & d_{i+w-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ d_i^m & d_{i+1}^m & \cdots & d_{i+w-1}^m \end{pmatrix} \quad (2.4)$$

Note that: due to the discretization procedure it is possible to stack different metrics of the same hyperviosr even if they are produced with different frequencies or in slightly different timesteps, because after discretization each univariate window will have the same length.

If we consider a time window as a sample of our dataset, we go back to the original formulation (Definition 2.1), but instead of representing

a generic multidimensional point, our sample represents a Multivariate Temporal Window of a specific hypervisor.

In the next section we translate everything as a problem statement for our CERN specific data center scenario.

2.5.1 Anomaly Detection in the CERN Scenario

Our goal is to detect and identify the windows $W(h)$ of those hypervisors that are showing an anomalous behaviour. We define an instance window $W(h)$ anomalous if it is different enough from the windows observed in the past from that hypervisor and also from hypervisors in the same Openstack Cell. For this purpose we compare various anomaly detection approaches. To reach our goal we are given:

- A Group of n Hypervisors (Definition 2.2.2)

$$G = \{h_1, \dots, h_n\}$$

- Training Dataset: sequences of Multivariate Temporal Windows of length w contained in the history of those n Hypervisors with a total length of p discretized time steps.

$$V_{train} = \bigcup_{i \in CW} \bigcup_{h \in G} \{W(h)\}$$

where CW represents the set indices of all consecutive window segments of length w within p discretized time steps (where $p \gg w$).

- Testing Dataset: sequences of non-overlapping Multivariate Temporal Windows of length w referring to those n Hypervisors in a history of r discretized time steps (that come right after the p timesteps used for training). The Multivariate Temporal Windows

$$V_{test} = \bigcup_{i \in NOCW} \bigcup_{h \in G} \{W(h)\}$$

where $NOCW$ represents the set indices of non overlapping consecutive window segments of length w within r discretized time steps (where $r \gg w \wedge r = p$).

We are able now to define our objective:

Goal: $\forall W \in V_{test}$, assign an anomaly score that is monotonic with the level of anomalousness of that window. The notion of normality and anomalousness are learned from the past history. In addition the score should also let us discern with a simple threshold on it between anomalous and normal windows.

2.6 Assumptions

The assumption that are behind these problem formulations are:

- Hosts in the same OpenStack cells have similar characteristics that can legitimate us to think of them as a “swarm” therefore the deviation from a group behavior could be seen as a good way to spot possible anomalies;
- The discretization procedure of one time series with a mean aggregate every c minutes – where $c = \mathbf{10\ minutes}$ – is discarding high frequency components that are superfluous for the anomaly detection process;
- The selected window length w of analysis – $\mathbf{8\ hours}$ – contains all the meaningful information needed to identify anomalies, or in other words that lower frequencies components not captured by this windows are not relevant;
- The selected history length p – $\mathbf{1\ week}$ – is enough to learn a model of normality.

All the decisions about the length of temporal intervals considered were taken as result of iterative steps of data exploration and domain knowledge coming from CERN monitoring experts. In particular the window length of 8 hours and the granularity of 10 minutes for a single data point, we aimed at filtering out the high frequencies components to reduce the risk of raising false alarms. In addition the number of analysed metrics was fixed at the 11 most used by the domain experts during day to day operations and post-mortem analysis (Table 5.2), exploration on feature selection in such a complex unsupervised setting is left as future work.

Chapter 3

Background

In this chapter we introduce the background and concepts necessary to follow this research work. After an initial overview of the computing infrastructure scenario, we review both traditional outlier detection methods and Deep Learning approaches, and conclude with an overview of the ensemble techniques for Anomaly Detection.

3.1 Anomaly Detection for Computing Infrastructures

Nowadays, many applications are deployed on cloud computing platforms such as Amazon Web Services, Google Cloud Platform or Microsoft Azure. On such platforms, users expect a 24/7 availability of the services exposed. However, cloud systems, being specific software systems themselves, are subject to failures that can result in an anomalous behavior. Beside Cloud platforms and Data Center, every Large-scale and complex online service systems such as global commercial banks [12] and e-commerce such as Ebay [13] share similar characteristics.

Many of them are investing in and researching methods to cope with anomalies that can arise and undermine their operations. One of the closest use case to the CERN one is represented by Alibaba [14] that proposed Donut, a Variational Autoencoder approach, however, that is focusing only on a single metric and is limited to specific KPI (Key Performance Indicator) metrics. Other examples are represented by Microsoft and Facebook that proposed general methods to work with time series either in the direction of anomaly detection [15] or pure forecasting [16]. Nevertheless they address the single time series use

case, since it covers a large part of the considered application, but these works do not consider finding an anomaly in the correlation between multiple time series produced by the same entity, what in a Data Centre is represented by an hypervisor monitoring.

3.1.1 Challenges

The Data Center scenario is the perfect representation of a Time Series Anomaly Detection problem. Both Ebay [13] and Microsoft ([15]) identify the following challenges in the area of Time Series Anomaly Detection:

1. **Lack of labels.** Because of the low interpretability of a signal, it is reasonable to believe that it is expensive to find annotators for time series, e.g. compared to images. As a consequence, the datasets in the AD for Time Series domain are limited to relatively simple benchmark scenarios. Raising the need for the equivalent of the ImageNet dataset for the Time series field [17].
2. **Efficiency and scalability.** Given the large scale of the complex systems to monitor, it is required to monitor millions of time series at the same time, therefore the methods used in real industrial application cannot be compute-intensive during the inference step.
3. **Avoid alert spamming.** A complex system is monitored and managed by real humans that in the end have limited time and attention, so it is crucial to reduce the false positive rate as much as possible and raise the attention only on really crucial matters for the infrastructure.

The first point motivates the effort in putting in place an annotation procedure for anomaly detection, which we discuss in Chapter 4. The second point motivates the focus on relatively simple and computationally efficient methods rather than complicated ones, to respect the scalability requirement. The last point instead has been reinforced by [15] that envisioned as future work the use of an ensemble strategy to provide a more robust anomaly detection service.

Therefore every complex industrial system benefits from an efficient Anomaly Detection system, as the work of many entities in this field is witnessing. At the same time the alerting has to parsimoniously call for

human intervention to avoid spamming and losing trust in the service itself.

3.2 Algorithms for Anomaly Detection

In this literature review we present some of the main methods in Anomaly Detection including both traditional methods and Deep Learning ones; in particular our focus is on fully unsupervised techniques as motivated above. As mentioned in our Problem Formulation (Chapter 2), virtually any Anomaly Detection algorithm models or learns a scoring function that aims to assign a higher score to abnormal instances and a lower score to normal ones. Therefore, in the next part we present each method together with the way it computes its anomaly score. The results can then be either presented as a rank, starting from the most anomalous sample, or, as mentioned in the problem formulation, we can choose a threshold and highlight only the samples with a score higher than that.

3.3 Traditional Anomaly Detection Methods

We call *traditional* methods those invented before the Deep Learning era: they are generally based on a variety of different key ideas and assumptions. We present the most representative methods used in the literature from that period and also as baselines for current novel methods on Anomaly Detection: K-Nearest Neighbour (KNN) [18], Local Outlier Factor (LOF) [19], Principal Components Analysis (PCA) method [20], Isolation Forest [21] and One Class Support Vector Machine (OCSVM) [22].

3.3.1 K-Nearest Neighbour

The K-Nearest Neighbour (KNN) method [18] for Anomaly Detection is based on the following assumption:

Assumption: anomalies are samples that live in regions of the feature space *far* from the rest of the data, i.e. nearest neighbors have great distance.

This is related to the concept of density, therefore samples in areas of low density (absence of close neighbours) are considered anomalies. The closeness of the neighbours is assessed with the euclidean distance $d(\cdot, \cdot)$ in the multidimensional space. The final formula for computing the anomaly score is:

$$s_{KNN}(x) = \frac{\sum_{i=0}^k d(x, n_i)}{k} \quad (3.1)$$

where n_i is the i -th closest neighbour of x .

3.3.2 Local Outlier Factor

Local Outlier Factor (LOF) is probably the most known Outlier Detection method and the first to introduce the locality concept to spot outliers [19]. The main assumption is:

Assumption: anomalies are samples that live in regions of the feature space with low local density (i.e. local with respect to the closest neighbours).

It consists of these main steps:

1. Identify the k -nearest neighbours of every record.
2. compute the k -distance of every record as $k - distance(A)$ cooresponding to the distance of the object A to its k -th nearest neighbor. Usually the Euclidean distance is used.
3. Given the set of neighbours N_k , compute the Local Reachability Density (LRD).

$$LRD_k(x) = \left(\frac{\sum_{n \in N_k} dr_k(x, n)}{|N_k|} \right)^{-1} \quad (3.2)$$

where $dr_k(\cdot, \cdot)$ is the reachability distance defined as $dr(A, B) = \max(k - distance(B), d(A, B))$ and $d(\cdot, \cdot)$ is usually the Euclidean Distance.

4. Get the LOF anomaly score by comparing the LRD of the current record with those of its nearest neighbours.

$$s_{LOF}(x) = \frac{\sum_{n \in N_k} \frac{LRD_k(n)}{LRD_k(x)}}{|N_k|} \quad (3.3)$$

3.3.3 Principal Component Analysis for Anomaly Detection

The Principal Component Analysis (PCA) [20] is used to find a linear transformation of your data in such a way that they are projected on directions that maximize the total variance of your data. The procedure to find these directions is equivalent to finding the eigenvalues and eigenvectors of the covariance matrix of your data. The new directions of projection coincide with the eigenvectors just found. Moreover, if we sort the eigenvectors $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_p$ in increasing order of related eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_p$, we have that the eigenvectors with the highest eigenvalues explain most of the variance, whereas smaller eigenvalues signal direction of low variance in your data.

In our anomaly detection scenario, this means that for projected data onto the principal component y_p , assuming them to be normal, we observe high variance of the resulting transformed data, while if we project them onto y_1 they present a small variance, and a continuous behaviour with the eigenvalues in between. The main assumption for using PCA as an anomaly detection technique is:

Assumption: normal data exhibit a small variance if projected to the directions of the eigenvectors related to the smallest eigenvalues, while anomalous data show a high variance on those directions of low variance due to their abnormality.

Following this intuition, the anomaly score is defined as the sum of the deviations of every projected sample with respect to those directions. Following this idea [20] proposes the sum of squares of the standardized principal component scores:

$$s_{PCA}(x) = \sum_{i=0}^q \frac{y_i^2}{\lambda_i} \quad (3.4)$$

where $q < p$ aims at selecting only the small eigenvalues and $y_1, y_2, \dots, y_q$ represent the one-dimensional projections of record x onto the eigenvectors directions. These projections are also called principal components of the record x . Another possible approach, not considered in this work, is the use of PCA via the reconstruction error.

3.3.4 Isolation Forest

Isolation Forest (IFOR) was presented by [21]. This method is based on random partitions of the feature space with binary splits. These binary splits are repeated multiple times and represented by a binary trees data structure called iTree. The presence of more than one iTree configures IFOR as an ensemble technique. It relies on the main assumption that:

Assumption: anomalies are few and different from normal data and therefore can be likely isolated even few with random partitions.

During training time, IFOR follows the following steps:

1. Take a subsample X from the full dataset, with $|X| = n$;
2. Construct an iTree by randomly partitioning this sub-sampled dataset X with binary splits. An iTree is build by recursively selecting a random feature of your data and a random value to split the data in two branches. This process stops when either a predefined maximum height of the tree is reached or when the dataset cannot be split further because it contains only one element $|X_l| = 1$ at level l .

Repeat this multiple times and create a predefined number of iTrees, then at test time:

1. Pass each test instance to an iTree and starting from the root of the tree and the attributes of the instance follow the path according to the nodes' splits. Once it reaches a leaf, record the height (called path length);
2. Repeat this for every iTree in the ensemble;
3. Compute the average of the path length for that test instance;
4. Convert this into an Anomaly Score using the following formula:

$$s_{IFOR}(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \quad (3.5)$$

where:

- x is the test instance.
- $E(h(x))$ is the average value of path length in an ensemble of tree.
- $c(n) = 2 \cdot H(n - 1) - \left(\frac{2 \cdot (n-1)}{n}\right)$ where n is the number of instances in the dataset X and H is the Harmonic Number approximated via the following equation $H(i) = \ln(i) + \gamma$ (Euler's constant = 0.577215).

3.3.5 One-Class Support Vector Machine

As the name suggests, this strategy is a variant of the Support Vector Machine algorithm devised for classification [23, 24, 25]. The One-Class variant [22] aims at estimating the density of the normal data. The basic idea is to find a hyperplane in the feature space that separates the normal data points from the origin with maximum margin. In this case, the origin is taken as representative of the anomalous class. Although this method was initially used in a semi-supervised setting with only normal samples, it can also be used with the soft-margin variation [22] in a complete unsupervised scenario as we do in our case.

The quadratic program we solve is the following:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{\nu n} \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & \rho - \langle \phi_k(\mathbf{x}_i), \mathbf{w} \rangle \leq \xi_i, \xi_i \geq 0, \forall i \end{aligned} \quad (3.6)$$

Then the OCSVM anomaly score is computed via the signed distance between every point and the hyperplane.

$$s_{OCSVM}(x) = \text{sgn}(\langle \phi_k(\mathbf{x}_i), \mathbf{w} \rangle - \rho) \quad (3.7)$$

3.4 Deep Anomaly Detection Methods

Given the increasing adoption of Deep Learning in various fields, Anomaly Detection has recently also benefited from the application of Deep Neural Network techniques. A recent study [26], for instance, suggests that Deep Learning methods have the potential to surpass traditional ones in Network Intrusion Detection. Further motivation for Deep Learning techniques is given by two challenges in Anomaly Detection that can be tackled with deep methods:

- **need for large-scale anomaly detection:** the increased volume of data requires methods that can process them in a scalable way, regardless of their quantity. Unfortunately, many of the traditional methods, in their standard formulation, rely on the computation of distances that in a large dataset becomes computationally demanding.
- **complex data types:** real use cases for Anomaly Detection frequently include complex data types such as image or time series data. The automatic feature learning capability of Deep Learning techniques becomes extremely useful in such contexts to avoid the need for developing handcrafted features by domain experts.

3.4.1 AutoEncoder

In terms of techniques used, a recent survey on Deep Learning for Anomaly Detection [27] identifies AutoEncoders as one of the main methods in this field.

An AutoEncoder consists of a symmetric neural network structure composed by an encoder $e(\cdot) : R^D \rightarrow R^d$ that condenses the input $x \in R^D$ to a lower dimensional representation $e(x) \in R^d$ and a decoder $d(\cdot) : R^d \rightarrow R^D$ that is expanding the compressed representation into the output $\hat{x} = d(e(x))$. The objective is then to minimize the reconstruction loss, usually given by $\sum_{x \in Dataset} (x - \hat{x})^2$, therefore it is trying to learn the identity function. A crucial component of AEs is the bottleneck represented by $d \ll D$ where d is the compressed dimension from which the network has to learn how to reconstruct the input of size D . Therefore, it has to capture relevant high level features of the data in that reduced representation.

The AutoEncoder technique has been used mainly for feature extraction or for removing noise from the input data [28]. In the Anomaly Detection field, their use has been proposed by Hawkins et al. and later used also by Sakurada and Yairi. The underlying assumption for using this architecture for Anomaly Detection is that an AutoEncoder trained on mostly normal data will learn how to reconstruct them properly, whereas if we feed it with abnormal data, the reconstruction performance of the network on abnormal inputs will be inferior. The reconstruction error is therefore usually used as a measure for the degree of anomalousness of an input and to discriminate between normal

data and anomalies.

$$ReconstructionError(x) = \sum_{x \in Dataset} (x - d(e(x)))^2 \quad (3.8)$$

AutoEncoder Variants

Besides the typical characteristics of a generally symmetric structure and the bottleneck principle, the rest of the AutoEncoder architecture can be composed in various ways. In fact there exist different variants of AutoEncoder; depending on the main building block we can distinguish the following classes for our use case:

- Fully Connected Autoencoder [31]: it consists of a traditional multilayer perceptron architecture with a funnel structure leading to the bottleneck and a symmetric expansion to reconstruct the input.
- CNN Autoencoder [32]: it is based on the convolution operation that is typically used to work with image data, since its ability of using the information about the neighborhood of pixels. It has recently received lots of attention since its successful application to many high-dimensional and complex pattern recognition problems such as feature extraction [33].
- RNN/LSTM Autoencoder [34]: it is based on a neural network component that can not only take the output of previous neurons in the network as input but it keeps also information in the form of memory from the previous outputs it fired. This is crucial to work with sequence data as in Natural Language Processing or with Time Series inputs. A network including this type of components is called Recurrent Neural Network, and the most recent variant of this unit component are: Long-Short Term Memory (LSTM) [35] and Gate Recurrent Unit (GRU) [36].

3.5 Time Series specific Methods

From a time series perspective, we can see our metrics coming from the hypervisors as a stochastic process.

Definition 3.5.1. Stochastic Process: a collection of random variables indexed by some mathematical set. In the time series field this set is typically composed of natural numbers representing subsequent time instants.

The main difference between working with time series data and other kinds of data is that the subsequent time steps in a time series cannot be considered Independent and Identically Distributed random variables (*i.i.d.*). This is because previous values usually have some correlation with the future ones. This lack of independence between the variables at different time steps has been modeled and exploited extensively in the literature.

3.5.1 Forecasting Error: Anomaly Detection on Time Series

One way of using time series models for AD is to create a model that fits the normal data reasonably well, so that at test time if the real new measured data d_t at time t is considerably different from the prediction of our model $\hat{d}_t$, then we flag it as an anomaly. This is also known as *forecasting* or *prediction error* and it is obtained via a distance measure (e.g. euclidean distance) between d_t and $\hat{d}_t$. The assumption behind this kind of models is that a model that fits the majority of normal data will be a good predictor for situation of normality, whereas in abnormal scenarios the real data d_t will be very far from the expectation $\hat{d}_t$ of our model. In the literature we find both traditional and Deep Learning application of this principle [37] [38]; moreover we also mention a previous internal work at CERN that is based on it [39].

3.5.2 Traditional Time Series Models

In the traditional time series literature various model families have been proposed to model this dependency between subsequent timesteps:

- Autoregressive - AR(p):

$$d_t = \mu + \sum_{i=1}^p \beta_i d_{t-i} + \epsilon_t \quad (3.9)$$

where p is the order of the model that defines how many past timesteps of the original timeseries we consider for the prediction,

μ is a constant representing the mean of the series and ϵ_t is the error represented by white noise.

- Moving Average - MA(q):

$$d_t = \mu + \sum_{i=1}^q \theta_i \epsilon_{t-i} + \epsilon_t \quad (3.10)$$

where ϵ_i are white noise error terms and q is the order of the model that defines how many error terms we consider, μ is a constant representing the mean of the series and ϵ_t is the error represented by white noise. The parameters to learn are the β_i coefficients from $i = 1$ to $i = p$, where each of them represents how much a previous time step count in the prediction of the next time step value.

- Autoregressive-Moving-Average - ARMA (p, q):

$$d_t = \mu + \sum_{i=1}^p \beta_i d_{t-i} + \sum_{j=1}^q \theta_j \epsilon_{t-j} + \epsilon_t \quad (3.11)$$

that combines the two models in one, modelling both previous value of the time series and error terms at the same time. The parameters to learn are the θ_i coefficients from $i = 1$ to $i = q$, where each of them represents how much to weight the previous (observed) white noise error terms.

- Autoregressive Integrated Moving Average (ARIMA) where one or more initial differencing steps (“Integrated” part) are used to get a time series that has to certain degree the stationarity property¹. Then the same formula of the ARMA model is applied on the new differenced timeseries.
- Vector Autoregressive - VAR(p): it consists in an Autoregressive generalization designed for multivariate time series.

$$\vec{d}_t = \vec{\mu} + \sum_{i=1}^p A_i \vec{d}_{t-i} + \vec{\epsilon}_t \quad (3.12)$$

where A_i is a $m \times m$ matrix with m being the number of metrics and both $\vec{d}_i$, $\vec{\mu}$ and $\vec{\epsilon}_t$ are $m \times 1$ column vectors.

¹A time series is stationary if its properties do not depend on the time at which the series is observed. More precisely, if y_t is a stationary time series, then for all s , the distribution of $(y_t, \dots, y_{t+s})$ does not depend on t .

3.5.3 Deep Time Series Models

Deep Learning methods in this context make also use of the *forecasting error*. Given a sequence $d_{t-w}, \dots, d_{t-1}$ of the last w timesteps at time t of our time series pluss the most recent value d_t , we train a deep model to forecast the timestep d_t given the history $d_{t-w}, \dots, d_{t-1}$ as input. The predicted input $\hat{d}_t$ will then be compared with the real available data d_t and the difference between the two is considered as an indication of anomalousness.

We have considered two variants:

- CNN forecaster: we have a receptive field sliding on the time series just in the temporal dimension to aggregate information coming from the same time range of different metrics. [37] is representative for the most recent work in that direction, their method consists in a traditional CNN trained on the forecasting objective. However this work benchmark their method mainly on point anomalies, not windows.
- LSTM forecaster: we use the LSTM cell to learn the long term dependencies of a time series to predict the output. A similar work was already done at CERN in this direction by our colleague Ulrich [39]. However, an evaluation on a real dataset with labels is still missing.

3.6 Ensemble Methods in Anomaly Detection

Mapping all the methods to a specific problem formulation, therefore making them comparable in terms of input-output, is beneficial both for evaluating them in a fair way and also for exploring ways of combining them in an ensemble method. In [40], the authors identify the *diversity* of models as a requirement for the success of an ensemble method. They review different ways to introduce diversity in the ensemble:

1. Combining the same type of base detector. One technique is feature bagging where each detector uses a different subset of features; an example is [41]. Another alternative to introduce diversity among detectors consists in using a different subset of the dataset for training or a different parametrization, a prominent example of this type applied to time series outlier detection is

represented by [42] that uses an ensemble of RNN Autoencoders. Another way to achieve diversity among the same method can be done by introducing a random component during the model learning itself (e.g. isolation forest).

