POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

GENERIC APPROACH TO ANOMALY PREDICTION ON AN ENTERPRISE SYSTEM

Relatore: Prof. Pier Luca Lanzi

Tesina di Laurea di:
Nicola Ghio, matricola 854619

Anno Accademico 2016-2017

A chi mi è stato vicino.

Abstract

Enterprise systems are put under heavy pressure by users, if the underlying configuration is not adequate they often incur into periodic downtime which can become a great expense in terms of time, resources and customer satisfaction. Nowadays there is no general approach to this problem since these kind of system are very heterogeneous in terms of quantity and quality of the information they provide. For this reason, in this thesis we propose a generic approach to implement an anomaly prediction algorithm to switch the paradigm of the response to a system fault from a reactive to a proactive one. This algorithm works by exploiting the infrastructural metrics collected by the system along with the applicative logs produced by the application running on it. Our experiment has been conducted on a small test environment for which the results are not satisfying, therefore this thesis is focused on describing our approach by motivating the reasons of the decisions we take in order to serve as a guide for the reader who wants to implement an anomaly prediction algorithm on its own system.

Sommario

I sistemi informativi delle grandi aziende sono sottposti a grande sforzo da parte degli utenti, se la configurazione sottostante non risulta adeguata spesso incorrono in periodici disservizi che possono risultare in un grande costo in termini di tempo, risorse e soddisfazione del cliente. Oggigiorno non esiste un generico approccio a questo problema perchÃ¨ questi sistemi sono molto eterogenei in termini di quantitÃ e qualitÃ delle informazioni che forniscono. Per questo motivo, in questa tesi proponiamo un generico approccio per implementare un algoritmo di predizione delle anomalie per modificare il paradigma di risposta a un guasto di sistema da reattivo a proattivo. Questo algoritmo funziona sfruttando le metriche infrastrutturali collezionate dal sistema assieme ai log applicativi prodotti dalle applicazioni in esecuzione sul sistema. I nostri esperimenti sono stati condotti utilizzando un piccolo ambiente di test per i quali i risultati non sono soddisfacenti, per questo motivo questa tesi Ã¨ concentrata nel descrivere il nostro approccio motivando le ragioni e le decisioni che abbiamo preso in modo da servire come guida per il lettore che voglia implementare un algoritmo per predire le anomalie sul proprio sistema.

Contents

Chapter 1

Introduction

Companies often uses large-scale clusters of machines to host their services which rely on the underlying infrastructure to achieve maximum performance. Unfortunately, infrastructural anomalies are a common issue in an enterprise context, having the possibility to predict them would mean sparing a considerable amount of time and resources while improving the customer experience.
In this thesis we propose a custom solution anomaly prediction solution that, starting from the logs of an applications running on a cluster of machines and by its infrastructural metrics is able to predict the next infrastructural anomaly.
The demand for this algorithm has been raised by an important Italian bank that suffered constant downtime of some application used by internal employee of the bank, which caused severe delays in terms of services offered to the customer and sometimes the necessity of overtime labor by the employees themselves in order to restore the service. These issues caused also the need of intervention by highly skilled technicians which had act as soon as possible: a work that could require up to hours of extensively paid time. We were asked to find a solution for this problem by, if possible, building an algorithm that could work also on other application used by the customer. For this reason, we focused our work to be as general as possible in such way that it could work agnostically with respect to the machine and the application that is running on it.
The application given us for developing the model was the family banker portal, an important service for the business of the customer which incurred into severe faults every few days. A fundamental point of this work is that the machine that were given to us for developing our solution were not the ones on which the family banker portal actually run, neither we could access

the data coming from it. We were instead given access to a single machine of the cluster using for testing purposes of this service. This was a further motivation to approach the problem in a general way, such as we had to think of a solution that could be deployed without knowing a priori to the type of underlying machine since we were guaranteed that test and production machine would differ one from the other but we could not know how much.
To develop this solution, we explored two main fields: anomaly detection and log analysis therefore the elaborate is structured in the following way: first it gives an analysis of the current state of art for both the fields of study, then it explains in detail the problem and the methodologies used for solving our specific situation and then it concludes with the analysis of the results along with some considerations about possible future development.
The algorithm has been developed for a customer during a non-curricular stage at Oracle Italia and, for policy issues, we divulgate as many information as we are allowed to.

This thesis is organized as follows. In Chapter 2 we give an overview of the current state of the art for the techniques we used to build the algorithm, anomaly detection and log parsing, followed by a paragraph dedicated to anomaly prediction. In Chapter 3 we describe in details the enterprise system we worked on and the information it provides about its health status. In Chapter 4 we explain step-by-step our approach, the decisions we made and their motivation. We first introduce anomaly detection, then we show how we parsed the applicative logs and we conclude showing how we combined the outputs of the previous steps. In Chapter 5 we finally show the results of our approach and the reasons for which our algorithm performed poorly.

Chapter 2

State of the art

In this chapter we perform an analysis about the current state of the art in the challenges that characterize our work, these are log analysis and log parsing, anomaly detection and anomaly prediction. For each of them we give an overview of the main techniques used to face them highlighting how they can be interfaced with our problem.

2.1 Log Analysis

We start by giving the definition of a log: for "log" we intend an automatically produced and time-stamped documentation of events relevant to a particular system. Virtually all software applications and systems produce log files.
In our particular case we had to analyze applicative logs, such as the ones produced by an application which have the characteristic of containing a huge amount of information, according to how they have been programmed they can contain general information, events, warnings and alerts.
Applicative logs are used by developers and system experts for the analysis of an application in order to monitor its status and gather insights about any issue that occurred along time. Their nature and the information they carry can be very heterogeneous depending on the application they refer to. Independently from their nature however, applicative logs share a common characteristic: they are all composed by fixed part, defined by the developers, and variable parts that depends on the situation. To give an example, a log saying *"Connection opened on port 3343"*, communicating the beginning of a connection, is composed by a fixed part *"Connection opened on port"* and a variable part such as the port number *"3343"*, therefore one must

decide how to deal with this kind of information.

2.1.1 Log Parsing

Another important characteristic of logs is that the majority of them is given in free text format, this raises the problem of parsing them. There are two main approaches for parsing logs:

- Clustering based approach: (LogSig [1], log key extraction [2]) calculate distances between logs using different techniques, like edit distance or other custom distance measure, in order to group them and finally they generate a template for every cluster found and they use it as a representation for any log belonging to that group for any next step.
- Heuristic-based approach: (SLCT [3], iPLOM [4]) count the occurrences of each word on each position in the log and then select frequent words to use them as event candidate. Some of this candidate are finally chosen to become log events. There are other minor approaches, such as the one proposed by Loghound algorithm [5] that discovers frequent patterns from event logs by utilizing a frequent itemset mining algorithm, for example using Apriori algorithm.

All the above mentioned methods were of great inspiration for our work but they did not completely fit our needs. In our case we explicitly knew some information about the log, for example its severity level, but the rest of the information had to be extracted to a more abstract level. For this reason, we implemented a log parsing algorithm based on a clustering techniques exploiting the tf-idf algorithm [6] in order to find more relevant word. This approach is discussed in details in chapter 3.

2.1.2 Feature Extraction

After the parsing phase is concluded, logs must be transformed in something that can be used by a machine learning algorithm such as a feature vector. To do so, logs are sliced in sequences so that can be grouped using different techniques, this process often takes the name of windowing. Independently of the technique used, at the end of this phase we have a matrix, called event count matrix, which describes the log history over time. This matrix has a column for each cluster found in the parsing phase and a number of rows equal to the number of windows.
For selecting the number of windows there are three main techniques:

- Fixed window: is a timestamp based method which requires the tuning of only one parameter: the window size each set of log is grouped in a window of fixed size in such way that each log appears in only one window. Figure 2.1 explains this concept.

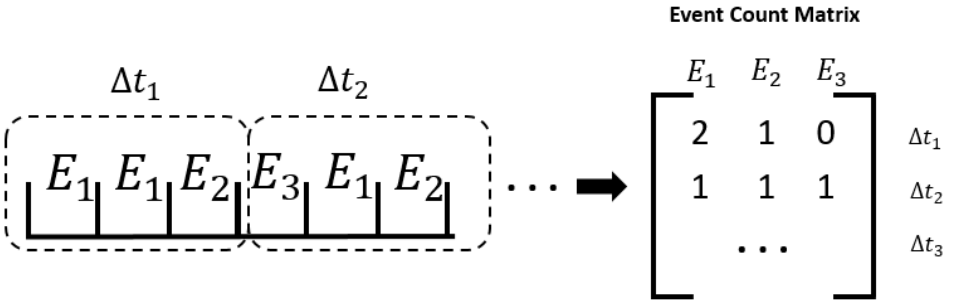

Figure 2.1: Fixed window feature extraction: the two time windows are completely disjoint