2. Combining different base detector methods and unifying their score as explained in [43].

The intuition behind combining different base detectors is that various modeling strategies capture different characteristics of the data, therefore bringing them together can be useful to mitigate their weaknesses and enhance their strengths, as has been recently observed by [44] in the context of Explainable AI.

Given the importance of the *diversity* among the single detector used in the ensemble, it is crucial to have a way to assess similarity between those detectors. Unfortunately, as already argued by [45], the AUC-ROC is a metric based on a single final number that does not tell anything about the similarities of different methods. Thus the same authors proposed the Kendall’s τ ([46]) correlation coefficient to compare the ordering of rankings.

3.6.1 How to practically combine models in an ensemble

Usually all the AD algorithms returns an Anomaly Score for each sample. This score represents in some way the degree of anomalousness of a certain sample, but unfortunately every algorithm has its own interval and scale of this score. Therefore, a first step is to merge the prediction of different detectors to make them comparable. Two natural approaches exist for this purpose:

- **Ranking-based:** for each detector, we order the samples by anomaly score, assign at each sample a position in a ranking system, in which the most anomalous is the 1st and the most normal is the last; then we combine the ranking of the various detectors.
- **Score-based:** for each detector we transform (e.g. translate and scale) the anomaly scores such that the anomaly scores produced by all detectors are similar in range; then we combine them with an arbitrary aggregation function (e.g. average).

The first approach can borrow techniques from the information retrieval field and some ways to merge the ranking of the various algorithms,

but it discards the absolute information on the degree of anomalousness. For example this causes a problem when we have only normal data, but according to the ranking there will always be a sample that is the most anomalous and if we adopt the approach of flagging as anomaly the first, this might be a false positive by being slightly different from normal data. Another observation in this regard is that if the algorithm is working well, we expect to have a large part of data with similar anomaly scores (i.e. normal data) and few of them with high anomaly score. We would also aim at having the two groups well separated, and we measure this accordance between the order and the ground truth with the ROC AUC.

Combination of ranking

Given the loss of information that we have by converting the score into a ranking, we mention just one of the most intuitive and popular strategy to combine rankings in an ensemble:

- **breadth-first** [41]: it constructs a final ranking merging the single ranking by taking the elements in the 1st position of every method, followed by the 2nd elements of every method, and so on, up to the last elements of the ranking. Given that the methods rank the same elements, we have duplicate copies of the same elements appear multiple times, therefore we drop them by keeping only the first copy of each element.

Although simple, this method has the limitation of being sensible to the order in which we iterate on the various methods; given n AD methods in our ensembles we have $n!$ possible orderings and correspondent final breadth-first results. Due to this characteristic the more the single rankings are different, the more diverse will be the $n!$ final ranking produced by the breadth-first strategy.

Combination of scores

Given the flexibility on aggregating the raw outlier scores, Kriegel et al. [43] propose a two steps process in that direction to make scores comparable:

1. **regularization**: to convert the score to a positive number, where close to zero means normal instance while far from it is an anomaly.

2. **normalization**: to scale the scores to a similar interval, so that the algorithms are comparable.
3. **probability conversion (optional)**: to obtain a probability, normalizing in a range $[0,1]$. This is useful if we want to convert the score in an absolute range that is also more interpretable by the expert. The challenge here is that if we train on some data and we set an anomaly score threshold value for which we consider a datapoint d to be an outlier with probability $P(d \text{ is outlier}) = 1$, then it become impossible to discern between the degree of anomalousness of test-time instances that are above that threshold.

After these steps, we can merge the actual normalized scores form the different methods. There exist different strategies that we consider in this work:

- **Max**: takes the max score that is assigned to a sample over the various methods; this is in favour of the most confident method.
- **Min**: intuitively give a high anomaly score only if all the methods agree on a high score.
- **Average of Maxima**: divides the methods in groups, takes the max of each group, and then computes the average among the various groups.
- **Simple average of the scores**: compute the mean of all the methods. $s_{avg}(x) = \frac{1}{|Methods|} \sum_{i \in Methods} s_i(x)$.
- **Cumulative sum**: every sample gets an anomaly score that is the sum of all the anomaly scores assigned to it by the various methods. $s_{cumsum}(x) = \sum_{i \in Methods} s_i(x)$
- **Convex Linear combination**: computes a weighted average of the single scores, where the weights are learned by fitting a linear regression using few labelled samples. By ensuring the convex property (i.e. all weights sum up to 1) we enforce the resulting anomaly score to have the same order of magnitude it had before the combination.

The authors [47] reviewed the averaging and maximization combination strategies. From their experimental section they conclude that averaging is a low risk-low reward approach that always reduce the

variance. On the other hand, they acknowledge that maximizing can sometimes deteriorate performance but in other cases is beneficial to emphasize extra outliers that would have passed unseen with an averaging approach. Nonetheless the scope of their work is limited to ensembles of base detectors of the same kind, focusing on LOF or KNN.

Chapter 4

Dataset Acquisition and Annotation

As is common in industrial settings, companies tend to have an incredibly large data lake but usually very little or no labeled data. As a matter of fact, this is also the case for CERN, where the time series data are available in large quantities and they are produced at an impressive rate but no curated dataset for the Anomaly Detection task with labels for evaluation has been prepared yet. With these premises, we have created an Annotation Pipeline and made it possible to collect an Anomaly Detection benchmark dataset on CERN data with the two following major objectives:

- supporting the evaluation of this scientific work with experimental results on the actual CERN data;
- giving CERN, as an Organization, the possibility to decide to invest in creating new datasets that can foster future work.

Among the two, the primary aim of this dataset is for benchmarking the Anomaly Detection methods under study in this thesis for the CERN use case; this provides a fast and objective way of judging the anomaly detection system compared to a qualitative feedback that often comes after months of extensive utilization of the system. The second objective is also important because it makes CERN users think about their monitoring infrastructure as a bi-directional interface that is not only exposing information, but also gives to the users the ability to input their knowledge in the form of anomaly annotations, that can function also as a reminder for future colleagues about past problems. We un-

derstand that this shift of approach requires time to be fully adopted by CERN users.

4.1 Need for a Dataset: Research Field Perspective

By reviewing a selection of surveys and publications in the field of anomaly detection for time series data [5, 48, 14, 49], we have noticed that every paper validates its new method on its own small restricted dataset. This has been recognized also by the authors of the USR data repository [50] as a crucial lack for the time series/signal processing community. Indeed, the scarcity of publicly available datasets is even more evident when compared to other fields like Computer Vision (ImageNet Dataset [17] and MVTec Dataset [51]) and Recommender Systems (The Netflix Prize Challenge [52]) in which the collection of huge and well created datasets led to a significant boost in the respective research communities. In this regard, we firmly believe that the work done for the dataset acquisition during this thesis is a first step to allow the creation of a bigger and more ambitious CERN dataset. Indeed also other world-recognized organization like NASA were able to achieve this in the past [53]. The ambitious aim is to release a dataset that can possibly lead to a less fragmented evaluation section in the various research papers and a more fair comparison among different methods with a hope to advance the research field, as well as solutions for the specific monitoring application.

4.2 Extension of Grafana Annotation

Among the technologies present in the Monitoring Infrastructure, Grafana is the platform of reference for the expert to visualize the Time Series data coming from the hypervisors.

“Grafana is a multi-platform open source analytics and interactive visualization web application.” Wikipedia

*“Grafana allows you to [...] create, explore, and share dashboards with your team and foster a data driven culture.”
Grafana.com*

Since we want to make it possible for the experts to input annotations in an easy and intuitive way, we select the Grafana platform that experts use on daily base for the annotation purpose. As a matter of fact, selecting a familiar tool like Grafana can possibly lead to a faster adoption of annotation practices by the CERN experts.

We can formalize our requirement as:

Requirement: the user should be able to insert “anomalous” and “normal” annotations when looking at the time series of an hypervisor and be able to trace back every annotation to its specific hypervirsor.

4.2.1 “Connecting the Dots”

Grafana is organized in dashboards that correspond to web pages where the user can visualise the data in a predefined way. For our particular CERN scenario, a single dashboard collects and visualises multiple time series data belonging to different hypervisors by means of a drop-down menu where the experts can select the hypervisor they want to inspect. In Grafana, the current value of this drop-down is called Template Variable ².

Feature 1: With a single dashboard the expert can visualise multiple hypervisors, once at the time, changing the values of the Template Variables.

After careful review of the documentation we found that Grafana provides a native support for point and interval annotation on time series data in the Grafana Annotation functionality ³. An annotation

²Grafana Template Variable: <https://grafana.com/docs/grafana/latest/variables/templates-and-variables/>

³Grafana Annotation functionality: <https://grafana.com/docs/grafana/latest/dashboards/annotations/>

consists of a start and end time of the interval plus an optional textual description and optional set of keyword that serve as tag for your annotation. Every annotation is dashboard specific and saved in the Grafana back-end database. We can therefore query Grafana to have all annotations related to a specific dashboard.

Feature 2: The expert can add interval annotations, that are dashboard specific. Moreover those annotations can be described by tags, and then based on those tags we can then query annotations.

With dashboard specific annotations, if we use the same dashboard to visualise multiple hypervisors, and we put an annotation while inspecting data of one of them, then the same annotation is visible also when inspecting other hypervisors. In addition, we will not have a way to retrieve all the annotations inserted while the specific hypervisor was visible.

A first possible solution consists in enforcing the user to manually insert tag with the name of the hypervisor it refers to for every annotation, but this is clearly just a first rudimentary solution to the problem, since we have the following drawbacks:

- tedious and time inefficient;
- error prone due to mistyping.

Naive Solution: enforce the expert to add the hypervisor’s name to every annotation by hand as tag (tedious and error prone).

Therefore we wanted to fill this gap and find an better solution to ease the work of the expert. Given these premises, we decide to extend the Grafana functionality to automatically append the hypervisor’s name as tag with a one-click procedure. We implemented this adding two buttons to the Annotation form (Figure 4.1) in order to automatically flag the annotation with a tag “anomaly” or “normal”. In addition the triggered JavaScript code has been extended to append all the template variables of the Grafana dashboard as tags in the Annotation database.



Figure 4.1: Modification introduced in the Grafana Annotation interface. Template Variables are automatically appended as tags.

Our solution: automatically appends info the current value of the template variables of the dashboard (including the hypervisor's name).

Having access to the open-source code of Grafana⁴ we could verify that the only needed change was on the client-side JavaScript, leaving the back-end written in GO untouched. The advantage of this purely client-side solution is that with a simple local override in the user web browser we are able to ship the functionality, without having to ask for an entire redeployment of Grafana at CERN. The core code modification is available in Appendix C.1, whereas the complete version is available in our data-analytics repository⁵.

Soon we realised that this modification could be of help in all those cases in which we want our annotation to depend not only on the dashboard but also on the Template Variables' values that modify the plotted data. Therefore we also started a discussion with the Grafana upstream community for the generalization of our extension for other use cases⁶.

⁴Official Grafana Repo: <https://github.com/grafana/grafana>

⁵Full code modification available at the following link: <http://go.web.cern.ch/go/Jjb6>

⁶Discussion upstream about our Grafana Annotation extension: <https://github.com/grafana/grafana/issues/24674>

4.3 Dataset Creation

Once the annotations are available in Grafana, we can extract them using the Grafana HTTP API ⁷, to create an offline, standalone labelled dataset. Since we leave the experts the freedom to decide which interval to annotate without forcing them to annotate always contiguous portions, we had to design how to convert the Grafana annotated intervals into anomaly labels. We evaluated two options:

1. label every data point in the Grafana annotated intervals;
2. divide the Grafana annotated intervals into chunks of a given time interval length.

Given that we partition the data stream into time windows in our problem formulation, we opt for the second option. Therefore we convert each Grafana annotation into window annotations of the predefined length of a time window, which generally is 8 hours long, and generates three non-overlapping intervals per day: 00:00 - 08:00, 08:00 - 16:00, 16:00 - 00:00. All the possible window annotation configurations and relative annotation resolution strategies are shown in Table 4.1.

4.4 Datasets Description

Given the presence of two clearly distinct user categories we also committed in creating two different benchmark datasets: one for a *Batch* group and one for a *Shared* group. Next we summarize the characteristics of the two labelled Datasets showing the resulting label distribution given the resolution strategy proposed in Table 4.1. We recall that these datasets are based on window lengths of 8 hours and each window is assigned a label among the following: anomalous, normal, mixed or empty.

4.4.1 Batch Group

For the *Batch* group, we annotated 25 hypervisors. The problems exhibited by the machines were related to high or low *CPU load*, *swap* misconfiguration or abnormal memory activity. The interval annotated

⁷Grafana Annotation HTTP API: https://grafana.com/docs/grafana/latest/http_api/annotations/

Window Annotation Case	Description	Resolution Strategy
Single and complete annotation	We have only one type of expert annotation (anomalous or normal) that covers the window entirely.	We label that window as anomalous or normal respectively.
Single but incomplete annotation	The window contains one or more annotated intervals of the same type and other portions that are not annotated.	We label that window with the type of annotation(s) present in that window.
No annotation	The window does not contain any annotated interval by the experts.	We label the window with a special placeholder (e.g. NaN) to represent that we do not have any annotation for it.
Multiple mixed type annotations	The window contains both anomalous and normal annotated intervals.	Regardless of their length, sequence of appearance and overlapping, we label the windows as <i>mix</i>

Table 4.1: Declarative description on how we convert the intervals inserted by the expert (with an arbitrary start and end) into window annotations with predefined start, end and length.

start from the 13th of February 2020 and end on the 11th August 2020, for a total interval of 180 days. In Figure 4.2 we present an overview of the various classes involved, whereas in the Appendix Figure A.1 we present a heatmap giving a visual overview of all the annotations for the *Batch* group scenario.

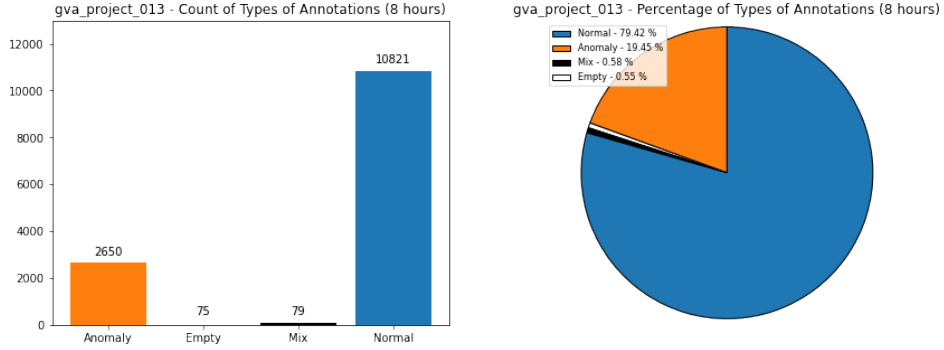


Figure 4.2: Statistics of Annotation Labels for Batch Openstack Cell: blue = normal, orange = anomaly, black = mixed, white = empty interval.

4.4.2 Shared Group

For the *Shared* Group we annotated 73 hypervisors. the problems showed by the machines were related to high or low *CPU load*, abnormal *Context Switch* activity or high *I/O memory* operations. The interval annotated starts from the 1st of January 2020 and ends on the 12th August 2020, for a total interval of 224 days. In Figure 4.3 we present an overview of the various classes involved, whereas in the Appendix Figure A.2 we present a heatmap giving a visual overview of all the annotations for the *Shared* group scenario.

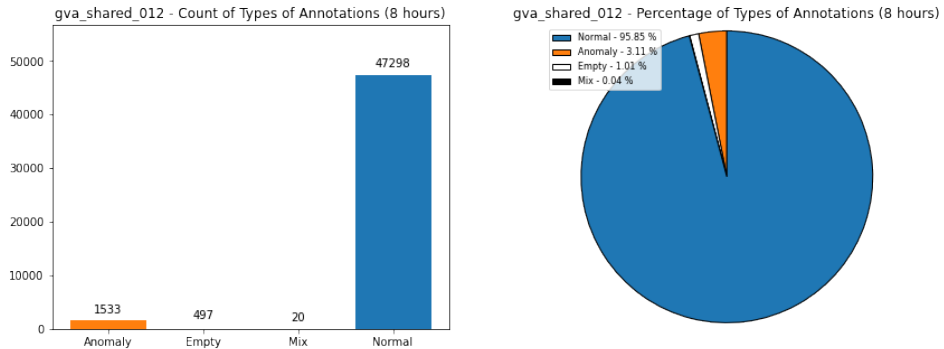


Figure 4.3: Statistics of Annotation Labels for Shared Openstack Cell: blue = normal, orange = anomaly, black = mixed, white = empty interval.

4.5 Artificial Setting Creation

Given the effort required to create the dataset of labelled anomalies, in the first stage we relied on an artificial anomaly setting for the

evaluation of the various methods. For this, we exploit the presence of two user categories: *Batch* hypervisors and *Shared* by user/service hypervisors.

The idea was to train the models only on historical data of *Batch* hypervisors and then test them on unseen future windows of data where we inject some hypervisors from the other *Shared* category.

The goal of the algorithm trained only on *Batch* hypervisors is to assign a high anomaly score to the hypervisors coming from the *Shared* category, as they are generated by another real world process, and a low anomaly score to the *Batch* hypervisor.

We took inspiration from the previous work of [54] where they identified some characteristics that a good AD benchmark should have:

1. **Normal data points should be drawn from a real-world generating process.** We achieved this by using real data coming from a different user category.
2. **The anomalous data points should also be from a real-world process that is semantically distinct from the process generating the normal points.** This is guaranteed by the substantial difference in the usage pattern of the two categories.
3. **Many benchmark datasets are needed.** This is ensured by the large quantity of data available and the virtually infinite combination of *Batch* and *Shared* dataset we can create.
4. **Benchmark datasets should be characterized in terms of well defined and meaningful problem dimensions that can be systematically varied.** This can be obtained by varying the input metrics, that can be up to 170 metrics with 2^{170} possible combinations. For reasons of complexity of the problem this option was not pursued, but alternatively we used the similarity between hypervisors of *Batch* and *Shared* cells as an indicator of difficulty.

In practice we also explored another way of controlling the difficulty of the task:

- **Similarity-based injection.** At test time we inject only *Shared* hypervisors that are *similar* to the training data, namely *Batch* hypervisors. This is obtained by means of dot product similarity

function between hypervisors' windows. Before computing the similarity we assure that every hypervisor vector is normalized with $norm(\vec{h}) = 1$ where $\vec{h}$ is the vector representing one hypervisor.

- **Standardization of metrics.** In order to make *Batch* and *Shared* more similar and hence increase the difficulty we normalized the input data of both with respect to their own mean and variance so to obtain vectors of metrics with zero mean and variance one. We also explored the option to change only the mean and keep original variance.

Given the availability of real CERN datasets, we have used this artificial setting mostly in an exploratory stage and the details about the procedure adopted are explained in the Section 7.3.

Chapter 5

Anomaly Detection Solution: System Engineering

Currently, the main use of the CERN monitoring infrastructure is for (i) real-time inspection of the data center status, (ii) threshold-based alarming on the infrastructure components, and (iii) post-mortem analysis of issues after notifications by CERN users. The adopted threshold-based alarm mechanism is in part embedded in Grafana, the CERN monitoring platform, and in part developed in a number of sensors running in each infrastructure entity. In order to increase the level of automation in Anomaly Detection and support the system managers in this complex task, a new flexible Anomaly Detection system based on Machine Learning approaches, is therefore proposed and prototyped for the CERN Cloud Infrastructure. In this Chapter we explain the main design choices and technologies behind the system, together with a detailed explanation of the data preparation procedure necessary for the Anomaly Detection methods discussed in the next Chapter.

5.1 System Design Challenges

Given the necessity for a Machine Learning Anomaly Detection System, the first part of my work at CERN consisted of putting in place a Proof-of-Concept system infrastructure. Machine Learning is possible without data, of course, and the challenges at CERN were several including:

- **Big Data processing:** the input data are stored in Hadoop Distributed File System (HDFS) folders and each of them occupy a

space in the order of Terabytes (Figure 5.1). Therefore we require a scalable approach to prepare this data for our algorithms.

- **Integration with the Monitoring Infrastructure:** the system must be able to communicate with the rest of the infrastructure already in place at CERN, not only in terms of input, but especially in terms of results produced, i.e. candidate anomalies discovered by the algorithms. The system must be able to publish them in the monitoring infrastructure so that the experts can inspect them and can take action.
- **Reproducible analysis:** the entire system must keep track of the operations executed on the ingested data from the start to the end of the pipeline. In this way a candidate anomaly has a clear data provenance and the computation that led to it can be reproduced.

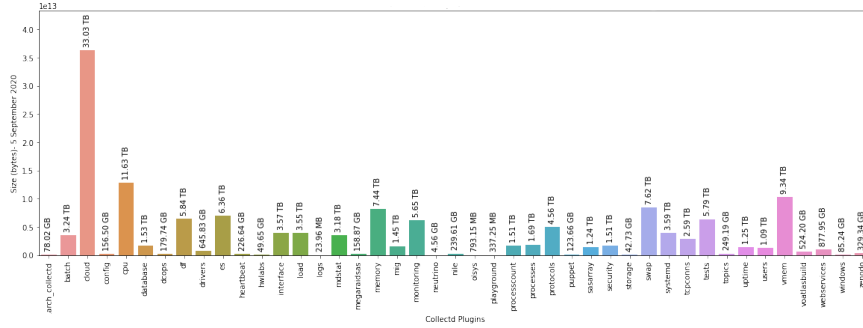


Figure 5.1: Space Occupied by Collectd Plugins. We report the total space occupied by every Collectd Plugins in HDFS. Every plugin is typically containing more than one metric and for all the hypervisors at CERN since 2018.

5.2 Core Technologies and their Scope

To face these challenges we rely on the following core technologies and solutions:

- **Apache Spark for the Data Preparation:** we employ the CERN Spark cluster to read data from HDFS and pre-process them.