- Sliding window: is another timestamp method which, differently from fixed windows, requires the tuning of two parameters: the window size and the step size. Usually step size is smaller than window size so that windows are overlapped, this means that the same group of log (defined by the timestamp step size) could be duplicated in different consecutive windows. Figure 2.2 explains this concept.

- Session window: differently from the previous techniques, this one is based on an identifier representing the of session that is going to be considered. This is used for example in distributed system for example in Hadoop where a session takes place on different blocks and there is the necessity to group together all its logs.

Given this high variety of methods, we understood that using a sliding window was the better choice for our task because it gives the possibility to the better model the progressive growth of anomalies in time when related an anomaly score. This is better discussed in the following chapters [19].

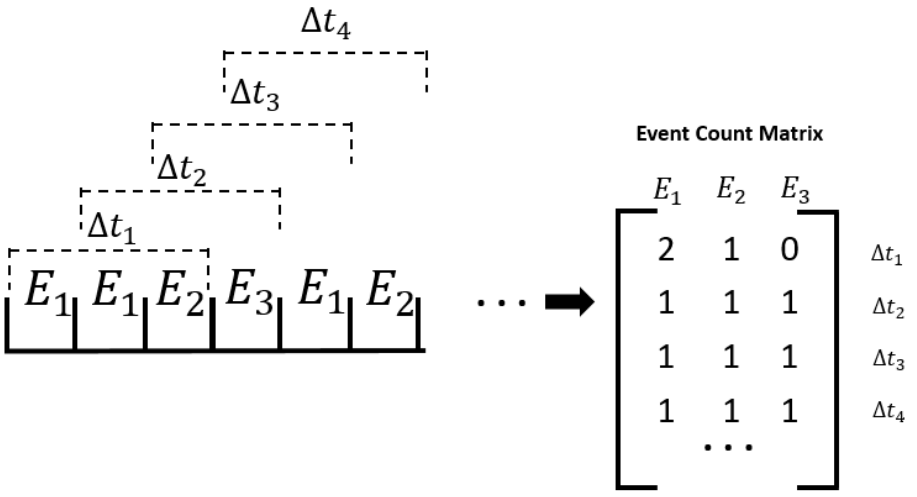

Figure 2.2: Sliding window feature extraction with a width of 3 time steps and a pace of 1

2.2 Anomaly Detection

To build an anomaly prediction algorithm we first need to define what is an anomaly in the system we considered, in order to do so we implemented an anomaly detection algorithm.
Anomaly detection (sometimes called outlier detection) is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset. The importance of anomaly detection is due to the fact that anomalies in data translate to significant, and often critical, actionable information in a wide variety of application. Typical use cases for this task includes fraud detection in credit card transaction for example by noticing an unexpected high withdrawal of money from a location which does not correspond to the normal behavior of the customer, intrusion detection on a Linux system and also cases in system health monitoring in which for example an anomalous MRI image may indicate the presence of malignant tumors.

2.2.1 Taxonomy of Anomalies

An important aspect of anomaly detection is understanding the nature of the anomaly we are going to face; they can be classified into the following three categories:

- Point anomalies: if an individual point can be considered anomalous with respect to the rest of data then we have a point anomaly. This is the simplest and most common kind of anomaly and it is the one we are going to face in this work. An example of this anomaly is the above mentioned credit card fraud using as only dimension the amount, a point anomaly occurs when we record a monthly withdrawal of 1000$ while for the rest of the year the usual withdrawal is 100$ per month. An example is shown in Figure 2.3.

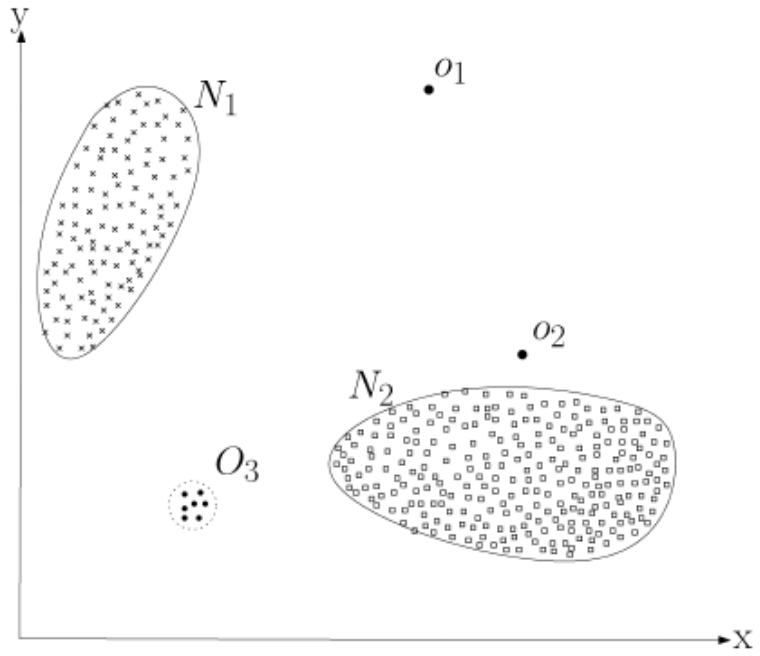

Figure 2.3: A simple example of anomalies in a two-dimensional dataset.

- Contextual Anomalies: If a point is anomalous in a specific context but not otherwise. They are also referred as conditional anomalies [7]. For this particular kind of anomaly, context has to be defined and become part of the problem formulation. Each data instance then is now composed in two parts:
  - Contextual attributes: used to describe the context the data is in. An example could be time attribute in a time series data.
  - Behavioral attributes: the set of non-contextual characteristic of the instance, using the time series example this is the actual value of the point. To use the same example of credit card fraud

mentioned above, a contextual attribute could be the time of withdrawal. Suppose an individual has a usual withdrawal of 100$ per month except on December because of Christmas where the amount reaches up to 1000$, a withdrawal of 1000$ in July is considered a contextual anomaly. An example is shown in Figure 2.4

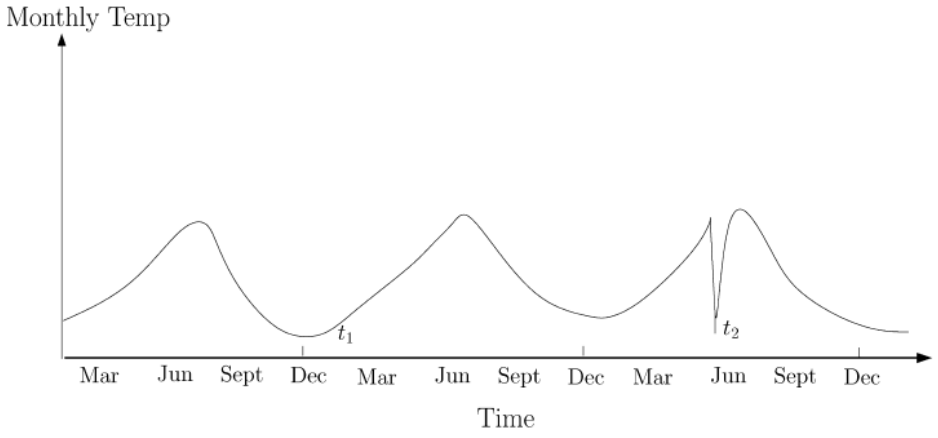

Figure 2.4: Contextual anomaly t2 in a temperature time-series. Note that the temperature at time t1 is same as that at time t2 but occurs in a different context and hence is not considered as an anomaly.

- Collective Anomalies: If a collection of related data instances is anomalous with respect to the entire data set, it is termed a collective anomaly. The individual data instances in a collective anomaly may not be anomalous by themselves, but their occurrence together as a collection is. An example could be the one in Figure 2.5 picturing the rhythm of a heart in an electrocardiogram. If taken together the highlighted points compose a collective anomaly because the same low value exists for an abnormal amount of time while, if taken individually, such low value is not a point anomaly. It emerges then that collective anomalies can emerge only in a dataset where instances are related.