- **ElasticSearch as document storage:** we designed a document format in JSON to store all the information describing a candidate anomalies produced by our system. Details about this format can be found in Appendix C.2.
- **ElasticSearch and Grafana for publishing candidate anomalies:** we design the above mentioned document format also to be compatible with the visualization platform Grafana, so that the candidate anomalies found by the algorithms are directly visible as annotated intervals in every dashboard that displays the anomalous time series.
- **Grafana Extension for improved interactivity:** we design and implement an extension to the Grafana platform (Section 4.2) to let CERN colleagues insert their annotations directly on top of the time series they see via the monitoring infrastructure.
- **Apache Airflow to automate the result production:** we use Airflow to code the tasks performed by the system in a Direct Acyclic Graph (DAG). In this way every step is well documented and it is possible to reproduce the analysis or to automate it by running it periodically on different data as input.

5.3 Anomaly Detection Pipeline

The overall system is represented in Figure 5.2 and reaches its goals by implementing a process composed of the following steps:

1. Read the data from the Hadoop Distributed File System (HDFS), where the metrics of every hypervisor are safely stored by the monitoring infrastructure (code in Appendix C.3.1);
2. Prepare the data by filtering the group of hypervisors under analysis, aggregating the data via the mean function into bins of x minutes for creating Discretized Time Series of every metric (Definition 2.5.1) and creating the Multivariate Temporal Window (2.5.3);
3. Analyze the time series window with the trained Anomaly Detection algorithms;

4. Aggregate the predictions of the single algorithms (optional) via an ensemble layer to increase the confidence on every prediction and reduce false alarms as much as possible;
5. Push the candidate anomalies to ElasticSearch storage and visualise them in Grafana;

At the end of the chain the expert can also annotate normal or anomalous data directly on the Monitoring Dashboard in a faster and easier way thanks to the proposed modification to Grafana (Section 4.2). Moreover, the annotations collected from Grafana can be quickly extracted to create annotated datasets, as those proposed in Section 4.4 by us.

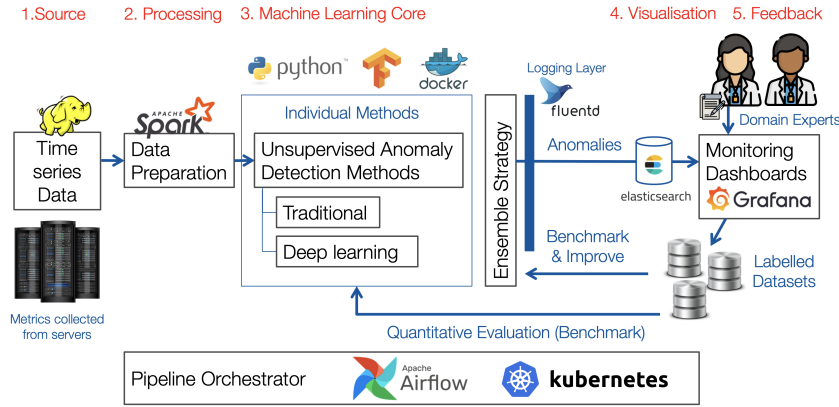


Figure 5.2: Anomaly Detection Pipeline with Expert feedback. Note that the experts can see the result of the algorithms directly in the Monitoring infrastructure via the used dashboard and interact for an easy and fast labelling procedure.

In the context of Figure 5.2 with Feedback loop we refer to the possibility of using the small annotated dataset for other purposes than the mere benchmarking of the algorithms, for example, this dataset can be used to develop a supervised ensemble logic layer to combine prediction from multiple algorithms or for exploring work in the field of Active Learning, that we leave as future work.

5.4 Data Preparation

The *Extraction-Transformation-Load* (ETL) procedure summarized with the Apache Spark block in Figure 5.2 incorporates multiple steps and it

includes important design decisions heavily dependent on the specific algorithms that we use in the Data Analytics core presented in Chapter 6. Before starting with a careful description of every step, we summarize in Table 5.1 all the temporal scales of the analysis chosen in Section 2.6. In Table 5.2 we summarize the names of the metrics used together with the *Collectd* plugin that is responsible for their production.

Variable	Description	Chosen Value
w	Window Length	48 (nr timesteps)
c	Temporal Length of one timestep summarized in a single data point.	10 (minutes)
m	Number of metrics used in a window of analysis.	11 (metrics)

Table 5.1: Temporal Scales of Analysis. We refer to the same nomenclature of Section 2.6

Collectd Plugin	Metric's name
CPU	cpu_percent_idle
CPU	cpu_percent_system
CPU	cpu_percent_user
Interface	interface__if_octets_tx
Interface	interface__if_octets_rx
Load	load_longterm
Memory	memory__memory_free
Swap	swap_swapfile_swap_free
Swap	swap_swapfile_swap_used
VMEM	vmem__vmpage_io_memory_in
VMEM	vmem__vmpage_io_memory_out

Table 5.2: Selected Metrics and their Collectd plugin of origin.

5.4.1 Discretization of Time Series

The rate at which every metric is produced by a hypervisor and goes into the monitoring infrastructure is unpredictable because of network or processing delays. In addition, different configurations for different sensors are directly responsible for different production rates of the respective time series. Therefore, in these cases we can have an

unbalanced situation in which there are many more readings for a sensor with respect to the others. As mentioned in our modelling part (Section 2.5), we decide to convert every time series into a *Discretized Time Series* (Definition 2.5.1). In practice we aggregate our data per machine and per sensor in non overlapping intervals of 10 minutes and we keep the statistical mean of the sensor’s reading in that temporal range as summary. The implementation has been done in a distributed way using the Spark DataFrame API and is available in the Appendix C.3.2.

5.4.2 Input Normalization

The metrics collected by the hypervisor’s sensors carry specific information such as memory available, CPU usage, disk operations, or network activity. Each of them represents a different quantity that might be a percentage, a real or discrete value, and most importantly they have typically many order of magnitudes of difference, for example *CPU load* represents a percentage (between 0 and 100) whereas *SWAP space free* is in the order of Gigabytes, namely 10^9 . This difference in scale of the various metrics is harmful for most anomaly detection algorithms since they will erroneously give weight to the metrics with higher value, in particular those that use distance measures like LOF and KNN. We therefore normalize the data based on our training set and separately for every metrics. In particular we standardize by removing the mean and dividing by the standard deviation:

$$x_{normalized} = \frac{x - \mu_m}{\sqrt{\sigma_m}} \quad (5.1)$$

where m is a hypervisor metric and μ_m , σ_m are respectively the mean and the standard deviation of that metric computed on the training set (one week of data). The implementation of this step is available in the Appendix C.3.3.

5.4.3 Minimal Unit of Analysis

Every algorithm is designed to take a single Multivariate Temporal Window (Definition 2.5.3) as an input one at a time and produce an anomaly score; this window refers to a specific hypervisor and a specific time interval covered by the window itself. Given the differences between traditional Machine Learning methods and Deep Learning ones,

we adapt the input based on the methods. In particular, the vast majority of the traditional methods (Section 3.3) are not able to model the temporal dimension off the shelf, therefore we propose two distinct input data formats:

- *greymap* representation: that corresponds to the matrix representation described in Definition 2.5.3. Here the temporal dimension is preserved in one axis of the matrix.
- *vectorized* representation: where all the different metrics and their timesteps given in a window are flattened and each timestamp of each metric becomes a simple feature of our datapoint.

In particular we use the *greymap* representation for algorithms that can model the temporal dimension (AECNN, AELSTM, FORVAR, FORCNN) and the *vectorized* representation for those not able to model it (OCSVM, LOF, KNN, PCA, IFOR and AEFC). We also refer to methods taking the *vectorized* representation as *static* anomaly detection methods because they are suitable for all kind of dataset with features without taking into account in any way the temporal information.

In both cases the input contains the same amount of data. It consists of a temporal window of data containing w time steps in the past and m time series related to the same hypervisor. The window length is $w = 48$ where every time step summarizes 10 minutes of data with the mean. In total, every window represents the last 8 hours of metrics for a specific hypervisor as described in the Problem Formulation with the concept of Multivariate Temporal Window (Definition 2.5.3).

5.4.4 Greymap for TimeSeries AD Algorithms

The *greymap* representation corresponds to what we think as the most natural representation of a time window, that is the matrix representation described in our Problem Formulation if Multivariate Temporal Window (Definition 2.5.3). Each row contains the values of a single metric in that window of time and therefore the columns refers to subsequent time steps of the sensors' values. Here we report the version given by Definition 2.5.1 indexed with the specific number of metrics

($m=11$) and timestamps ($w = 48$) we are considering in our pipeline:

$$greymap_i = \begin{pmatrix} d_i^1 & d_{i+1}^1 & \cdots & d_{i+47}^1 \\ d_i^2 & d_{i+1}^2 & \cdots & d_{i+47}^2 \\ \vdots & \vdots & \ddots & \vdots \\ d_i^{11} & d_{i+1}^{11} & \cdots & d_{i+47}^{11} \end{pmatrix} \quad (5.2)$$

The name *greymap* comes from the visual representation we adopt to inspect data with a heatmap plot (Figure 5.3).

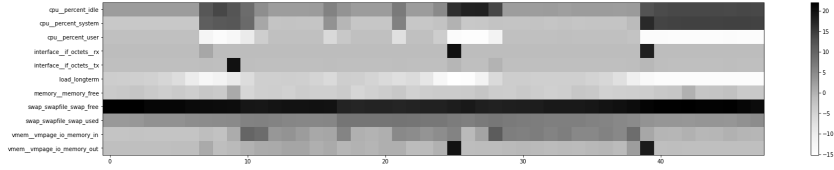


Figure 5.3: *Greymap* representation. It consists in a heatmap that represents the timeseries input for a single hypervisor, each row is the timeseries of one of the many timeseries M metrics considered in this multivariate setting (in this case, $M = 11$). Each column represents the past time steps in a window of length w (in this case, $w = 48$).

5.4.5 Vectorization for Static AD Algorithms

For methods that use vectorial inputs we discard the temporal information and condense all the data from different metrics (i.e. rows) into a single vector. We call this procedure *vectorization* and the resulting encoding a *vectorized* encoding of our window. By applying this process, we accept to lose some information since we do not have anymore a direct representation of time in this representation. The final formalization of this vector is the following:

$$vectorized_i = [d_i^1, d_{i+1}^1, \cdots, d_{i+47}^1, \cdots, d_i^{11}, d_{i+1}^{11}, \cdots, d_{i+47}^{11}] \quad (5.3)$$

5.5 Production Implementation with Apache Airflow

We adopt Apache Airflow⁸ as the main orchestrator for all the steps of our pipeline. It is an open-source workflow management platform

⁸Apache Airflow Project Description: <https://airflow.apache.org/docs/stable/project.html>

initiated by Airbnb in October 2014, where a workflow can be defined as a Directed Acyclic Graph (DAG). Each node of the DAG represents a step in the pipeline and is implemented with an *Airflow operator*⁹. In Figure 5.4 we see our implementation and the Bash Operator node used to schedule bash commands. Moreover Airflow can handle also repetitive analysis scenarios, where a predefined number of tasks has to be repeated every fixed amount of time. In particular for our experimental part we repeated the analysis every week, visible in Figure 5.5. The steps encoded are the following:

1. prepare the configuration file containing the information about which group of hypervisors we want to monitor, metrics we want to consider and interval for our training set and testing set;
2. download the training data and normalize them by computing and storing the normalization coefficients for every metric;
3. download the testing data and normalize them with the normalization computed on the train set;
4. run the analysis on the downloaded data, (with training on the train set and testing on the test set);

Note that every step, apart from the configuration files' preparation is run in a Docker container environment to have a precise definition of the libraries necessary in every step. In addition the last step has two possible optional outputs for the anomaly score, they can either go to the real monitoring infrastructure via a Fluentd¹⁰ logging driver attached to the container or can be saved in a local file for benchmark production.

⁹Airflow Operators: https://airflow.apache.org/docs/stable/_api/airflow/operators/index.html

¹⁰Fluentd is an open-source data collector for unified logging layer. More info can be found here: <https://docs.fluentd.org/>

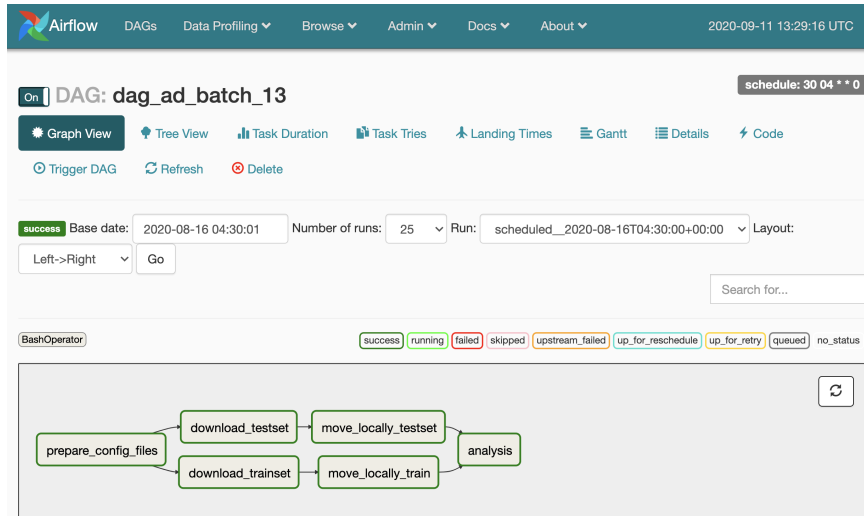


Figure 5.4: Airflow Production DAG. Direct Acyclic Graph representing the task done for analysing one week on data with the various algorithms. Note that the step of data preparation is split in two sub steps (download_*, move_locally_*), because the aggregation happens only in Apache Spark given the large volume of data processed and only then the data is moved to the local machine for analysis.

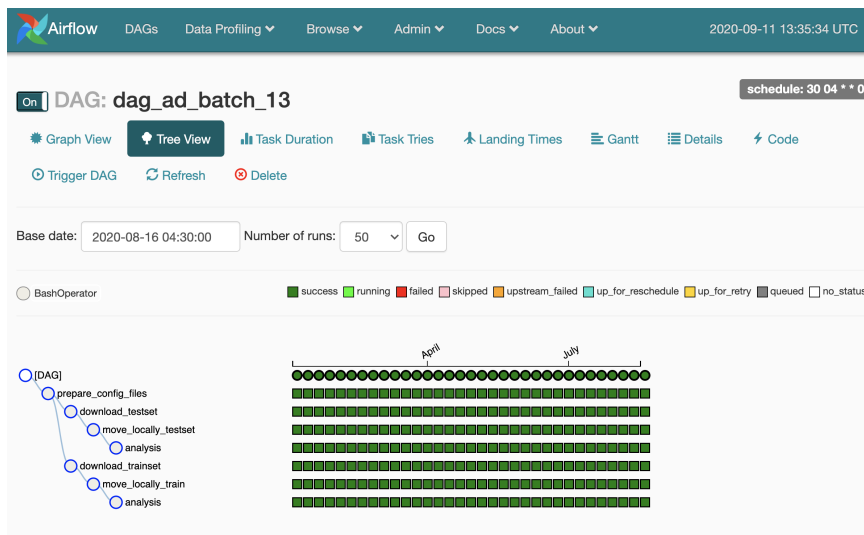


Figure 5.5: Airflow Recurrent Analysis. Recurrent analysis happening for every week of data in a Batch group of hypervisors.

Chapter 6

Anomaly Detection Solution: Data Analytics Core

The main component of the system described in Chapter 5 is the Data Analytics core that contains the actual algorithms and methods to discover anomalies. We select both traditional anomaly detection (AD) methods and Deep ones. For each of these categories we also propose one method that is specific for time series. They are summarized in Table 6.1 and they are divided into two families.

Traditional AD	Deep Learning Based
One Class SVM (OCSVM)	AutoEncoder Fully Connected (AEFC)
Local Outlier Factor (LOF)	AutoEncoder with CNN (AECNN)
Isolation Forest (IFOR)	AutoEncoder with LSTM (AELSTM)
Principal Component Analysis (PCA)	Forecaster based on CNN (FORCNN) - <i>Time Series Specific</i>
K-Nearest Neighbors (KNN)	
Forecaster based on Vector Autoregression (FORVAR) - <i>Time Series Specific</i>	

Table 6.1: Summary of methods under study divided into the two main categories: Machine Learning and Deep Learning. In addition, the last row refers to methods specific for Time Series Anomaly Detection.

The idea of including traditional Machine Learning methods is also motivated by the presence of a vast amount of literature on Outlier De-

tection methods for non-temporal data. Testing their performance and suitability for the CERN use case is crucial to justify the usage of more powerful but computationally more intensive methods. From the implementation point of view, since CERN requires solid software tools for a production environment, we select the reliable and well tested PyOD library ¹¹ that is also attracting interest of both industry and academia [55].

6.1 Train and Test Set

For our training and test dataset we decided to work on weeks of data with our choices summarized in the Table 6.2, with one week for training and one week for testing. This procedure is then repeated for every couple of consecutive weeks of the two curated CERN datasets. During train we fit our models with data in an unsupervised way. Deep Autoencoder models learn their reconstruction objective: AEFC, AECNN, AELSTM. Forecasting methods fit the data to get a low prediction error. PCA learns the projection, KNN and LOF keep the samples in memory for comparison at test time. Moreover we also store the mean and standard deviation of every input metric data, that we then use to normalize all the training data of the train week. We also save the mean and standard deviation of the anomaly scores computed on the training week.

At test time we normalize the data metrics of the test week using the mean and standard deviation from the week of training, so that the metrics are in a similar range. We then apply the algorithms and get an anomaly score for every window in the test week.

6.2 Anomaly Score Normalization

An important design choice common to all algorithms is the Anomaly score normalization. We normalize the test anomaly scores with the mean and standard deviation of scores obtained on the train week. Here we assume that this statistical quantities remain relatively stable from one week to the next one, in particular for the normal instances. We do this step to guarantee that all the methods get more or less the

¹¹Python Outlier Detection <https://github.com/yzhao062/pyod>

Variable	Description	Chosen Value
$tmp(X_{train})$	Temporal length covered by the training dataset.	1 (week from Sunday at 00:00 to Saturday at 23.59)
$tmp(X_{test})$	Temporal length of covered by the testing dataset. Note that the time of testing is always directly temporally contiguous to the training dataset.	1 (week from Sunday at 00:00 to Saturday at 23.59)

Table 6.2: *Temporal Scales of Analysis. We refer to the same nomenclature of the Problem Formulation*

same range of anomaly scores and the comparison between them is on the same scale. This is also beneficial for the ensemble strategies that will combine similar quantities.

6.3 Novel Time Series Methods

Beside the adaptation of traditional outlier detection methods we propose also an adaptation of forecasting methods introduced in Section 3.5.1 to our Problem Formulation. They are usually performant in detecting *point anomalies*, namely when a single point is anomalous in the entire time series. Nonetheless in our formulation we are aiming at labeling regions (i.e. windows) of the time series that are showing an unexpected behavior. Therefore we adapt them by aggregating the anomaly score for each point in the window in a single anomaly score using the summation of prediction errors on different time steps:

$$s_{forecast}(W_{m,w}) = \sum_{i=1}^w \sum_{j=1}^m (d_i^j - \hat{d}_i^j)^2 \quad (6.1)$$

where i is the temporal index among the w timesteps of the window $W_{m,w}$, j refers to the m metrics of our multivariate scenario and $\hat{d}_i^j$ is the prediction of the model produced from the historical data preceding the i^{th} timestamp.

6.3.1 Forecasting with VAR

Among the traditional Time Series models introduced in 3.5.2, we choose to use the Vector Autoregressive (VAR) model for our CERN use case, since it works on the multi-time series scenario and the prediction of a metric's value can be dependent on the past values of other metrics too. One limitation of the VAR model is that it can only learn from complete timeseries belonging to the same hypervisor. It is not trivial to decide how to learn parameters from windows belonging to different hypervisors as well. Our solution consists in creating an ensemble of models, each of them learned on a different single hypervisor and to contain the computational cost, we only learn on a random subset of the hypervisors in the group. Then at test time we use these single models together to assess the abnormality of all the hypervisors in the groups, even those hypervisors for which we did not learn a specific model. In this way we assume that the selected subset of machines is a good representation of the normal behaviour and we expect it to generalize well also on new unseen hypervisors.

The order of a single model is set once and for all the models in the ensemble to $p = 2$ by minimizing the Akaike information criterion (AIC), that is a measure of quality of a predictor that rewards the goodness of fit but considers also the complexity of the model in terms of number of parameters to estimate. This is the formula:

$$AIC = 2k - 2 \ln(L) \quad (6.2)$$

where k is the number of parameters and $\hat{L}$ is the likelihood of the model on the data.

Given an ensemble model composed of $c = 20$ individual models learned on the same number of hypervisor, we have two levels of aggregation for the creation of the anomaly score:

- **window-level:** each single VAR model is given 2 timesteps of history in our window and predicts the next timestep. We then compute the euclidean distance between prediction and the actual data to find an error measure. Since our window is $W = 48$ timesteps long we repeat this for 46 times. The first aggregation level consists in summing all the errors of the various timestep to create an overall anomaly score for this window.
- **models-level:** the presence of $c = 20$ models will give us c different window-level anomaly score that we aggregate by taking the

mean

The overall formula is the following:

$$s_{VAR}(W_{m,w}) = \frac{1}{c} \sum_{i=1}^c \sum_{j=p}^w (VAR_model_i(W_{m,w}(j-p, j)) - W_{m,w}(j))^2 \quad (6.3)$$

where c is the number of models in the ensemble, p the order of the VAR model, $VAR_model(\cdot)$ is the prediction of one model, $W_{m,w}(a, b)$ is the Multivariate Temporal Window matrix (Definition 2.5.3) from column in position a to position b excluded and $W_{m,w}(a)$ is just the column at position a of that windows.

6.3.2 Forecasting with CNN

We employed a model inspired by the LeNet [56], with few CNN layers followed by a funnel of densely connected layers. In a given window with W timesteps and m time series we represented it as an image, that we defined *greymap* in Section 5.4.4.

The convolution operation has a certain receptive field of dimension $3 \times m$, so that every convolution operation can take information from all the other m time series as well from the neighbouring time steps.

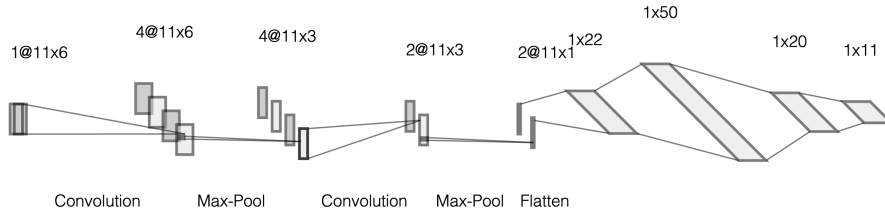


Figure 6.1: CNN Forecaster architecture

Online Anomaly Detection methods that focus on point anomalies would use the entire window of $w - 1$ timesteps to forecast the current timestep with the CNN model, but since we are interested in anomalies appearing in the entire next window we adopted a different approach. Instead we trained the Neural Network model by feeding a shorter segment of the image with H consecutive timesteps (where $H < w$) and then predict the next timestep after the shorter chunk of length

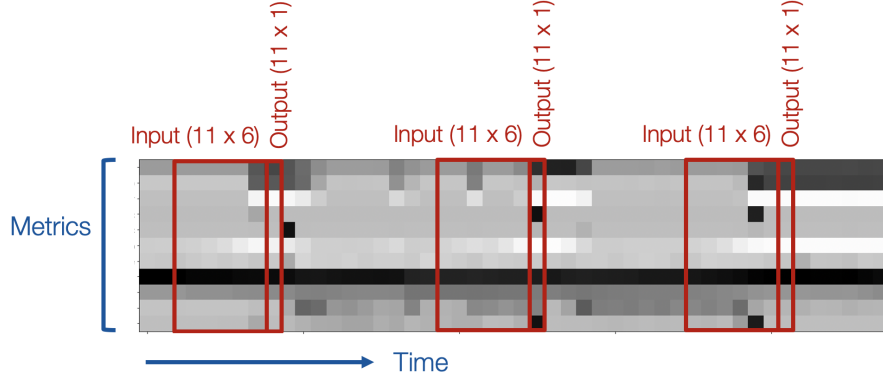


Figure 6.2: Sliding predictions that can be done on a single greymap. Here we show 3 of the total 42 pairs of Input-Output squares that are used to produce the reconstruction error.

H . The model prediction is then compared with the real value to compute the prediction error. Then an overall anomaly score is created by summing all the prediction errors in the window. Hence, we apply the same approach used for point anomalies but replicated for every timestep of the window. For a typical example where we have $w = 48$, a reasonable value would be $H = 6$.

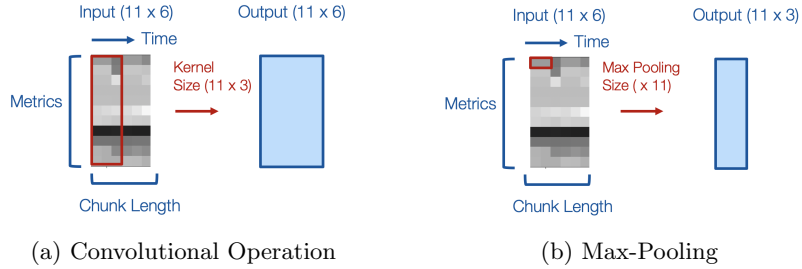


Figure 6.3: Details about the convolutional and max-pooling operations used in the proposed network.

6.4 Autoencoders for Time Series Modelling

As described in the Background 3.4.1 the most prominent Deep Anomaly Detection method is the Autoencoder, therefore, taking inspiration from previous literature [31, 32, 34, 37], we propose standard versions of these Autoencoder variants.

6.4.1 Fully Connected Autoencoder

We propose a fully connected neural network architecture with 1 input layer to take our *vectorized* representation as input, 3 hidden layers to implement the bottleneck principle and 1 output layer to produce another vector with the same dimension of the input. A indicative representation is given in Figure 6.4 We adapt the PyOD implementation

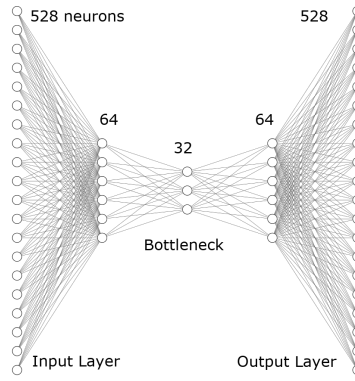


Figure 6.4: Fully Connected Autoencoder architecture. It is made of five dense layer of 528, 64, 32, 64, 528 neurons respectively. Note that the number of neurons is just smaller to avoid cluttering the diagrams with edges.

of this method to work with the latest version of TensorFlow 2.1.

6.4.2 CNN Autoencoder

We propose an architecture made of an alternation of convolution and max pooling operations for the encoder part as well as deconvolution and up-sampling for the symmetric decoder side. As anticipated, the input of this method is the *greymap* representation that serves as an image in our case, but contrary to usual kernels used for traditional image processing we designed our convolutional kernel to dimensions able to cover all the metrics when sliding in the temporal dimension. Moreover, for the input reduction with max pooling operation we reduce the dimension of the area span by it so that it summarizes with the max only pixel from the same metrics. We avoid to put more layers to keep the training relatively fast. Figure 6.5 shows the details of the architecture.

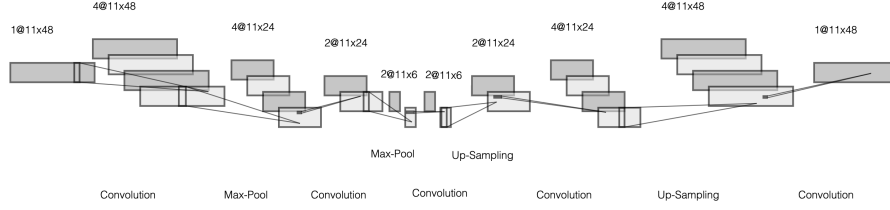


Figure 6.5: CNN Autoencoder architecture

6.4.3 LSTM Autoencoder

We design an architecture made of two LSTM layers with hidden state of dimension 10. We keep this representation relatively low to ease the training and also because we want to enforce the bottleneck principle in this way. Figure 6.6 shows the overall architecture and also the hidden states movement.

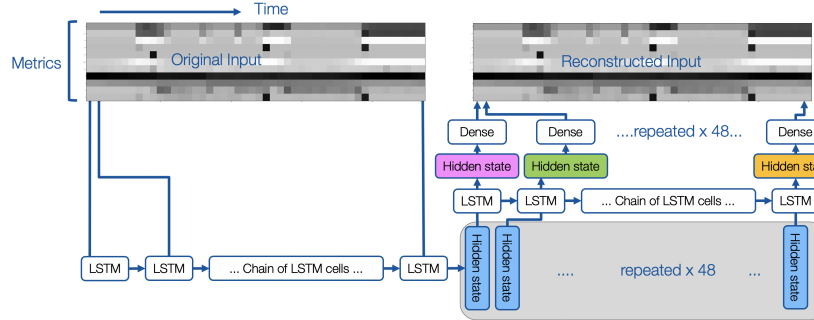


Figure 6.6: LSTM Autoencoder architecture. Note that the blue hidden state is just the last hidden state of the pass in the first layer and it is repeated to match the number of timesteps (48), whereas the second LSTM layer is producing different hidden layers and from each of them we produce a timestamp in the output image (that corresponds to a column). Note that the reconstructed image shown as output is the ideal situation and it has not been produced by our algorithm in the real use case.

6.5 Ensemble

Thanks to the normalization of anomaly scores explained in Section 6.2, we explore a way to combine the prediction produced by the different models. Indeed this normalization step corresponds to the one described by [43] and introduced in Section 3.6.1; it converts all the

anomaly scores in the same intervals so that they are comparable in size. We explore some of the aggregation techniques introduced in the Section 3.6.1: max, min, median and average.

6.5.1 Logistic Regression Ensemble

In addition, we propose an additional combination technique that makes use of some annotated labels. It consists in fitting a logistic regression classifier that learns how to combines the scores from different methods using some labelled data:

$$s_{ENS-LOGREG}(W_{m,w}) = \frac{1}{1 + e^{c_0 + \sum_{i=1}^{m=|\text{Methods}|} c_m \cdot s_m(W_{m,w})}} \quad (6.4)$$

where c_i is the weight learned by the Logistic Regression and assigned to method m , whereas $s_m(W_{m,w})$ is the anomaly score assigned by method m to the window $W_{m,w}$. In particular, beside being a supervised technique, this method can be seen as higher level learning procedure since we learn how to combine predictions made by other learners.

Note that we use 10 days of the two curated datasets of *Batch* and *Shared* hypervisors to learn the regression coefficients and then remove these windows of data from the computation of the performance metrics.

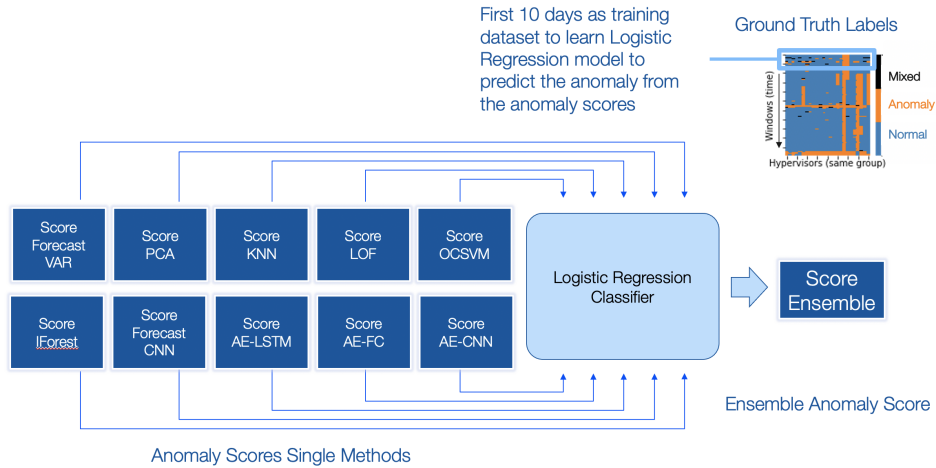


Figure 6.7: *Logistic Regression Ensemble*. We can see the various anomaly score of the single algorithms are used to predict the anomaly label with a logistic regression classifier. The model is learned on the first 10 days of data that are then excluded from test.

Chapter 7

Experimental Evaluation

Given the interest in the CERN use case the focus is on assessing the performance on CERN data. We identify two scenarios:

1. **Experiment A: Two Curated Annotated Dataset from CERN:** collected with the new functionalities and user interface introduced by our extension to the Grafana CERN monitoring dashboard. They are collected by CERN domain experts on one *Batch* Openstack Cell and one *Shared* Openstack Cell as described in Chapter 4.
2. **Experiment B: Artificial Anomalies:** following the directives of [54] we combine time series coming from two different real world processes representing the two user categories introduced in the Problem Formulation. The *Batch* time series are considered as normal data, whereas the more irregular and diverse *Shared* time series are considered anomalous. For details, we refer to the description given in 4.5.

Another option that exists in the literature [49] is the *Server Machine Dataset (SMD)*, but it is not compliant with our problem formulation since it has been designed for point anomaly detection.

7.1 Performance measures

Every method in our study produces an anomaly score, therefore by applying a threshold as explained in 2.4 is evaluated with the same performance metrics of a binary classifier, where the positive class is represented by label 1 and corresponds to an anomalous instance, whereas

the negative one corresponds to label 0 and means normal instance. We therefore propose the use of the following four quantities:

- True Positives (TP) or True Alarms: number of anomalous hypervisors that the algorithm correctly identifies as anomalies.
- True Negatives (TN): number of healthy hypervisors that the algorithm correctly identifies as normal.
- False Positives (FP) or False Alarms: number of healthy hypervisors that the algorithm *wrongly* identifies as anomalies.
- False Negatives (FN): number of anomalous hypervisors that the algorithm *wrongly* identifies as normal.

Those four situations are summarized in Figure 7.1. With these, we build the derived evaluation measures described in the sub sections below.

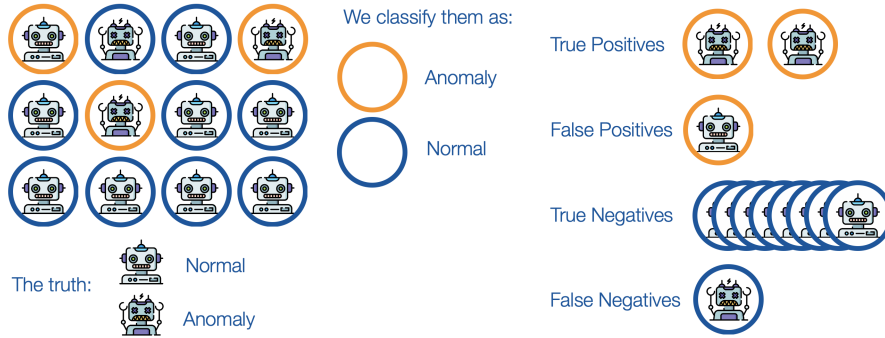


Figure 7.1: Main performance measures to evaluate a binary classifier. On the left we see a group of machines represented by robot icons, it can be either normal or anomalous. Then the binary detectors can declare it as normal (blue circle) or anomalous (orange circle). On the right we have the four possible situations where the algorithm is good at spotting anomalies (True Positives) or normal data (True Negative) and where it is bad, spotting false anomalies (False Positive) or false normal data (False Negative).

7.1.1 Receiver Operating Characteristic

In this alarming context the most relevant measure for assessing the performance of an Anomaly Detection method is the False Positive rate since the experts do not have enough time to cope with many False Alarms given the limited amount of time. Comparing also to

similar literature [57], the Receiver Operating Characteristic (ROC) is the tool used to compare the methods under study in terms of True Positive Rate (TPR) and False Positive Rate (FPR). The ROC consists in plotting the TPR on the y-axis and the FPR on the x-axis for different values of thresholds for the anomaly scores to discern between normal and anomalous. They are computed with the following formulas, where P and N are respectively the total number of anomalies and normal data:

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{P} \quad (7.1)$$

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N} \quad (7.2)$$

The main reason for using these quantities in this context is that we do not have to choose a threshold for every algorithm but we can compare their performance over various threshold values. In addition we also use the Area Under the Curve (AUC-ROC) to summarize the performance of the algorithms in a single number. It consists of the integral of that curve and therefore the possible values are in the range $(0, 1)$ with 1 being the best. Even though some limitations have been reported for specific contexts [58], we still believe it is a good indicator to consider together with other metrics.

7.1.2 Precision, Recall, F1 Score

Other indicators we import from the Information Retrieval literature are the following:

$$Precision = \frac{TP}{TP + FP} = \frac{\text{No. Correct Predictions of Anmalous Hypervisors}}{\text{No. Hypervisors Predicted as Anomaly}} \quad (7.3)$$

$$Recall = \frac{TP}{TP + FN} = \frac{\text{No. Correct Predictions of Anmalous Hypervisors}}{\text{No. of real Anmalous Hypervisors}} \quad (7.4)$$

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (7.5)$$

We highlight that the Recall coincides with the True Positive Rate used in the ROC and the F1 score is the harmonic mean of the other two indicators: Precision and Recall.

7.1.3 Distribution of Anomaly Scores

The final goal of the anomaly score is to discern between normal instances and anomalous ones. Therefore the distribution of anomaly score of the two groups is crucial to visually verify that the two distributions can be separated with a threshold on the anomaly score value. In fact, in a good algorithm for Anomaly Detection we expect that the normal instances have mostly a lower score than the anomalous ones, which on the contrary should have a higher anomaly score.

7.1.4 Computation performance

Besides the detection performance measures, the average training time is another crucial characteristic to compare methods used in an operational context such as the one at CERN.

7.1.5 Similarities among Methods

Once we have the results of single methods we then want to also see if they are significantly different from each other. Kendall's Tau coefficient [46] is an option proposed by [59] to measure the agreement of two anomaly detection rankings (Section 3.6.1). We use this as a tool to analyse the diversity required for the creation of a good ensemble strategy. We considered the original formulation of [46] and the implementation of scipy¹²:

$$\tau(a, b) = \frac{(P - Q)}{\sqrt{(P + Q + T) \cdot (P + Q + U)}}$$

where a and b are the two rankings and P is the number of concordant pairs, Q the number of discordant pairs, T the number of ties only in a , and U the number of ties only in b . Note that ties that are present in both rankings are not considered.

Nonetheless the ranking approach has a very important limitation with regard to our scenario. In fact the longer the compared rankings are the more probable it is to have a low coefficient, and consequently low measured similarity. Therefore we confine the use of Kendall's tau only to an analysis of a single week of data, whereas for the comparison of all anomaly scores we rely on the correlation of normalized anomalies

¹²Scipy Python library: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html>

scores. In fact after the normalization of the anomaly scores, we have that every algorithm assigns a score that is somehow centered in zero on training data, therefore we can compare their similarities from a scatter plot representing each window of data analysed with the x and y coordinate corresponding to the normalized score of two algorithms.

7.2 Experiment A - CERN curated Dataset

We run the analysis of the data on about six months of annotated data. We repeat this experiment for both a **Batch** and a **Shared** group separately. We analyse the non-overlapping window data with the selected algorithms. For each method we produce the anomaly scores of every hypervisor in the two Openstack cells under study for a length of around six months. In a final step we compare the produced anomaly scores with the ground truth labels collected by our experts and produce the ROC curve and compute the AUC, Precision, Recall, False Positive Rate and F1. Note that not all the data are completely annotated, therefore we exclude the windows for which a hypervisor is not annotated from the computation of performance measures. These are the performance measures on the two cases:

- *Batch* dataset (described in Section 4.4.1). Complete results are visible in the Appendix in Figures B.1 and B.2 for Traditional and Deep methods respectively.
- *Shared* dataset (described in Section 4.4.2). Complete results are visible in the Appendix in Figures B.3 and B.4 for Traditional and Deep methods respectively.

7.2.1 Formalized Training and Testing Procedure

For the comparative study, we adapt all the methods to work with the same quantity of input data both at train and test time, the only difference is in the format, either *greymap* or *vectorized*. In Algorithm 1, we summarize the steps we undergo both during training and testing for a single algorithm. The main points concern the fact that we train on one week and test on the consecutive week, and we rank one window of data at a time. Moreover, we have a two step normalization procedure as explained in Sections 5.4.2 and 6.2. We normalize the data to have the features in the same range and we normalize the anomaly scores

to have the various methods producing a score centered in zero for the normal case. Note that in the Train procedure (Algorithm 2) we create all possible windows from the train data, we also allow overlapping windows, therefore for example with $c = 10$ minutes, that means that a datapoint summarizes 10 minutes of data, and $w = 8$ hours we have 144 different windows for every hypervisor in one day. On the other hand for the Test procedure (Algorithm 3) we create only non-overlapping windows that can be created within the current week test data. With the same parametrization just mentioned we will have 3 different windows for every hypervisor in one day. Another note on the amount of training data, for the Deep learning methods we give all the possible windows as input, whereas for the traditional method we subsample 1000 samples since they are not scaling as well as the Deep methods, especially these distance based methods like KNN and LOF that store the data seen at train time.

Algorithm 1 Train-Test Procedure

```

1: all_results = empty list                                ▷ Prepare the result list
2: dataset = either Batch or Shared Dataset                ▷ Select your benchmark
   Dataset
3: weeks = split(dataset)                                  ▷ Divide dataset in chunks of one week each
4: sorted_weeks = sort(weeks)                             ▷ Sort the weeks, the oldest first
5: algo = initialize algorithm
6: for index in range[1, len(sorted_weeks)] do           ▷ Iterate over weeks
7:     current_week = sorted_weeks[index]
8:     previous_week = sorted_weeks[index - 1]
9:     (tr_data_means, tr_data_st_devs, tr_mean_score, tr_st_dev_score) =
       TRAIN(previous_week, algo)
10:    week_results = TEST(current_week, algo, tr_data_means,
        tr_data_st_devs, tr_mean_score, tr_st_dev_score)
11:    all_results.append(week_results)
12: end for

```

7.2.2 Algorithms Training Time

In Figure 7.2 and 7.3 we summarize the training time for the traditional and deep methods respectively. We recall that every algorithm undergoes an entire training procedure for every week. An important decision that heavily influences the comparison between deep and tra-

Algorithm 2 Train Routine

```
1: procedure TRAIN(previous_week, algo)
2:   previous_week = sorted_weeks[index - 1]
3:   tr_means = empty list                                ▷ Vector of metrics' means
4:   tr_st_devs = empty list                              ▷ Vector of metrics' st.dev.
5:   for m in metrics do
6:     mean_m, st_dev_m = get_mean_stdev(previous_week, m)
7:     tr_means.append(mean_m)
8:     tr_st_devs.append(st_dev_m)
9:   end for
10:  std_prev_week = standardize(previous_week, tr_means, tr_st_devs)
11:  train_dataset = get_all_possible_windows(std_prev_week)
12:  if algo is not Deep Learning then
13:    train_dataset = subsample(train_dataset, n=1000)
14:  end if
15:  algo.fit(train_dataset)                                ▷ Train the algorithm
16:  train_anomaly_scores = algo.assign_score(train_dataset)
17:  tr_mean_score = mean(train_anomaly_scores)
18:  tr_st_dev_score = st_dev(train_anomaly_scores)
19:  return (tr_means, tr_st_devs, tr_mean_score, tr_st_dev_score)
20: end procedure
```

Algorithm 3 Test Routine

```
1: procedure TEST(current_week, algo, tr_means, tr_st_devs,
   tr_mean_score, tr_st_dev_score)
2:   week_results = empty list
3:   st_current_week = standardize(current_week, tr_means, tr_st_devs)
4:   test_dataset = get_non_overlapping_windows(std_current_week)
5:   test_anomaly_scores = empty list
6:   for window in test_dataset do
7:     w_score = algo.assign_score(window)
8:     n_w_score = normalize(w_score, tr_mean_score, tr_st_dev_score)
9:     week_results.append(n_window_score)    ▷ Add also information
       about hypervisor and time of end of the window
10:  end for
11:  return week_results
12: end procedure
```

ditional methods consists in feeding the traditional algorithms only a subsample of size 1000 windows among all the possible windows we can build with the week of training data.