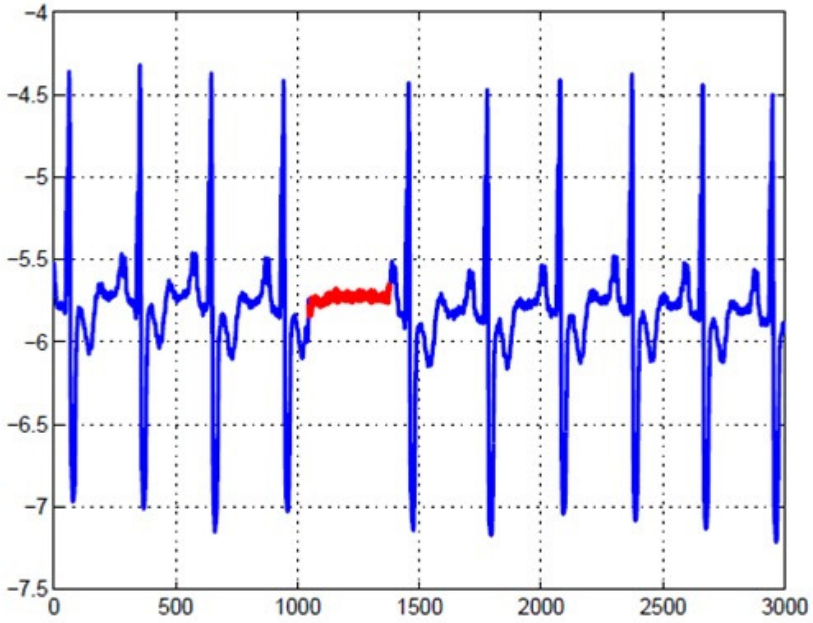

Figure 2.5: Collective anomaly corresponding to an atrial Premature contraction in a human electrocardiogram.

2.2.2 Anomaly detection techniques

Anomaly detection is often performed by using a rule-based model which sends alerts when a certain attribute exceeds a previously set threshold. This approach is not so reliable since it often does not take in account all the attributes at once and, if it does so, it could produce a significant overhead. For this reason, machine learning is often used for this task, we give now an overview on hot it is used. There are three main categories of machine learning anomaly detection techniques:

- Supervised anomaly detection: there techniques are trained in supervised mode and they assume the availability of a training data set that has labeled instances for normal as well as anomaly classes. Typical approach is to build a predictive model which output is a label categorizing each new input as anomalous or not or specifying a probability. There are two typical problems these methods incur into: the first is the high imbalance between the two class (there are lot less anomalies with respect to normal data) and the other one is finding representative labels for the anomaly class since some anomalies are more severe than others. Usual models for this task are Neural network, Bayesian net-

works, Support Vector Machines and Rule-based algorithms, another important contribution is given by K-Nearest-Neighbor techniques.

- Semi-supervised anomaly detection: Techniques trained in semi-supervised mode assume the availability of a training data set that has labeled instances only for the "normal" class. Such models usually are trained to recognize the normal behavior and are then used to identify anomalies. An example could be the detection of anomalies on an aircraft where incidents are difficult to model. These models are quite rare because of the lack of datasets available since it is difficult to obtain a training data set that covers every possible anomalous behavior that can occur in the data.

- Unsupervised anomaly detection: Techniques that do not require training data and thus are more widely applicable. These methods make the implicit assumption that normal instances are far more frequent than anomalies. Clustering methods are part of this category, going deep into this particular methodology would require an entire essay, for our work it is enough to say that many are based on the concept of local outlier such as a point that is anomalous with respect to his neighbors.

As emerged from this resume of topologies and techniques the world of anomaly detection is very wide and heterogeneous [20]. We identified that in our situation the best representation was given by point anomalies and we used an unsupervised technique, named local outlier factor [8][9] which is better discussed in chapter 4.

2.3 Anomaly Prediction

The final step of our work is to put together the output of feature extraction of the applicative logs and the anomalies found with an anomaly detection technique to train an anomaly prediction model.
Anomaly prediction raises from the need of switching from a reactive anomaly management system, that could bring to prolonged hours of service downtime, to a proactive that take preventive actions on the system component beforehand. As in the previous analysis on anomaly detection it emerged that the nature of anomalies can be very heterogeneous, finding reliable works on anomaly detection in a cluster of machines is not an easy task. The actual state of the art is composed by the following families of anomaly

prediction techniques, sometimes used in combination one with the others. Those families are:

- Context-aware anomaly prediction model: [10] that takes into high consideration the context in which the underlying cluster of machine is working. For example, in the morning a webserver often receives a higher workload than in the evening. A difficult task that these models face is the one of identifying which variable denote each context and how they change over time but once they are discovered, these models are able to perform a much more precise prediction since it becomes possible to train the more appropriate model for each context.