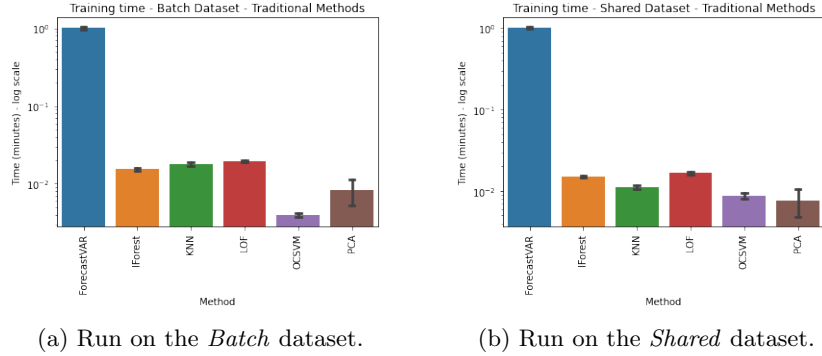


Figure 7.2: Training Time of Traditional methods under study. We see the mean value and variance (error bar) of the time required by the specific method to train on one week of data. They are trained only on a random subsample of 1000 windows, to avoid scale issues on distance based methods. Note that the y-axis is in log-scale.

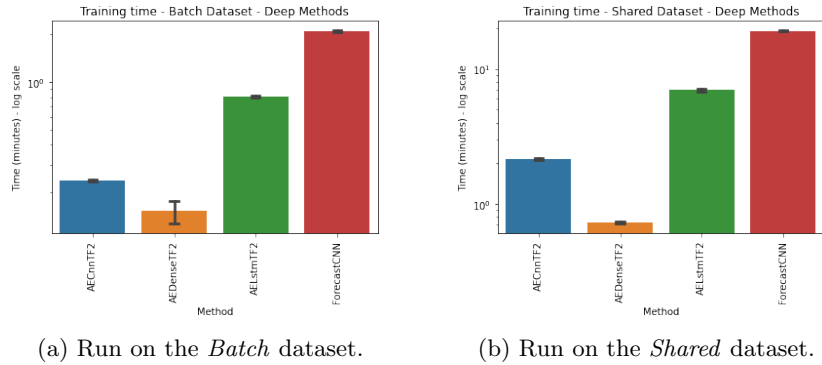


Figure 7.3: Training Time of Deep methods under study. We see the mean value and variance (error bar) of the time required by the specific method to train on one week of data. The number of epochs of train is 20. Note that the y-axis is in log-scale.

7.2.3 Detection Performance

In Table 7.1 and 7.2 we report all the mean and variance of the Area Under the Receiving Operating Characteristic Curve (AUC-ROC) for all methods: traditional, deep and ensemble. The mean and variance are computed along the various runs, one per week, for a total of 26

runs in the *Batch* case and 32 for the *Shared* one.

Method	Mean	StD
KNN	0.697	0.173
LOF	0.877	0.168
PCA	0.910	0.124
OCSVM	0.814	0.140
IForest*	0.921	0.134
ForecastVAR	0.704	0.234
AEDenseTF2	0.909	0.124
AECnnTF2	0.753	0.208
AELstmTF2	0.829	0.203
ForecastCNN	0.910	0.159
ENS-MAX	0.883	0.158
ENS-MIN	0.739	0.196
ENS-MEDIAN	0.884	0.153
ENS-AVERAGE	0.877	0.158
ENS-CUMSUM	0.877	0.158
ENS-LOGREG	0.711	0.231

Table 7.1: AUC-ROC of all methods on *Batch*. It is computed as average on all the weeks for the *Batch* scenario. The best mean is highlighted with a *.

Method	Mean	StD
KNN	0.739	0.065
LOF	0.698	0.121
PCA	0.713	0.137
OCSVM	0.693	0.084
IForest	0.752	0.092
ForecastVAR	0.724	0.095
AEDenseTF2	0.713	0.139
AECnnTF2	0.777	0.071
AELstmTF2*	0.793	0.058
ForecastCNN	0.715	0.138
ENS-MAX	0.754	0.092
ENS-MIN	0.717	0.087
ENS-MEDIAN	0.750	0.103
ENS-AVERAGE	0.768	0.096
ENS-CUMSUM	0.768	0.096
ENS-LOGREG	0.722	0.096

Table 7.2: AUC-ROC of all methods on *Shared*. It is computed as average on all the weeks for the *Shared* scenario. The best mean is highlighted with a *.

7.3 Experiment B - CERN Artificial Setting

As described in Section 4.5, in this artificial setting we merge *Batch* and *Shared* considering them normal and anomalous data respectively. In practice, we start from a week of windows of 183 *Batch* hypervisors and a week of 200 *Shared* hypervisors, we divide them into windows of 8 hours. Subsequently we iterate over all the non overlapping 8 hours intervals of *Batch* windows and we train every method on the 183 *Batch* hypervisors then we test its performance by scoring the same 183 *Batch* hypervisors plus 91 *Shared* hypervisors extracted from the

week of *Shared* windows. The quantity of injected windows is chosen to keep a 2:1 relationship between normal and anomalous data.

In the ideal case the methods should be able to tell if at test time an hypervisor is a *Batch* or *Shared* one assigning respectively a low and high anomaly score. To increase the difficulty of the artificial task we select the *Shared* hypervisors to inject based on a similarity score with the *Batch* hypervisors in the same window. The similarity score is computed with the dot product of the normalized vectors representing the hypervisors, we select the one that is on average most similar. Moreover we design three different scenarios of increasing complexity:

- **unprocessed data.** We keep the data unprocessed, they have their original mean and variance, and the disproportion among metrics is also left untouched, meaning that for example the *CPU idle* is a percentage while the *SWAP space free* is a number in Gigabytes therefore billions.
- **same mean.** We report both windows from *Shared* and *Batch* hypervisor to have zero mean on all the input metrics.
- **same mean and variance.** We report both windows from *Shared* and *Batch* hypervisor to have zero mean and unit variance on all the input metrics.

Given that the performance of the final anomaly detection system is not measured on the ability to discern between *Batch* and *Shared* hypervisors we decided to restrict these experiments only to some representative methods from the two families: traditional and deep. The goal is to understand if the two user categories are significantly different from each other and at the same time measure the ability of the two families to capture patterns that go beyond the mean and variance.

In Figures 7.4, 7.5, 7.6 we report ROC and score distributions of the three scenarios tested on one window of data for four traditional methods (PCA, IForest, LOF, KNN) and two Deep Learning ones (Autoencoder Fully connected and CNN). In the first row we see that all methods can effectively distinguish between the two categories when they process the data with their mean and variance, and this is in accordance with the expert belief that the two groups show different behaviour: *Batch* trying to be fully utilized whereas *Shared* that have a significant lower utilization. In the second row we notice that all methods are successful and therefore the variance alone is also containing enough information to distinguish between the two groups. In the third row, both nearest neighbors methods (KNN and LOF) and more advanced Autoencoders (CNN and LSTM) show a superior ability in recognizing the pattern also when the anomalies have same mean and variance of normal data. This motivate the use of more complex Neural Networks and also the validity of quite simple methods like KNN. These method that decent performance even when normal and abnormal data have the same mean and variance might then be able to capture additional information.

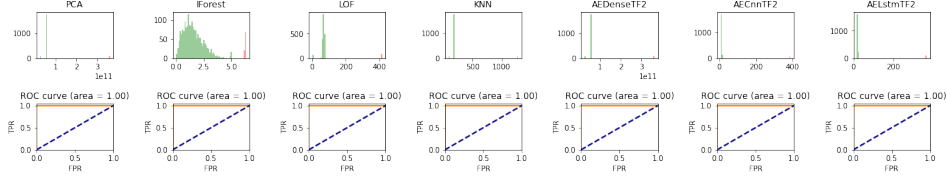


Figure 7.4: Unprocessed data. ROC and score distributions tested on one window of artificial setting with 1840 normal windows of Batch hypervisors and 90 anomalous windows of Shared hypervisors. Methods tested: PCA, IForest, LOF, KNN, Autoencoder Fully connected, AE CNN and AE LSTM.

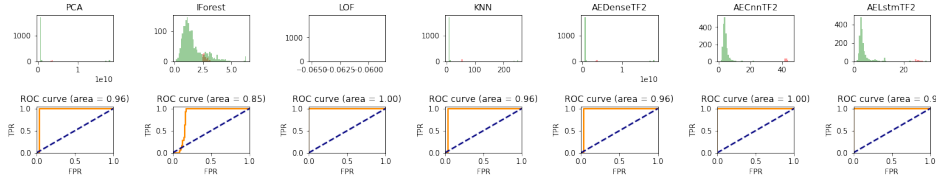


Figure 7.5: Same mean. ROC and score distribution tested on one window of artificial setting with 1840 normal windows of Batch hypervisors and 90 anomalous windows of Shared hypervisors. Methods tested: PCA, IForest, LOF, KNN, Autoencoder Fully connected, AE CNN and AE LSTM.

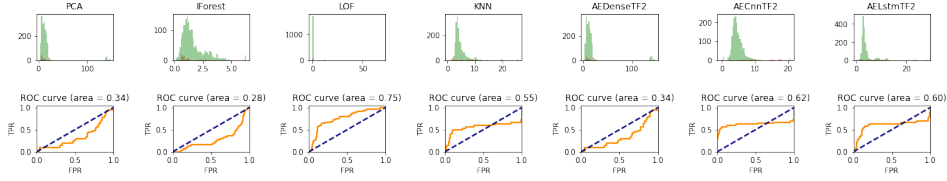


Figure 7.6: Same mean and Variance. ROC and score distribution tested on one window of artificial setting with 1840 normal windows of Batch hypervisors and 90 anomalous windows of Shared hypervisors. Methods tested: PCA, IForest, LOF, KNN, Autoencoder Fully connected, AE CNN and AE LSTM.

7.4 Algorithm Implementation Details

For the majority of the Outlier Detection methods we rely on the implementation provided by PyOD¹³ given the good level of code quality in terms of coverage and the support from the open source community. Note that for the three most popular traditional methods (LOF, iForest, OCSVM) the implementation of PyOD relies on the homonym

¹³Python Outlier Detection (PyOD) library, code available here: <https://github.com/yzhao062/pyod>

classes from scikit-learn ¹⁴. In the Table 7.3 every algorithm is paired with the library that implements it.

Algorithm Name	Implementation	Version
LOF	PyOD (scikit-learn)	0.8.0 (0.23.1)
IFOREST	PyOD (scikit-learn)	0.8.0 (0.23.1)
OCSVM	PyOD (scikit-learn)	0.8.0 (0.23.1)
PCA	PyOD	0.8.0
KNN	PyOD	0.8.0
ForecastVAR	this work	1.0
ForecastCNN	this work	1.0
AEFC	PyOD (adapted for TensorFlow 2 in this work)	0.8.0 (1.0)
AECC	this work	1.0
AELSTM	this work	1.0

Table 7.3: Algorithms used and their implementations

7.5 Discussion

We summarize in this section the main findings from the results of our three experiments: *Batch* dataset, *Shared* dataset and artificial setting. Firstly given the ease with which the anomaly detection methods can distinguish between the *Batch* and *Shared* hypervisors in the artificial setting (Figure 7.4), we can conclude that the two user categories are showing clearly different patterns in terms of both mean and variance (Figure 7.5). Therefore considering the two separate scenarios with two separate *Batch* and *Shared* benchmark datasets is a reasonable choice and confirms the initial experts' belief.

7.5.1 Performance and Task Complexity

If we consider the mean AUC of the various methods in Figure 7.7 for the *Batch* and *Shared* datasets, we can clearly see that the *Batch* datasets have the higher AUC results. We can hereby confirm that the *Batch* scenario represents an easier task; moreover, the lack of

¹⁴scikit-learn library code available here: <https://github.com/scikit-learn/scikit-learn/>

significant differences between traditional and deep methods in this case can be seen as an indicator of an easier anomaly detection scenario in case of *Batch* hypervisor monitoring. We explain this difference also from visual inspection of the anomalies on the *Batch* that are usually corresponding to almost perfect step functions in the metrics, most of the time the CPU Load, whereas this is not the case for the *Shared* dataset.

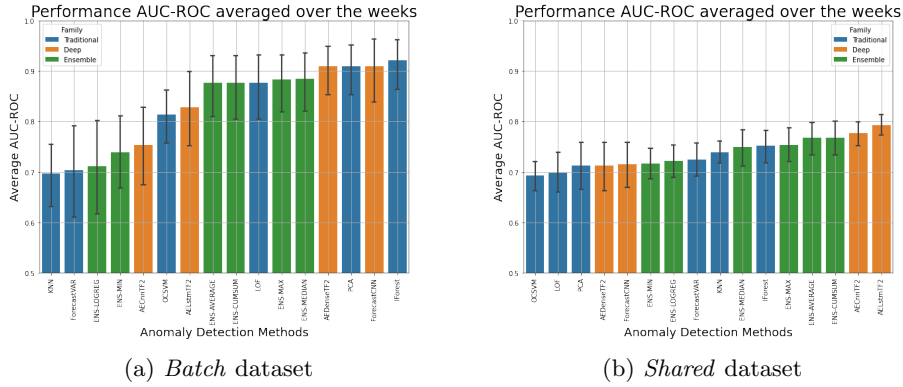


Figure 7.7: AUC-ROC all methods on *Batch* and *Shared*. It is computed as average over the weeks of analysis. The performance are sorted in increasing order and the color represent the families of traditional, deep and ensemble methods.

To reinforce this hypothesis that the *Batch* anomaly detection task probably does not need complex models to be solved we have that PCA performs incredibly well on par of other Deep learning methods like Autoencoder Fully Connected (AEDense). In fact, even if this deep model includes non linear activation functions it can still learn the linear solution of the problem. And the strong correlation between the PCA and AEDense anomaly scores in Figure 7.8 confirms the hypothesis that they converged on very similar solutions in both *Batch* and *Shared* dataset tasks. This good performance of the PCA is a good sign from a practical perspective, because it means that such a simple method can find most of the anomalies with a low False Positive Rate.

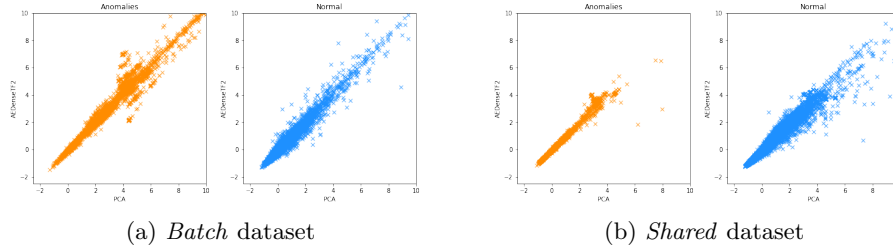


Figure 7.8: Anomaly scores correlation between Autoencoder Fully Connected and the PCA on both CERN curated dataset.

7.5.2 Input Data Quantity

The decision of subsampling the input for traditional methods prevents us from commenting on the training time difference between the traditional and deep families, but this choice is motivated by the fact that some of them are instance-based methods and therefore do not learn any model but simply store the data showing in the end a poor scalability. On the performance side subsampling does not affect traditional methods as much as deep learning ones that if trained on smaller dataset lead to a worse performance.

In terms of quantity of data required by the various methods we have two lesson learned in debugging problems with Autoencoder CNN and LOF methods respectively. We noticed that for the Autoencoder CNN in particular but also for other deep learning methods, learning on just non-overlapping windows of the previous week was not enough. In fact, that quantity of data for a common group of either *Batch* or *Shared* hypervisors contains around 4'200 windows and that is definitely not enough to cope with the need of Deep Learning methods. That is why we adopted the sliding windows approach in the training week. On the other hand for a traditional method like LOF, the usage of a disproportionately smaller number of neighbours than the size training set leads to a very poor result as shown in Figure 7.9

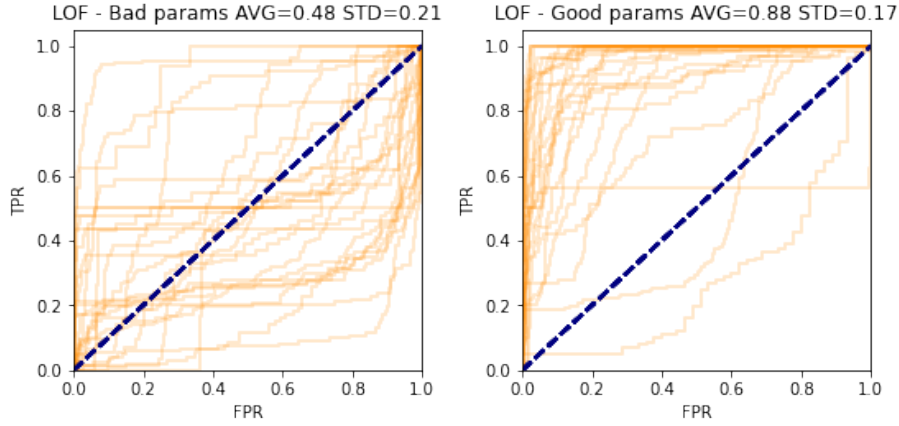


Figure 7.9: Example of bad and good LOF models. In both cases we use as training set 1000 samples. On the left we set the number of neighbours to 20 whereas on the right we set it to 200

We therefore highlight the importance to appropriately set a number of neighbours based on the a validation dataset if possible.

7.5.3 Novel Methods Discussion

The new methods proposed based on the forecasting principle (ForecastVAR and ForecastCNN) show varying behaviour in terms of AUC performances on the *Batch* dataset, wherese they achieve similar ones in the more complex setting of *Shared* hypervisors. Even though the different model could be enough to justify this difference, we also argue that the subsampling step present in the ForecastVAR method could be partially responsible for this high variance in the *Batch* case. In fact the ForecastVAR methods subsample 20 of the overall ca 200 hypervisors of the group and makes a model on top of them, this phenomenon can be less visible in the *Shared* dataset since we only subsample for a group of 80 hypervisors.

On the other hand the ForecastCNN is the best among the Deep Learning group in the *Batch* and the second best in the *Shared* one. We observe that it is equivalent in term of AUC-ROC to the Autoencoder Fully Connected (AEDense) in both cases. This trend is confirmed in Figure 7.10 where we see a clear correlation between the two methods' anomaly scores, with the Autoencoder that tends to give a higher score to normal data whereas Forecast CNN gives consistently lower scores. The clear advantage of our model is in terms of model complexity:

632'048 parameters for the Autoencoder and only 2'803 for the Forecaster (Figure C.4 and C.1 of the Appendix). We can attribute this saving in parameters to the better suitability of Forecast CNN to handle temporal data. Nonetheless we also point out that Forecast CNN requires a sensibly longer training time (Figure 7.3).

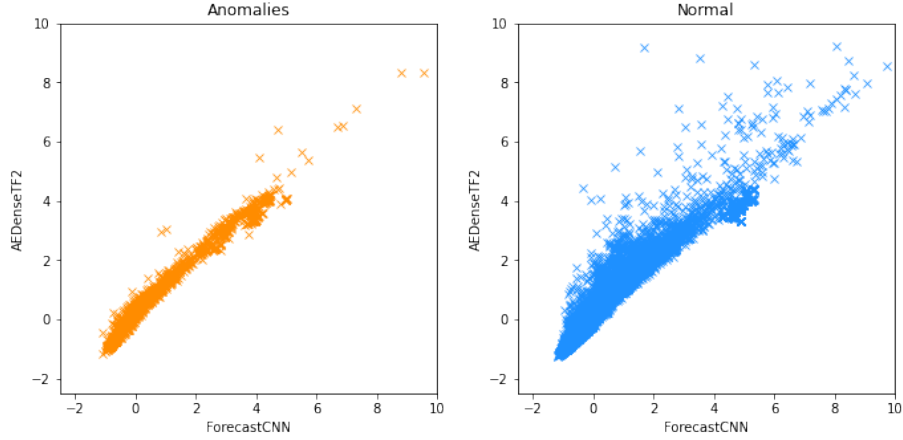


Figure 7.10: Anomaly scores correlation on the Shared dataset between Autoencoder Fully Connected and the Forecaster CNN.

Inspecting the results produced by the ensemble methods in Figure 7.7, we notice that most of the strategies are leading to similar AUC score with the exception of the min strategy that is clearly degrading the performance compared to the singular performances. We can explain this with the fact that if at least one method in the ensemble is erroneously classifying a window as normal when it is not, this drives the entire ensemble to say that it is a normal sample. Therefore the min strategy is more sensitive to the worst method in the ensemble with a tendency to overlook anomalies. On the other hand the two methods analyzed and recommended by [47], namely max and average, are not showing a significant difference between each other in this context. On the contrary our novel ensemble method based on a simple Linear regression combination using a small amount of labels is able to achieve the highest score among the proposed ensemble strategies in the *Shared* scenario. We see this as a indication that also in a learner of second level the use of few labeled data can lead to a superior performance in a similar way it usually happens thanks to semi-supervision as in other Anomaly Detection methods [60].

Chapter 8

Conclusion

In this work we have tackled the problem of Anomaly Detection on Time Series data for the CERN large-scale computing infrastructure with both traditional Machine Learning algorithms and Deep Learning ones. The main contributions and outputs of this thesis are the following:

- **Dataset Collection:** we created two curated CERN datasets representing the problem of anomaly detection on time series on a large-scale computing infrastructure, with two levels of task complexity;
- **Comparative evaluation study:** we compared the performance of a selection of algorithms on the newly collected CERN datasets;
- **System Implementation:** we implemented a Proof of Concept Anomaly Detection system for the CERN Cloud Infrastructure.
- **Novel variants:** we propose two new Anomaly Detection variants on time series (Forecaster VAR and Forecaster CNN) together with a new combination strategy for an ensemble method.