- Markovian anomaly prediction model: capture the changing patterns of different measurement metrics that are used as features by a classifier [11][12]. These algorithms are a good way to model sequences of events but they run into minor problems. To model the states of the machine it is necessary to discretize continuous attributes into a finite number of bins which could lead to lose some information. Another issue that they must face is that not all attributes follow the Markov property, for example if an attribute value exhibits a sinusoidal pattern the model has to rely on both the current value and the previous value to determine whether the attribute value on an increasing slope or a decreasing slope. To solve this problem some tricks are used like using two dependent Markow chains.

- Monolithic anomaly prediction model: algorithms that see the cluster as a unique entity. These are widely used because they are easier to implement but suffer from two throwbacks: they cannot distinguish which components are attributed to the performance anomaly, secondly the prediction accuracy of one monolithic model is significantly worse since the attribute value prediction errors accumulate as the attributes of all components is included into only one model [10].

- Per-machine anomaly prediction model [12]: builds a single model for every machine analyzed in the cluster. In such way it is possible to overcome the limitations which characterized the previous technique but lose the efficiency of considering the cluster as a collection of entities.

All the above mentioned techniques perform online anomaly prediction. Each of these methods have been developed for specific situations and, of course, could not be just taken and reused for our goal. In our situation,

since this work has been done for a customer, we had constraints regarding the simplicity of the model and the time to deliver it. After having considered all the suggested approach we concluded that for our goal a per-machine anomaly prediction model could be the one that best fitted our needs because of it simple but still valid logic.

Chapter 3

Definition of the problem

In this chapter we are going to explain in details the problem we had to face. We begin by giving an overview of the actual situation of the customer and the issue he incurs into every day, then we proceed by carefully explaining the information available and of the motivations that lead us to think about an anomaly prediction solution.

3.1 Overview of the situation

The customer is one of the biggest banks in Italy which everyday activities, like many other companies, is strictly related to its software applications. These go from internal-only usage applications, like an analytical CRM, to the ones accessed by its customers like the home banking portal. Each of these applications has an impact level on the operational capacity of the bank, some of them are of vital importance while others are more or less negligible. To give an example, if the above mentioned home-banking portal becomes inactive, even for few minutes, thousands of customer would incur into a disruption of service which, of course, would leave an unhappy memory about their user experience. Since no company wants unhappy customers, this has to be considered of maximum importance because repeated disservices are to be avoided as much as possible.
Other applications, instead, are not so crucial for the reputation of the bank but are still fundamental for the correct functioning of the bank ecosystem and if they do not respect certain availability conditions they may cause far worse damage than unhappy customers. One of those application is the portal used internally by family bankers which is also the target of our work. Family bankers are freelancers that represent the primary link between the bank and its customer also embodying the role of financial advisors, they

use this portal to manage their everyday work. Even if a disservice of an application used by internal-employees only is not as severe as one occurring on an application used by customers, a periodic downtime could lead to significant delays in terms of customer service delivery since family bankers are part of the core functionality of the bank.
Up to the current situation, each time an applicative fault occurs, a quick intervention of technical experts is needed to restore the service: this could require up to hours of intense, manual search to look for the root cause of the fault. Therefore, for all the above mentioned reasons, every fault results in a loss of resources in terms of time, money and efficiency.
Our response to find a solution to this problem has been suggesting a switch of paradigm: from a reactive anomaly management system to a proactive one.
There are two kinds of faults: infrastructural, such as caused by a hardware issue like a pitch in the CPU response time, or applicative, such as caused by some logical error like for example a Java exception. These two faults are correlated one with the other since an infrastructural fault could cause an applicative fault and vice versa. Moreover, it has also to be considered that some light infrastructural anomalies could be the cause of a greater one in the future and such reasoning is valid also when talking about applicative anomalies.
Unfortunately, we were asked to find and implement a quick solution because of business requirements so, since it would take a considerable amount of time and resources to develop an anomaly prediction system that considered all kinds of faults and every possible correlation between them, to meet them we could not develop a full solution. For this reason, we decided to develop an anomaly prediction system that could forecast infrastructural anomalies starting from the applicative logs of the family banker portal by keeping an eye on how finding a generic solution that could fit other applications as requested by the customer. In this thesis we present how we implemented this algorithm and, since this is just one part of the final model, for the sake of completion we discuss on how we would have proceeded with the implementation of the final model.