From the experimental results we noticed how traditional methods are incredibly good at spotting anomalies in groups of hypervisors that are running similar batch jobs with the goal of fully utilize the computing resource. For these scenarios PCA, Local Outlier Factor and Isolation Forest are achieving the goal with a minimal training time. On the other side, when the hypervisors are used in an on-demand fashion by multiple users and services with intermittent jobs, then it is harder to achieve good performance on the anomaly detection task and

Deep Learning methods have the power to achieve better performances than traditional methods if fed with enough data at training time. In particular architectures that model the temporal dimension, such as LSTM-based, can outshine the traditional Fully Connected architectures.

Among the limitations of the current thesis work, we highlight the following two points. The former is that we run our analysis following the expert opinion in most of our work, including the choice of metrics to monitor and the time frame of analysis. Nonetheless we believe that a systematic study on varying the number of input metrics and the time scale of analysis could have revealed respectively the robustness characteristic of the methods under study and the best time scale of analysis for the specific problem at hand. The second limitation is that we did not fully tune every single method to perform at the best, but we use the default settings in most of the cases. A detailed parameter search could have revealed the best configuration for every method comparing them at the best of their possibility on the current task. We did not explore these two directions due to a lack of time and the need of having a prototype ready, even in a Proof-of-Concept stage.

We therefore suggest some future directions worth exploring, in order of priority:

- explore the use of more metrics rather than the ones suggested by the expert and with different time scales of analysis;
- explore alternative network architectures for the proposed Deep Learning models for maximizing their performance on the hardest setting of *Shared* group of hypervisors;
- thanks to the two new labelled CERN datasets, extend the comparative study including supervised techniques;
- thanks to the now available workflow to annotate time series in the monitoring infrastructure, explore active learning approaches to decide on which temporal windows are worth to be annotated first;
- use the data from multiple Openstack Groups to train more robust Deep Learning Models for the two user categories separately: one model for *Batch* and one for *Shared*; in these way we expect to

learn from a larger pool of normal data and be more robust to small variation in the normal behaviour;

- explore a two-step procedure to combine the traditional machine Learning method and Deep Learning ones in a sequence to combine their strengths; for example, an Autoencoder architecture performs feature extraction and a traditional method detects anomalies in the new lower dimension feature space.

Bibliography

- [1] *Updated: Microsoft Azure's southern U.S. data center goes down for hours, impacting Office365 and Active Directory customers.* en-US. Sept. 2018. URL: <https://www.geekwire.com/2018/microsoft-azures-southern-u-s-data-center-goes-hours-impacting-office365-active-directory-customers/> (visited on 11/27/2020).
- [2] Rebecca Smith. “Hundreds die each year in NHS due to faulty machines: report”. en-GB. In: (July 2014). ISSN: 0307-1235. URL: <https://www.telegraph.co.uk/news/health/news/10988112/Hundreds-die-each-year-in-NHS-due-to-faulty-machines-report.html> (visited on 11/27/2020).
- [3] *Taking a closer look at LHC - LHC cost.* URL: https://www.lhc-closer.es/taking_a_closer_look_at_lhc/0.lhc_cost (visited on 11/27/2020).
- [4] Frank E. Grubbs. “Procedures for Detecting Outlying Observations in Samples”. en. In: *Technometrics* 11.1 (Feb. 1969), pp. 1–21. ISSN: 0040-1706, 1537-2723. DOI: 10.1080/00401706.1969.10490657. URL: <http://www.tandfonline.com/doi/abs/10.1080/00401706.1969.10490657>.
- [5] Manish Gupta et al. “Outlier Detection for Temporal Data: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.9 (Sept. 2014), pp. 2250–2267. ISSN: 1041-4347. DOI: 10.1109/TKDE.2013.184. URL: <http://ieeexplore.ieee.org/document/6684530/>.
- [6] Matteo Paltenghi and Giordano, Domenico. *data-analytics Gitlab Repository belonging to CERN cloud-infrastructure project.* en. URL: <https://gitlab.cern.ch/cloud-infrastructure/data-analytics> (visited on 11/27/2020).
- [7] *Hypervisor definition.* en-US. URL: <https://www.vmware.com/topics/glossary/content/hypervisor> (visited on 11/27/2020).

- [8] Tim Bell and Nikolay Tsvetkov. *CERN IT Monitoring*. Government & Nonprofit. Library Catalog: SlideShare. Dec. 2019. URL: <https://www.slideshare.net/noggin143/cern-it-monitoring> (visited on 11/27/2020).
- [9] Nikolay Tsvetkov. *Building scalable and reliable monitoring system*. URL: <https://indico.cern.ch/event/930896/> (visited on 11/26/2020).
- [10] D. Hawkins. *Identification of Outliers*. en. Monographs on Statistics and Applied Probability. Springer Netherlands, 1980. ISBN: 978-94-015-3996-8. DOI: 10.1007/978-94-015-3994-4. URL: <https://www.springer.com/gp/book/9789401539968> (visited on 11/27/2020).
- [11] Vic Barnett and Toby Lewis. *Outliers in Statistical Data*. en-us. 3rd Edition. Wiley, May 1994. URL: <https://www.wiley.com/en-us/Outliers+in+Statistical+Data%2C+3rd+Edition-p-9780471930945> (visited on 11/27/2020).
- [12] N. Zhao et al. “Automatically and Adaptively Identifying Severe Alerts for Online Service Systems”. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. ISSN: 2641-9874. July 2020, pp. 2420–2429. DOI: 10.1109/INFOCOM41043.2020.9155219.
- [13] Zezhong Zhang, Keyu Nie, and Ted Tao Yuan. “Moving Metric Detection and Alerting System at eBay”. In: *arXiv:2004.02360 [cs, stat]* (Apr. 2020). arXiv: 2004.02360. URL: <http://arxiv.org/abs/2004.02360> (visited on 11/27/2020).
- [14] Haowen Xu et al. “Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications”. In: *Proceedings of the 2018 World Wide Web Conference*. WWW ’18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, Apr. 2018, pp. 187–196. ISBN: 978-1-4503-5639-8. DOI: 10.1145/3178876.3185996. URL: <https://doi.org/10.1145/3178876.3185996> (visited on 11/27/2020).
- [15] Hansheng Ren et al. “Time-Series Anomaly Detection Service at Microsoft”. en. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage AK USA: ACM, July 2019, pp. 3009–3017. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330680. URL: <https://dl.acm.org/doi/10.1145/3292500.3330680> (visited on 11/27/2020).

- [16] Sean J. Taylor and Benjamin Letham. “Forecasting at Scale”. en. In: *The American Statistician* 72.1 (2018). Publisher: Taylor & Francis Journals, pp. 37–45. URL: <https://ideas.repec.org/a/taf/amstat/v72y2018i1p37-45.html> (visited on 11/27/2020).
- [17] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.
- [18] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. “Efficient algorithms for mining outliers from large data sets”. In: *ACM SIGMOD Record* 29.2 (May 2000), pp. 427–438. ISSN: 0163-5808. DOI: 10.1145/335191.335437. URL: <https://doi.org/10.1145/335191.335437> (visited on 11/27/2020).
- [19] Markus M. Breunig et al. “LOF: identifying density-based local outliers”. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. SIGMOD ’00. New York, NY, USA: Association for Computing Machinery, May 2000, pp. 93–104. ISBN: 978-1-58113-217-5. DOI: 10.1145/342009.335388. URL: <https://doi.org/10.1145/342009.335388> (visited on 08/25/2020).
- [20] M. Shyu et al. “A Novel Anomaly Detection Scheme Based on Principal Component Classifier”. In: 2003, pp. 172–179.
- [21] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation Forest”. In: *2008 Eighth IEEE International Conference on Data Mining*. 2008.
- [22] Bernhard Scholkopf et al. “Support Vector Method for Novelty Detection”. In: *NIPS 2000*.
- [23] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001. ISBN: 978-0-262-19475-4.
- [24] Konrad Rieck et al. “Support Vector Machines”. en. In: *Handbook of Computational Statistics: Concepts and Methods*. Ed. by James E. Gentle, Wolfgang Karl Härdle, and Yuichi Mori. Springer Handbooks of Computational Statistics. Berlin, Heidelberg: Springer, 2012, pp. 883–926. ISBN: 978-3-642-21551-3. DOI: 10.1007/978-3-642-21551-3_30. URL: https://doi.org/10.1007/978-3-642-21551-3_30 (visited on 11/27/2020).

- [25] Yongli Zhang. “Support Vector Machine Classification Algorithm and Its Application”. en. In: *Information Computing and Applications*. Ed. by Chunfeng Liu, Leizhen Wang, and Aimin Yang. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2012, pp. 179–186. ISBN: 978-3-642-34041-3. DOI: 10.1007/978-3-642-34041-3_27.
- [26] Nathan Shone et al. “A Deep Learning Approach to Network Intrusion Detection”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.1 (Feb. 2018). Conference Name: IEEE Transactions on Emerging Topics in Computational Intelligence, pp. 41–50. ISSN: 2471-285X. DOI: 10.1109/TETCI.2017.2772792.
- [27] Raghavendra Chalapathy and Sanjay Chawla. “Deep Learning for Anomaly Detection: A Survey”. In: *arXiv:1901.03407 [cs, stat]* (Jan. 2019). arXiv: 1901.03407. URL: <http://arxiv.org/abs/1901.03407> (visited on 11/27/2020).
- [28] Pascal Vincent et al. “Extracting and composing robust features with denoising autoencoders”. In: *Proceedings of the 25th international conference on Machine learning*. ICML ’08. New York, NY, USA: Association for Computing Machinery, July 2008, pp. 1096–1103. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390294. URL: <https://doi.org/10.1145/1390156.1390294> (visited on 11/27/2020).
- [29] Simon Hawkins et al. “Outlier Detection Using Replicator Neural Networks”. en. In: *Data Warehousing and Knowledge Discovery*. Ed. by Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 170–180. ISBN: 978-3-540-46145-6. DOI: 10.1007/3-540-46145-0_17.
- [30] Mayu Sakurada and Takehisa Yairi. “Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction”. In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. MLSDA’14. New York, NY, USA: Association for Computing Machinery, Dec. 2014, pp. 4–11. ISBN: 978-1-4503-3159-3. DOI: 10.1145/2689746.2689747. URL: <https://doi.org/10.1145/2689746.2689747> (visited on 11/27/2020).
- [31] Chuxu Zhang et al. “A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 1409–1416. DOI: 10.1609/aaai.v33i01.33011409.

- [32] Xuyun Fu et al. “Aircraft engine fault detection based on grouped convolutional denoising autoencoders”. In: *Chinese Journal of Aeronautics* 32 (Feb. 2019). DOI: 10.1016/j.cja.2018.12.011.
- [33] Jonathan Masci et al. “Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction”. en. In: *Artificial Neural Networks and Machine Learning – ICANN 2011*. Ed. by Timo Honkela et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 52–59. ISBN: 978-3-642-21735-7. DOI: 10.1007/978-3-642-21735-7_7.
- [34] Yifan Guo et al. “Multidimensional Time Series Anomaly Detection: A GRU-based Gaussian Mixture Variational Autoencoder Approach”. en. In: *Asian Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Nov. 2018, pp. 97–112.
- [35] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.
- [36] Kyunghyun Cho et al. “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches”. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. DOI: 10.3115/v1/W14-4012. URL: <https://www.aclweb.org/anthology/W14-4012>.
- [37] Mohsin Munir et al. “DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series”. en. In: *IEEE Access* 7 (2019), pp. 1991–2005. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2886457. URL: <https://ieeexplore.ieee.org/document/8581424/> (visited on 11/27/2020).
- [38] Mohsin Munir et al. “FuseAD: Unsupervised Anomaly Detection in Streaming Sensors Data by Fusing Statistical and Deep Learning Models”. In: *Sensors (Basel, Switzerland)* 19.11 (May 2019). ISSN: 1424-8220. DOI: 10.3390/s19112451. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6603659/> (visited on 11/27/2020).
- [39] Ulrich Schwickerath. *Update on the detection of anomalies in the Elasticsearch service*. URL: <https://indico.cern.ch/event/879832/> (visited on 11/27/2020).
- [40] Arthur Zimek, Ricardo J.G.B. Campello, and Jörg Sander. “Ensembles for unsupervised outlier detection: challenges and research questions a position paper”. In: *ACM SIGKDD Explorations Newsletter* 15.1 (Mar. 2014), pp. 11–22. ISSN: 1931-0145. DOI: 10.1145/2594473.

2594476. URL: <https://doi.org/10.1145/2594473.2594476> (visited on 11/27/2020).
- [41] Aleksandar Lazarevic and Vipin Kumar. “Feature bagging for outlier detection”. en. In: *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05*. Chicago, Illinois, USA: ACM Press, 2005, p. 157. ISBN: 978-1-59593-135-1. DOI: 10.1145/1081870.1081891. URL: <http://portal.acm.org/citation.cfm?doid=1081870.1081891> (visited on 11/27/2020).
 - [42] Tung Kieu et al. “Outlier Detection for Time Series with Recurrent Autoencoder Ensembles”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 2725–2732. DOI: 10.24963/ijcai.2019/378. URL: <https://doi.org/10.24963/ijcai.2019/378>.
 - [43] Hans-Peter Kriegel et al. “Interpreting and Unifying Outlier Scores”. en. In: *Proceedings of the 2011 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Apr. 2011, pp. 13–24. ISBN: 978-0-89871-992-5 978-1-61197-281-8. DOI: 10.1137/1.9781611972818.2. URL: <https://epubs.siam.org/doi/10.1137/1.9781611972818.2> (visited on 11/27/2020).
 - [44] Jacob Kauffmann et al. “The Clever Hans Effect in Anomaly Detection”. In: *arXiv:2006.10609 [cs, stat]* (June 2020). arXiv: 2006.10609. URL: <http://arxiv.org/abs/2006.10609> (visited on 08/01/2020).
 - [45] Erich Schubert et al. “On Evaluation of Outlier Rankings and Outlier Scores”. en. In: *Proceedings of the 2012 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Apr. 2012, pp. 1047–1058. ISBN: 978-1-61197-232-0 978-1-61197-282-5. DOI: 10.1137/1.9781611972825.90. URL: <https://epubs.siam.org/doi/10.1137/1.9781611972825.90> (visited on 11/27/2020).
 - [46] M. G. Kendall. “A New Measure of Rank Correlation”. In: *Biometrika* 30.1/2 (1938). Publisher: [Oxford University Press, Biometrika Trust], pp. 81–93. ISSN: 0006-3444. DOI: 10.2307/2332226. URL: <https://www.jstor.org/stable/2332226> (visited on 11/27/2020).
 - [47] Charu C. Aggarwal and Saket Sathe. “Theoretical Foundations and Algorithms for Outlier Ensembles?” en. In: *ACM SIGKDD Explorations Newsletter* 17.1 (Sept. 2015), pp. 24–47. ISSN: 19310145. DOI: 10.1145/2830544.2830549. URL: <http://dl.acm.org/citation.cfm?doid=2830544.2830549> (visited on 11/27/2020).

- [48] Ane Blázquez-García et al. “A review on outlier/anomaly detection in time series data”. In: *arXiv:2002.04236 [cs, stat]* (Feb. 2020). arXiv: 2002.04236. URL: <http://arxiv.org/abs/2002.04236> (visited on 11/27/2020).
- [49] Ya Su et al. “Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network”. en. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage AK USA: ACM, July 2019, pp. 2828–2837. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330672. URL: <https://dl.acm.org/doi/10.1145/3292500.3330672> (visited on 11/27/2020).
- [50] H. A. Dau et al. “The UCR time series archive”. In: *IEEE/CAA Journal of Automatica Sinica* 6.6 (Nov. 2019). Conference Name: IEEE/-CAA Journal of Automatica Sinica, pp. 1293–1305. ISSN: 2329-9274. DOI: 10.1109/JAS.2019.1911747.
- [51] P. Bergmann et al. “MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 2575-7075. June 2019, pp. 9584–9592. DOI: 10.1109/CVPR.2019.00982.
- [52] (16) (PDF) *Sensor Fault and Patient Anomaly Detection and Classification in Medical Wireless Sensor Networks*. en. URL: https://www.researchgate.net/publication/244987503_Sensor_Fault_and_Patient_Anomaly_Detection_and_Classification_in_Medical_Wireless_Sensor_Networks (visited on 08/25/2020).
- [53] Open Calgary. *Space Shuttle Main Propulsion System Anomaly Detection: A Case Study — NASA Open Data Portal*. en. URL: <https://data.nasa.gov/dataset/Space-Shuttle-Main-Propulsion-System-Anomaly-Detec/hwtc-f6bz> (visited on 11/27/2020).
- [54] Andrew F. Emmott et al. “Systematic construction of anomaly detection benchmarks from real data”. In: *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*. ODD ’13. Chicago, Illinois: Association for Computing Machinery, Aug. 2013, pp. 16–21. ISBN: 978-1-4503-2335-2. DOI: 10.1145/2500853.2500858. URL: <https://doi.org/10.1145/2500853.2500858> (visited on 11/27/2020).
- [55] Yue Zhao, Zain Nasrullah, and Zheng Li. “PyOD: A Python Toolbox for Scalable Outlier Detection”. In: *Journal of Machine Learning Re-*

- search* 20.96 (2019), pp. 1–7. URL: <http://jmlr.org/papers/v20/19-011.html>.
- [56] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998). Conference Name: Proceedings of the IEEE, pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.
- [57] Markus Goldstein and Seiichi Uchida. “A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data”. In: *PLOS ONE* (Apr. 2016). DOI: <https://doi.org/10.1371/journal.pone.0152173>.
- [58] Andrea Valassi. “Binary classifier metrics for optimizing HEP event selection”. In: *EPJ Web Conf.* 214 (2019), 06004. 8 p. DOI: 10.1051/epjconf/201921406004. URL: <https://cds.cern.ch/record/2699582>.
- [59] Arthur Zimek, Ricardo J. G. B. Campello, and Jörg Sander. “Data perturbation for outlier detection ensembles”. en. In: *Proceedings of the 26th International Conference on Scientific and Statistical Database Management - SSDBM '14*. Aalborg, Denmark: ACM Press, 2014, pp. 1–12. ISBN: 978-1-4503-2722-0. DOI: 10.1145/2618243.2618257. URL: <http://dl.acm.org/citation.cfm?doid=2618243.2618257> (visited on 11/27/2020).
- [60] Lukas Ruff et al. “Deep Semi-Supervised Anomaly Detection”. en. In: Apr. 2020. URL: https://iclr.cc/virtual_2020/poster_HkgH0TEYwH.html (visited on 11/27/2020).

Appendix A

Dataset

A.1 Annotations Overview

In this section we provide a visual overview of the annotations taken by the experts on the two datasets. We use a heatmap where every column represent an hypervisor whereas every row a window of data, note that we have 3 windows per day. We highlight with different colors the different types of windows:

- blue: normal window.
- orange: anomalous window.
- black: mixed (dropped for evaluation)
- white: no annotations provided by the expert on that window (dropped for evaluation).

For a precise description of every type of window refer to the Table 4.1 in the main text.

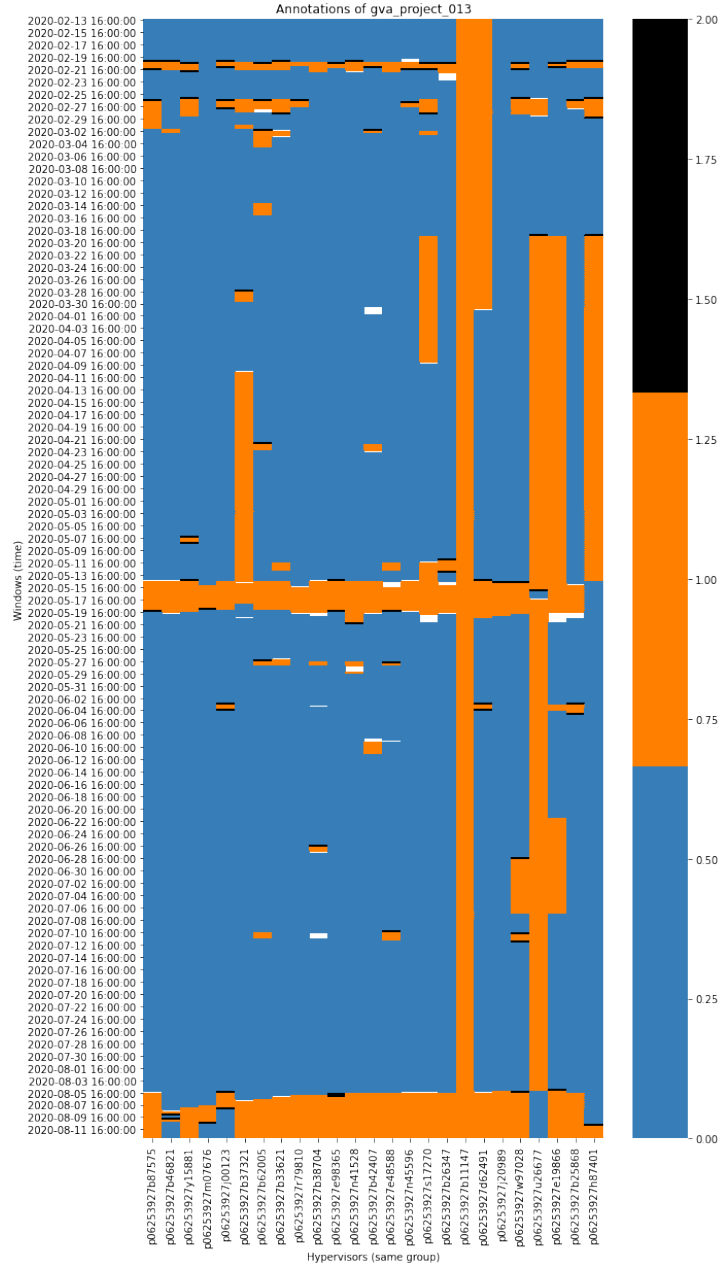


Figure A.1: Overview of Annotation Labels for Batch Openstack Cell: blue = normal, orange = anomaly, black = mixed, white = empty interval.

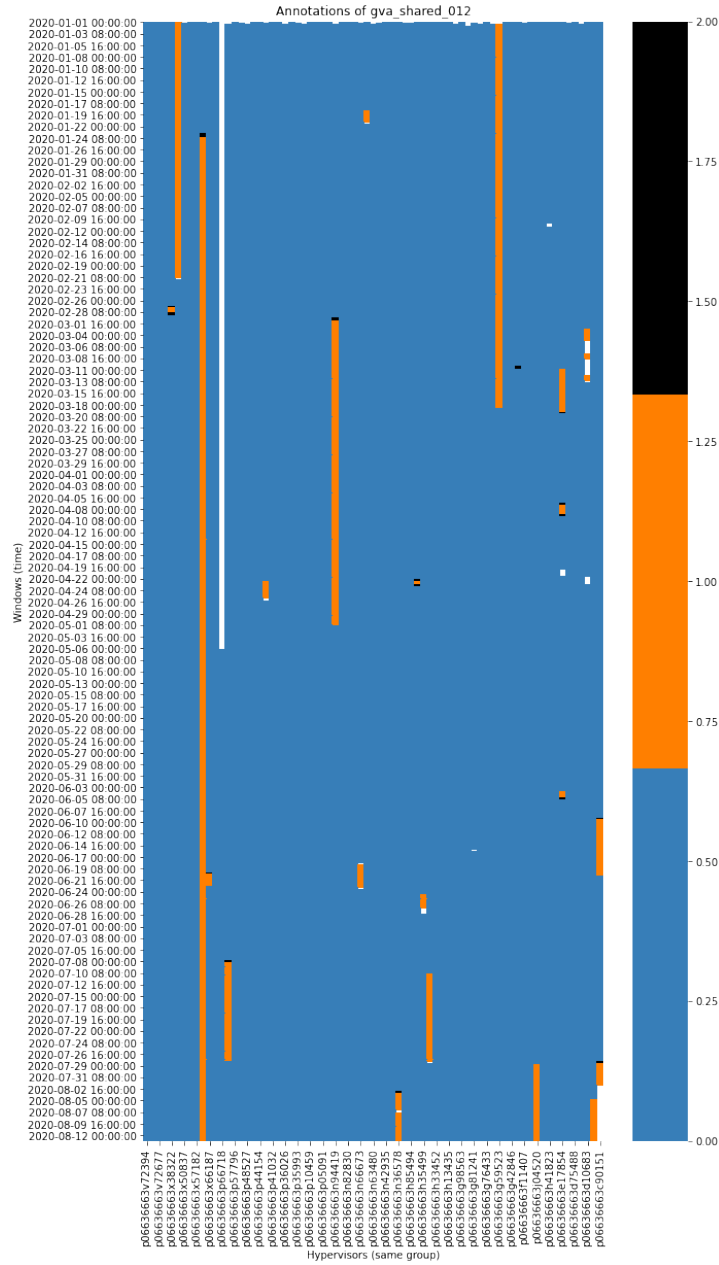


Figure A.2: Overview of Annotation Labels for Shared Openstack Cell: blue = normal, orange = anomaly, black = mixed, white = empty interval.

Appendix B

Complete Experimental Results

B.1 Performance Metrics: Complete Analysis

Next we show the results of the benchmark procedure of all the methods on the two curated datasets: for both *Batch* Dataset (Figures B.1 and B.2) and *Shared* Dataset (Figures B.3 and B.4).

B.2 Correlation of Anomaly Scores

In Figures B.5 and B.6 we show the comparison of the normalized anomaly scores of all algorithms with each other. Note that we analyse the correlation of scores of two algorithms in the range $[-2.5, 10]$ for both *Batch* and *Shared* dataset separately. Below the main diagonal we plot correlation among anomalous instances, whereas on the diagonal and above we visualise the normal samples.

B.3 Ensemble Performance

In Figures B.7 and B.8 we show the results of all the evaluated ensemble strategies: min, max, average, median, cumulative sum of all scores, linear regression combination (our method).

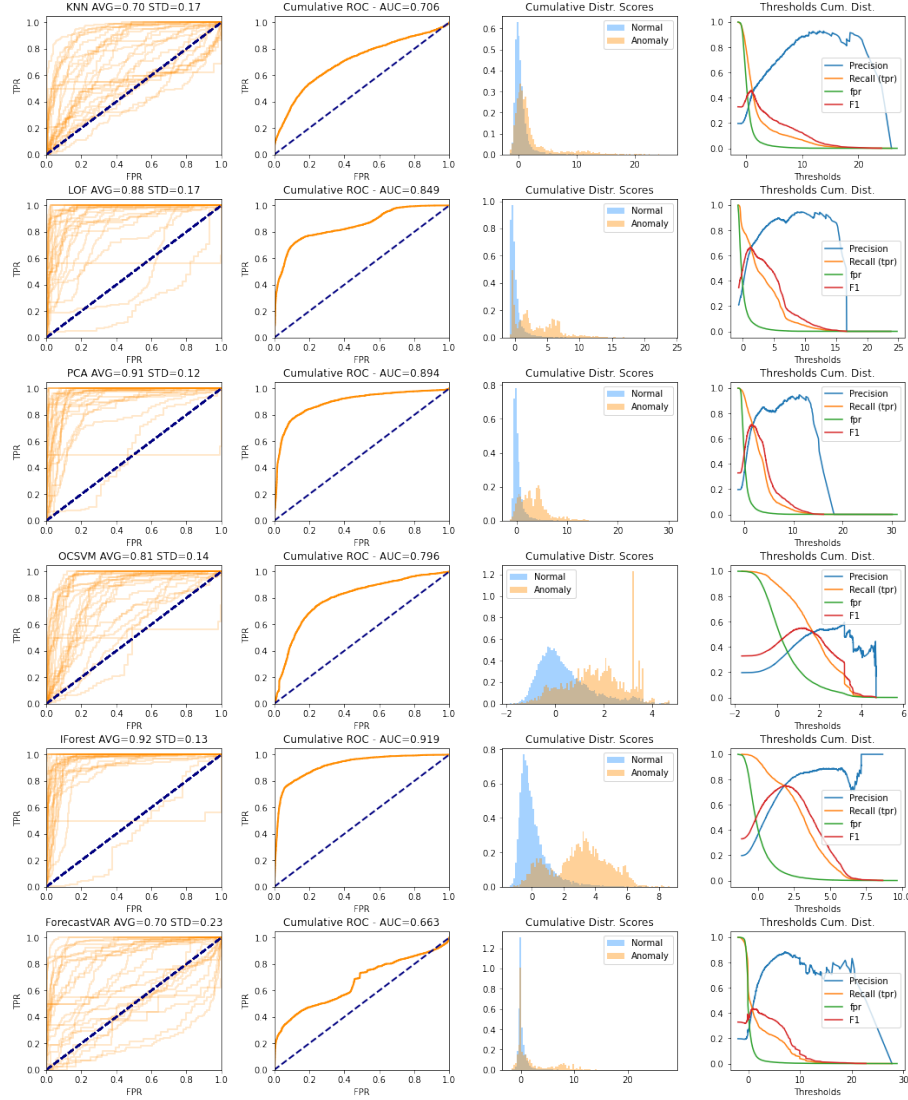


Figure B.1: Results on Batch dataset for Traditional Algorithms (KNN, LOF, PCA, OCSVM, IFOREST, ForecastVAR). The first column shows the multiple ROC curves build from the 26 weeks of data, the remaining represent cumulative quantities in which all weeks are represented together; we have in order the ROC, the cumulative distribution of anomaly scores and the performance metrics: Precision, Recall, False Positive Rate and F1.

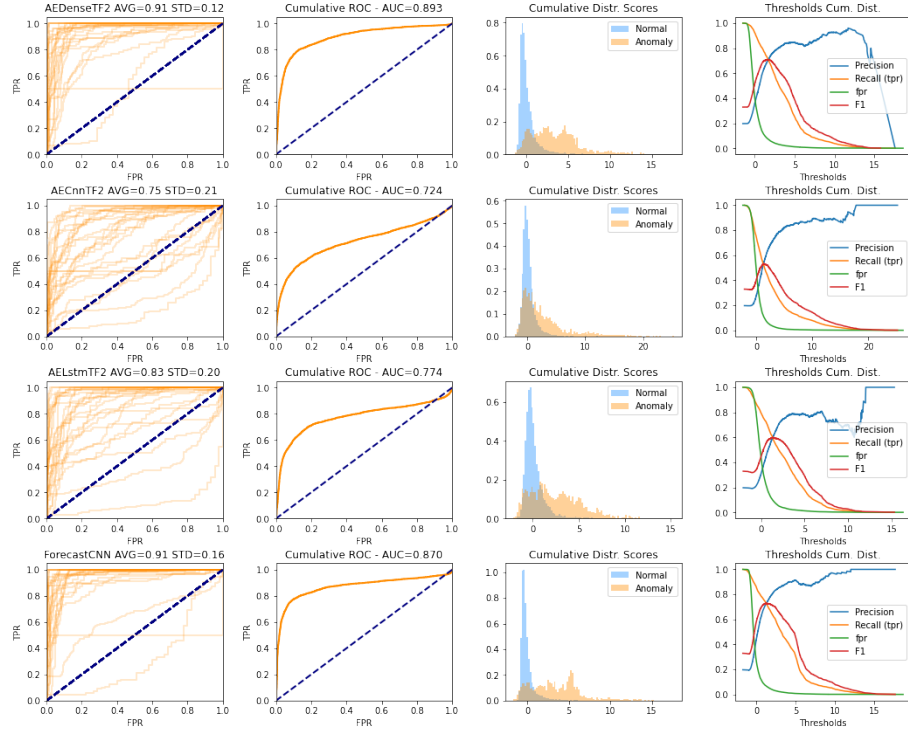


Figure B.2: Results on Batch dataset for Deep Algorithms (AEFC, AECNN, AELSTM, ForecastCNN). The first column shows the multiple ROC curves build from the 26 weeks of data, the remaining represent cumulative quantities in which all weeks are represented together; we have in order the ROC, the cumulative distribution of anomaly scores and the performance metrics: Precision, Recall, False Positive Rate and F1.

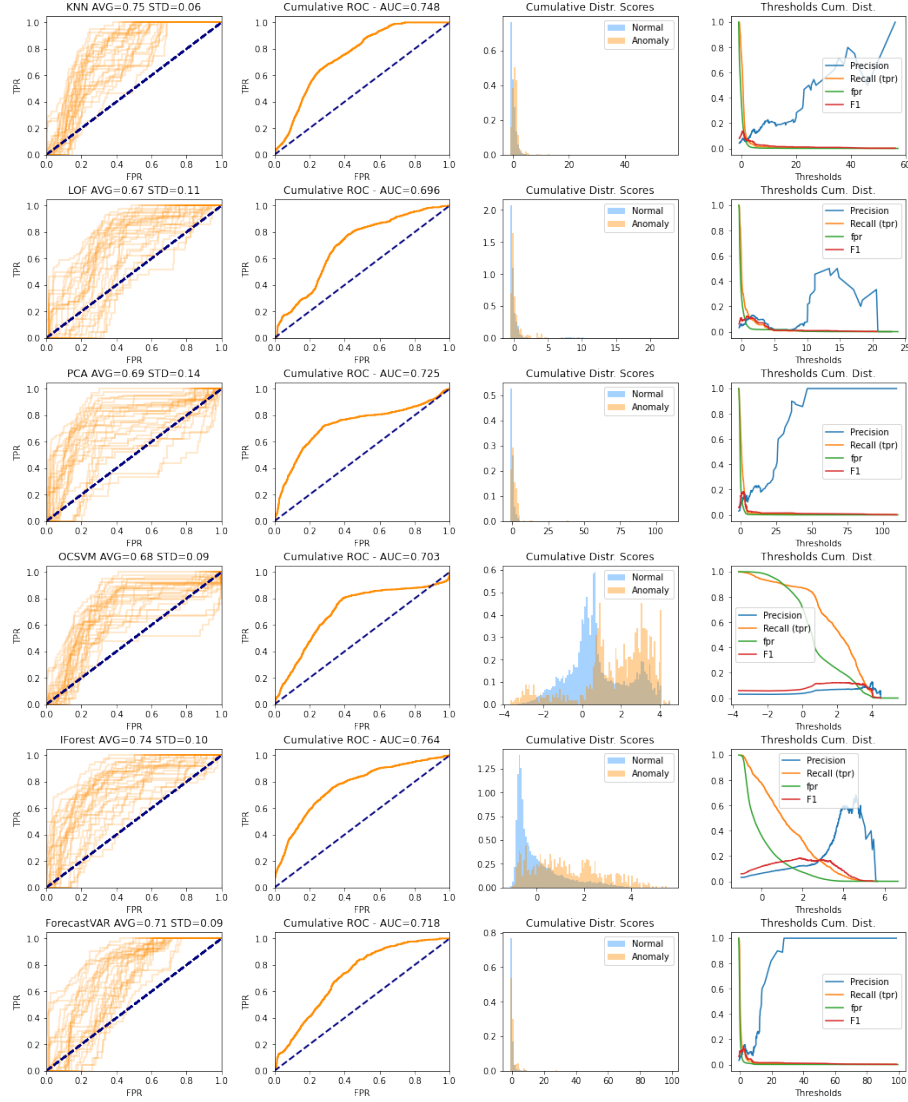


Figure B.3: Results on Shared dataset for Traditional Algorithms (KNN, LOF, PCA, OCSVM, IFOREST, ForecastVAR). The first column shows the multiple ROC curves build from the 32 weeks of data, the remaining represent cumulative quantities in which all weeks are represented together; we have in order the ROC, the cumulative distribution of anomaly scores and the performance metrics: Precision, Recall, False Positive Rate and F1.

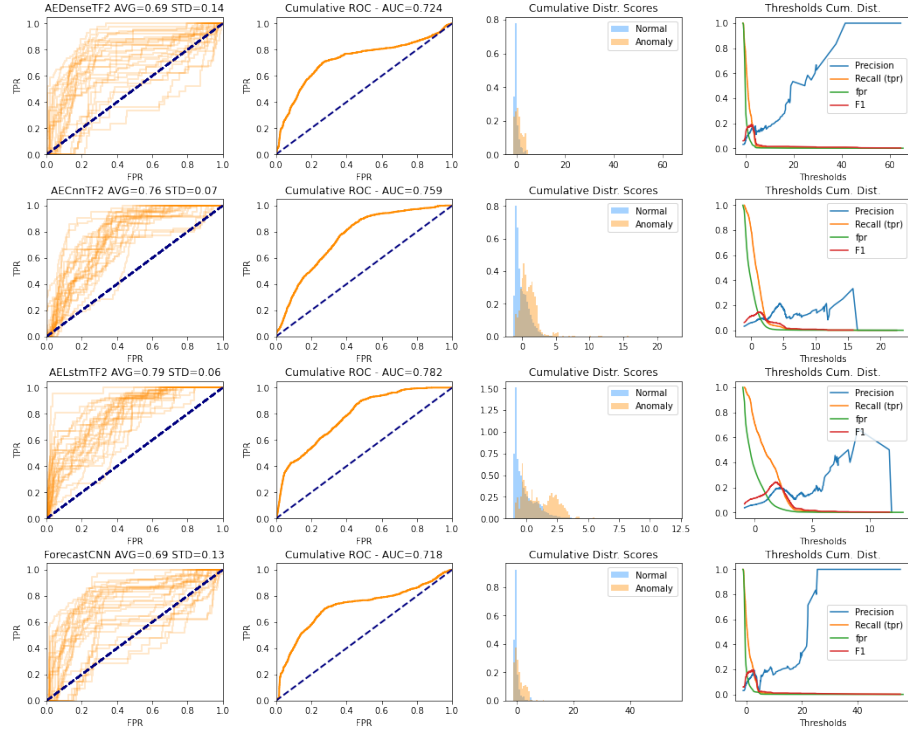


Figure B.4: Results on Shared dataset for Deep Algorithms (AEFC, AECNN, AELSTM, ForecastCNN). The first column shows the multiple ROC curves build from the 32 weeks of data, the remaining represent cumulative quantities in which all weeks are represented together; we have in order the ROC, the cumulative distribution of anomaly scores and the performance metrics: Precision, Recall, False Positive Rate and F1.

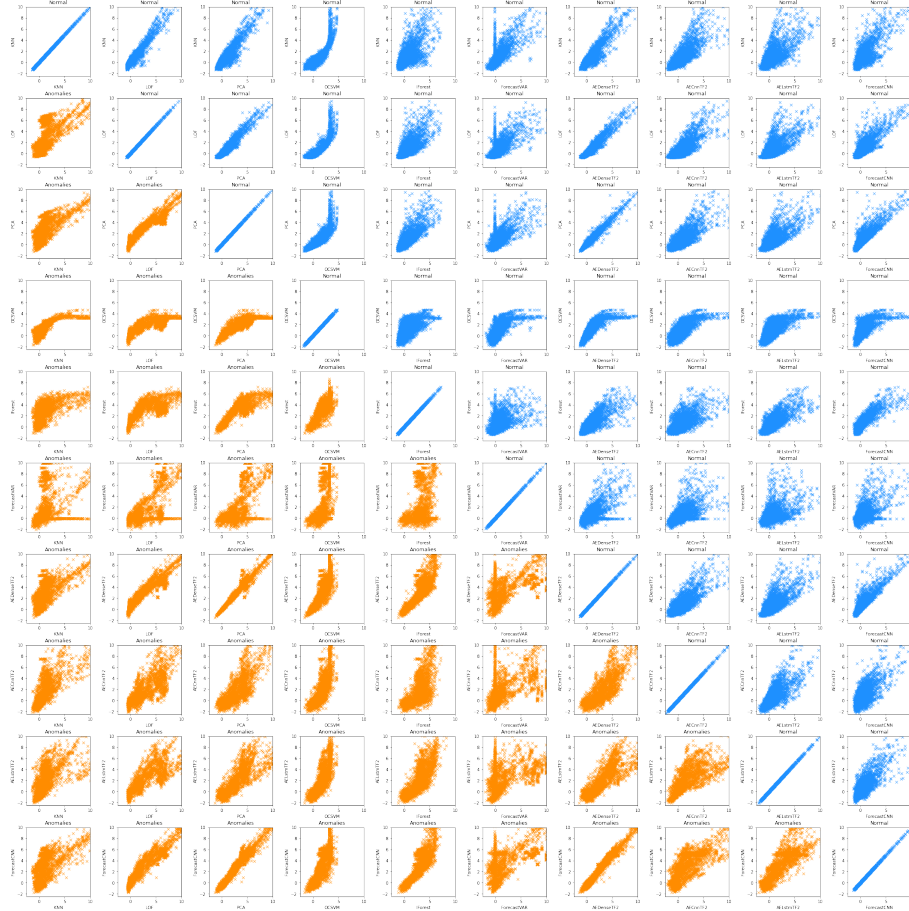


Figure B.5: Anomaly score correlation among all the pairs of algorithms for the Batch dataset. Below the main diagonal we have the anomalous instances, whereas on the diagonal and above we have the scores of normal instances.

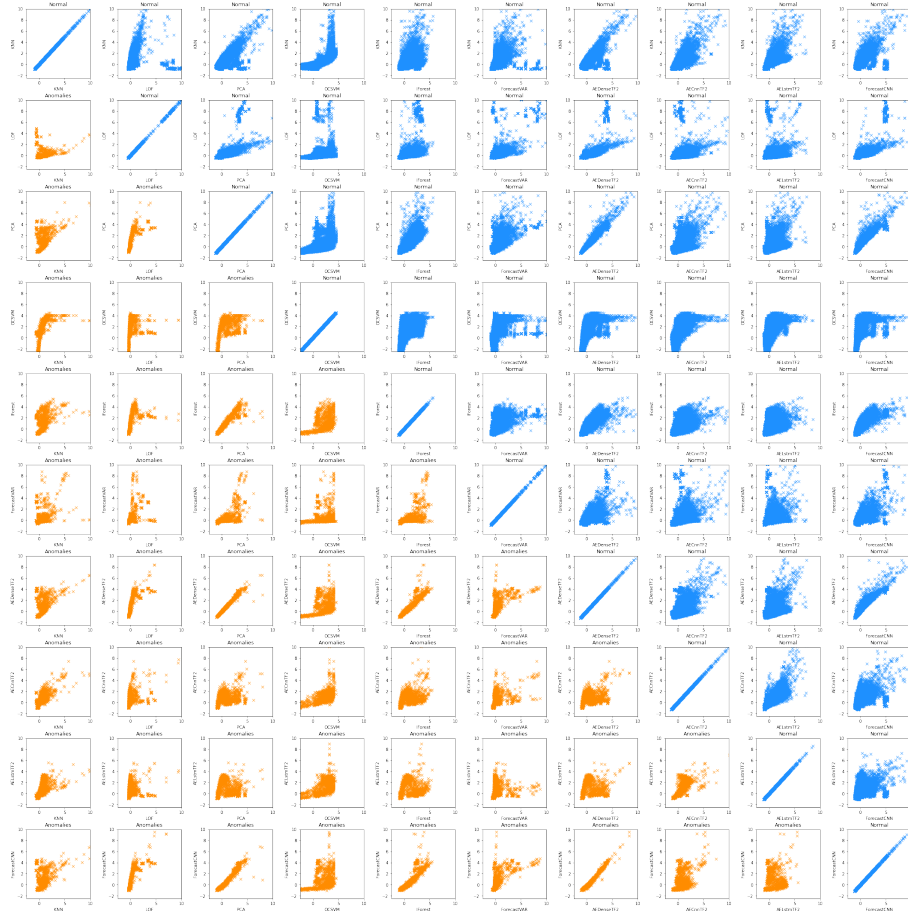


Figure B.6: Anomaly score correlation among all the pairs of algorithms for the Shared dataset. Below the main diagonal we have the anomalous instances, whereas on the diagonal and above we have the scores of normal instances.

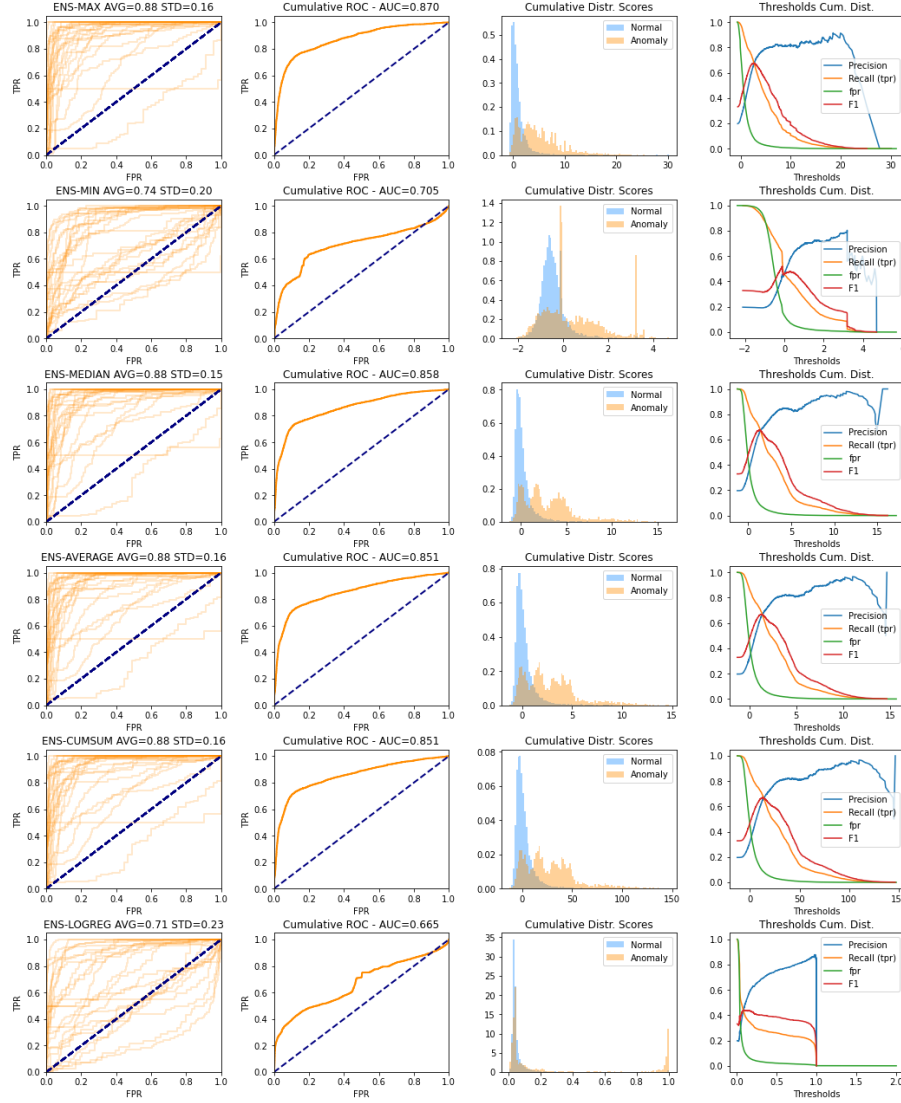


Figure B.7: Results on Batch dataset for Ensemble strategies (MIN, MAX, MED, AVG, CUMSUM, LOGREG). The first column shows the multiple ROC curves build from the 26 weeks of data, the remaining represent cumulative quantities in which all weeks are represented together; we have in order the ROC, the cumulative distribution of anomaly scores and the performance metrics: Precision, Recall, False Positive Rate and F1.

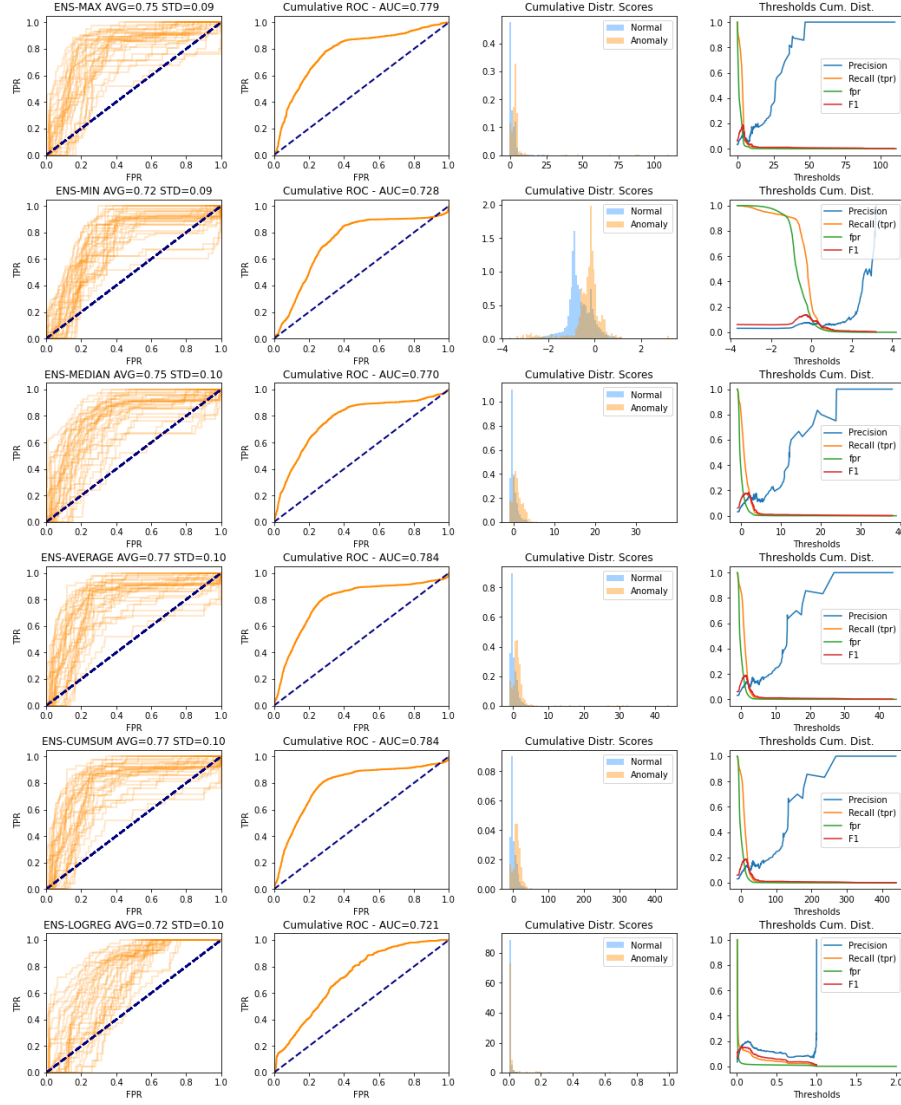


Figure B.8: Results on Shared dataset for Ensemble strategies (MIN, MAX, MED, AVG, CUMSUM, LOGREG). The first column shows the multiple ROC curves build from the 32 weeks of data, the remaining represent cumulative quantities in which all weeks are represented together; we have in order the ROC, the cumulative distribution of anomaly scores and the performance metrics: Precision, Recall, False Positive Rate and F1.

Appendix C

Appendix

C.1 Code Modification for Grafana Extension

Next we show the extension code for the client side Javascript of Grafana visualization platform.

```
1 // Get list of templated variables from the DOM
2 var mylist = document.querySelectorAll('[aria-label="
    Dashboard template variables submenu item"]');
3 //console.log(mylist);
4 for (var i = 0; i < mylist.length; i++) {
5     var mylabel = mylist[i].childNodes[0].childNodes[0].
        innerText;
6     var myvalue = mylist[i].childNodes[0].childNodes[1].
        innerText;
7
8     console.log("templated var label: " + mylabel);
9     console.log("templated var value: " + myvalue);
10    var mytag = mylabel + ":" + myvalue;
11    console.log(mytag);
12    //t.tags.push(myvalue);
13    t.tags.push(mytag);
```

C.2 Elasticsearch Document Format

Next we show the json format of the document representing one candidate anomaly found by our algorithms and sent to the monitoring infrastructure to be stored in Elasticsearch and queried later in Grafana.

```
1 "algo": {
2     "class_name": "PyODWrapperAnalyzer_IForest",
3     "version": -1,
```

```

4      "hparam_str": {},
5      "hparam_int": {},
6      "hparam_float": {},
7      "hparam_bool": {}
8  },
9      "entity_data": {
10         "entity": "p06253927b11147",
11         "cell_name": "gva_project_013",
12         "hostgroup": "cloud_compute/level2/batch/
gva_project_013"
13     },
14     "entity_metrics": {
15         "plugins": {
16             "swap_swapfile_swap_used": {
17                 "type": "swap",
18                 "type_instance": "used",
19                 "plugin_name": "swap"
20             },
21             "swap_swapfile_swap_free": {
22                 "type": "swap",
23                 "type_instance": "free",
24                 "plugin_name": "swap"
25             },
26             "vmem__vmpage_io_memory_in": {
27                 "plugin_instance": "",
28                 "type": "vmpage_io",
29                 "type_instance": "memory",
30                 "value_instance": "in",
31                 "plugin_name": "vmem"
32             },
33             "vmem__vmpage_io_memory_out": {
34                 "plugin_instance": "",
35                 "type": "vmpage_io",
36                 "type_instance": "memory",
37                 "value_instance": "out",
38                 "plugin_name": "vmem"
39             },
40             "cpu__percent_user": {
41                 "plugin_instance": "",
42                 "type": "percent",
43                 "type_instance": "user",
44                 "plugin_name": "cpu"
45             },
46             "cpu__percent_idle": {
47                 "plugin_instance": "",
48                 "type": "percent",
49                 "type_instance": "idle",
50                 "plugin_name": "cpu"
51             },

```

```
52     "cpu__percent_system": {
53       "plugin_instance": "",
54       "type": "percent",
55       "type_instance": "system",
56       "plugin_name": "cpu"
57     },
58     "load_longterm": {
59       "value_instance": "longterm",
60       "plugin_name": "load"
61     },
62     "memory__memory_free": {
63       "plugin_instance": "",
64       "type": "memory",
65       "type_instance": "free",
66       "plugin_name": "memory"
67     },
68     "interface__if_octets__tx": {
69       "type": "if_octets",
70       "type_instance": "",
71       "value_instance": "tx",
72       "plugin_name": "interface"
73     },
74     "interface__if_octets__rx": {
75       "type": "if_octets",
76       "type_instance": "",
77       "value_instance": "rx",
78       "plugin_name": "interface"
79     }
80   }
81 },
82 "preprocessing": {
83   "type": "standardization"
84 },
85 "temporal": {
86   "time_start_milli": 1593583200000,
87   "time_start": "2020-07-01T08:00:00.000Z",
88   "time_end_milli": 1593612000000,
89   "time_end": "2020-07-01T16:00:00.000Z",
90   "window_length": 48,
91   "step_resolution_minutes": 10
92 },
93 "anomaly": {
94   "score": 3.9858576333577713,
95   "rank_position": 1,
96   "rank_percentile": 100,
97   "explainability": {
98     "metric_0": "unsupported",
99     "nr_anomalous_metrics": -1,
100    "total_metrics": -1
```

```

101         }
102     },
103     "is_anomaly": true,
104     "slide_steps": 48,
105     "ids": {
106         "algo": "e5baa02f03953a43135216142ac60d0d38cf77e2",
107         "entity_data": "
f804e23d1862ac2eedcb4caacb179f223eb6a310",
108         "entity_metrics": "970
a503ce9349323e6651ce89ff84e1b6c302afb",
109         "preprocessing": "61
d6dbf3ad9cdca3b6a2e461ef5798d7b03a30bd",
110         "temporal": "45d11d1f3987dd1998ee00264b0d5dd94ada2d28
"
111     },
112     "document_id": "946169602798023
d3d89db2380eaac11462b0521",
113     "validity": true

```

C.3 Apache Spark - Big Data Preparation

This section contains the code used to prepare the data in the CERN Spark Cluster. We used the PySpark library to communicate with the cluster in Python code.

C.3.1 Data Reading

Next we show the code we use to read the data from HDFS and to keep only the hypervisors belonging to the Group of interest.

```

1 def data_reading(spark, plugins, days, hostgroups):
2     """Read all plugin and all days.
3
4     Params
5     -----
6     spark: spark context
7     plugins: dict.
8         Every key is the signal name. Then inside it we have
9         all the value for
10        the colums necessary to filter the correct data.
11    days: list of tuples
12        every tuple is (year, month, day)
13    hostgroups: list of str
14        full paths for the hostrgoroups you are interested in
15
16    Return

```

```

16  -----
17  all_data: PySpark DataFrame
18      containing data from the plugins and cells in the
following form:
19      timestamp | hostname | hostgroup | value | plugin
20      NB timestamp will be in seconds
21      NB hostgroup is the the extended version
22  """
23  plugins = copy.deepcopy(plugins)
24  inpaths = \
25      create_paths_for_collectd(basepath="/project/monitoring
/projectcollectd",
26                               plugins=plugins, days=days)
27  existing_inpaths = keep_only_existing_paths(spark=spark,
paths=inpaths)
28  [print(p) for p in sorted(existing_inpaths)]
29
30  try:
31      df_raw = spark.read.parquet(*existing_inpaths)
32  except AnalysisException:
33      print("path not found: %s" % existing_inpaths)
34      return
35
36  all_data = \
37      _keep_only_these_plugins_and_hg(df_raw, plugins,
hostgroups)
38
39  return all_data
40
41 def _keep_only_these_plugins_and_hg(df, plugins, hostgroups):
42     """Keep only the plugins and hostgroups.
43
44     Rename them also with the serialized version.
45     """
46     # filter hostgroups
47     only_my_hgs = df.filter(F.col("submitter_hostgroup").isin(
hostgroups))
48     # create the filter string
49     filter_str = ""
50     # for every plugin create the condition and then put them
in or
51     # df.filter('d<5 and (col1 <> col3 or (col1 = col3 and col2
<> col4))')
52     or_conditions = []
53     # filter all the plugin individually
54     for k in plugins.keys():
55         plugin_dict = plugins[k]
56         # for all attributes of this plugin
57         and_conditions = []

```

```

58     plugin_dict["plugin"] = plugin_dict.pop("plugin_name")
59     for sub_k in plugin_dict.keys():
60         value = plugin_dict[sub_k]
61         and_conditions.append(sub_k + " == '" + value + "'"
    )
62     plg_condition = "(" + " and ".join(and_conditions) + "
    )"
63     # print("plg_condition: ", plg_condition)
64     or_conditions.append(plg_condition)
65     # print("or_conditions: ", or_conditions)
66
67     # filter all interested line
68     filter_str = " or ".join(or_conditions)
69     print("filter_str: ", filter_str)
70     only_my_plugins = only_my_hgs.filter(filter_str)
71
72     df_list_plugin_ranamed = []
73     # filter line of each plugin to rename them
74     for plg_name, condition in zip/plugins.keys(),
    or_conditions):
75         df_current = only_my_plugins.filter(condition)
76         df_current = df_current.withColumn('plugin', F.lit(
    plg_name))
77         df_list_plugin_ranamed.append(df_current)
78
79     only_my_plugins_renamed = union_all(df_list_plugin_ranamed)
80
81     # project only relevant column
82     # cast timestamp to int (seconds)
83     relevant_cols = only_my_plugins_renamed\
84         .select("event_timestamp", "plugin",
85                 "host", "submitter_hostgroup", "value")\
86         .withColumnRenamed("event_timestamp", "timestamp")\
87         .withColumnRenamed("host", "hostname")\
88         .withColumnRenamed("submitter_hostgroup", "hostgroup")\
89         .withColumn("timestamp", (col("timestamp") / 1000).cast
    ("int"))
90
91     return relevant_cols

```

C.3.2 Time Series/Metrics Aggregation

Next we show the code we use to summarize 10 minutes of consecutive data of a machine with the mean.

```

1 def downsampling_and_aggregate(spark, df_all_plugins,
    every_n_minutes):
2     """Create the aggregate every x minutes.

```

```

3
4     Example: if every 10 min it means that data between 15:10
5     and 15:20 are
6     summarized with the mean statistic and they will get
7     timestamp 15:20.
8     This happens for each hostname and plugin separately.
9
10    Params
11    -----
12    spark: spark context
13    df_all_plugins: PySpark DataFrame
14    timestamp | hostname | hostgroup | value | plugin
15    every_n_minutes: int
16    nr of minutes you want to aggregate. The aggregation
17    function is the
18    average.
19
20    Return
21    -----
22    aggregated_data: PySpark DataFrame
23    containing data from the plugins and cells in the
24    following form:
25    timestamp | hostname | hostgroup | value | plugin
26    NB timestamp will be in seconds
27    Value contains the average values every x minutes for
28    that host on that
29    plugin.
30    """
31    aggregated_data = df_all_plugins\
32        .withColumn(
33            'minutes_group',
34            F.col("timestamp") - (F.col("timestamp") % (60 *
35every_n_minutes))
36        )\
37        .withColumn('timestamp', F.col("minutes_group") + (60 *
38every_n_minutes))\
39        .groupBy(['hostgroup', 'hostname', 'plugin', '
40minutes_group'])\
41        .agg(
42            F.mean('value').alias('value'),
43            F.max("timestamp").alias("timestamp")
44        )\
45        .orderBy("timestamp")
46
47    return aggregated_data

```

C.3.3 Time Series/Metric Normalization

Next we show the code we use to normalize the input data with respect to every single metric.

```

1 def normalize(spark, df, df_normalization):
2     """Normalize the data given a dataframe with the
3     coefficients.
4
5     Remove the mean and divide by the std deviation to the
6     value column.
7
8     Params
9     -----
10    spark: spark context
11    df: PySpark DataFrame
12         timestamp | hostname | hostgroups | value | plugin
13    df_normalization: PySpark DataFrame
14         hostgroup | plugin | mean | stddev
15         where mean and stddev are the mean and stddev of the
16         relative plugin
17         for the specified cell.
18
19    Return
20    -----
21    normalized_data: PySpark DataFrame
22         containing data from the plugins and cells in the
23         following form:
24         timestamp | hostname | hostgroup | value | plugin
25         NB hostgroup is the the extended version
26    """
27    F.broadcast(df_normalization)
28    to_be_normalized = df.join(df_normalization, ['hostgroup',
29    'plugin'])
30    normalized_data = to_be_normalized\
31        .withColumn("value",
32                    (F.col("value") - F.col("mean")) / F.col("
33    stddev"))\
34        .select("timestamp", "hostname", "hostgroup", "plugin",
35                "value")
36    return normalized_data

```

C.4 Deep Learning Architecture Details

In this section we summarize the details of all the four deep methods with the TensorFlow summary tool that gives us information about

the structure and type of layer. An important information is also the number of parameters of the overall model that gives us an idea of its complexity.

Layer (type)	Output Shape	Param #
=====	=====	=====
dense_1 (Dense)	(None, 528)	279312
dropout (Dropout)	(None, 528)	0
dense_2 (Dense)	(None, 528)	279312
dropout_1 (Dropout)	(None, 528)	0
dense_3 (Dense)	(None, 64)	33856
dropout_2 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2080
dropout_3 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 32)	1056
dropout_4 (Dropout)	(None, 32)	0
dense_6 (Dense)	(None, 64)	2112
dropout_5 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 528)	34320
=====	=====	=====
Total params: 632,048		
Trainable params: 632,048		
Non-trainable params: 0		

Figure C.1: Tensorflow Summary of the Autoencoder Fully Connected

C.5 Image Credits

The icon representing a normal and faulty robots and the experts come from freepik from Flaticon.com

- www.flaticon.com/free-icon/robot_3398611
- www.flaticon.com/free-icon/robot_3398643
- www.flaticon.com/free-icon/scientist_560228
- www.flaticon.com/free-icon/scientist_560227

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 11, 48, 1)]	0
conv2d (Conv2D)	(None, 11, 48, 4)	136
max_pooling2d (MaxPooling2D)	(None, 11, 24, 4)	0
conv2d_1 (Conv2D)	(None, 11, 24, 2)	266
max_pooling2d_1 (MaxPooling2D)	(None, 11, 6, 2)	0
conv2d_2 (Conv2D)	(None, 11, 6, 2)	134
up_sampling2d (UpSampling2D)	(None, 11, 24, 2)	0
conv2d_3 (Conv2D)	(None, 11, 24, 4)	268
up_sampling2d_1 (UpSampling2D)	(None, 11, 48, 4)	0
conv2d_4 (Conv2D)	(None, 11, 48, 1)	133

Total params: 937
 Trainable params: 937
 Non-trainable params: 0

Figure C.2: Tensorflow Summary of the Autoencoder CNN

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 10)	880
repeat_vector (RepeatVector)	(None, 48, 10)	0
lstm_1 (LSTM)	(None, 48, 10)	840
time_distributed (TimeDistributed)	(None, 48, 11)	121

Total params: 1,841
 Trainable params: 1,841
 Non-trainable params: 0

Figure C.3: Tensorflow Summary of the Autoencoder LSTM

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 11, 6, 1)]	0
conv2d_5 (Conv2D)	(None, 11, 6, 4)	136
max_pooling2d_2 (MaxPooling2)	(None, 11, 3, 4)	0
conv2d_6 (Conv2D)	(None, 11, 3, 2)	266
max_pooling2d_3 (MaxPooling2)	(None, 11, 1, 2)	0
flatten (Flatten)	(None, 22)	0
dense_8 (Dense)	(None, 50)	1150
dense_9 (Dense)	(None, 20)	1020
dense_10 (Dense)	(None, 11)	231
Total params: 2,803		
Trainable params: 2,803		
Non-trainable params: 0		

Figure C.4: Tensorflow Summary of the Forecast CNN

C.6 Disclaimer: Connection with previous work

Being this thesis the product of a Double Degree programme between Politecnico di Milano and TU Berlin in the frame of EIT Digital, it can be consider an improved version of the thesis discussed at Politecnico di Milano on the 2nd of October 2020 by the same author Matteo Paltenghi.