3.2 System Overview

To understand the problem and our solution, it is mandatory to explain how the system is built and how its components communicate. The entire portal of family bankers is built on top of a WebLogic cluster of four managed servers on Exalogic hardware. If all of this sounds confusing it is because

we are talking entirely about Oracle products so we proceed step by step. Oracle WebLogic Server is a scalable application server used for building and deploying enterprise Java EE applications with support for new features for lowering cost of operations, improving performance, enhancing scalability and supporting the Oracle Applications portfolio [13]. It provides a standard set of APIs for creating distributed Java applications that can access a wide variety of services, such as databases, messaging services, and connections to external enterprise systems. Each WebLogic Server has an administration domain which is a logically related group of WebLogic Server resources. A domain include a special WebLogic Server instance called the Administration Server, which is the central point from which it is possible to configure and manage all resources in the domain. Usually, one configures a domain to include additional WebLogic Server instances called Managed Servers. Web applications, Enterprise Java Beans, and other resources are deployed onto the Managed Servers and the Administration Server is used for configuration and management purposes only.

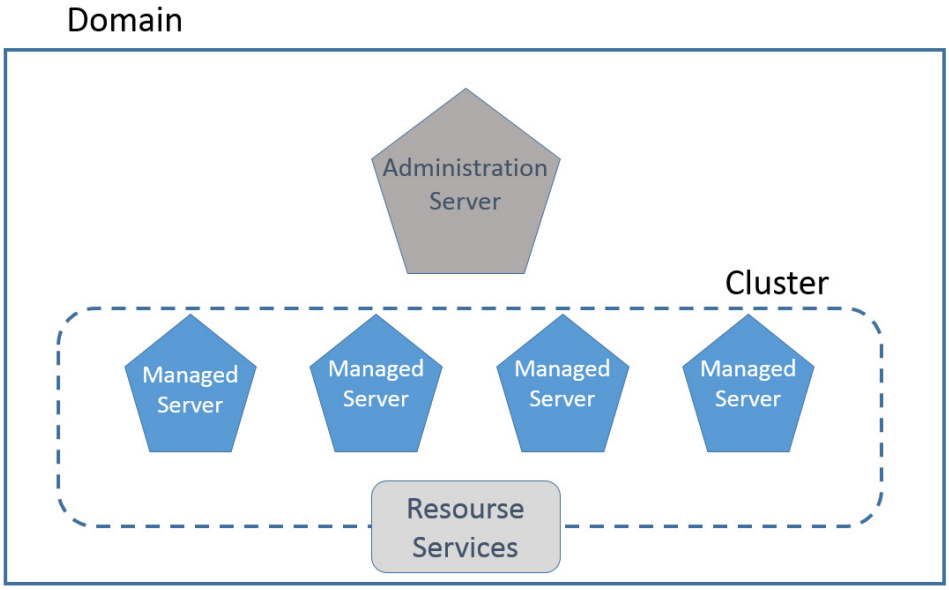

Figure 3.1: Abstract view of the WebLogic domain.

Multiple Managed Servers can be grouped into clusters, this allows multiple Managed Servers to operate as a single unit to host applications and resources while enabling balance loads and provide failover protection for critical applications. Figure 3.1 shows the abstract view of the WebLogic

Model Name	Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz
N. cores	4
Standard CPU Frequency	2.70 GHz
Max CPU Frequency	3.50 GHz
Cache	30 MB Smart Cache
Bogomips	5387.21
Address size physical	46 bits
Address size virtual	48 bits

Table 3.1: Processor characteristics

Server architecture, note that The domain configuration also includes information about resources and services associated with applications hosted on the domain [14].
Each managed server is installed on top of an Exalogic Virtual Machine which is a complete hardware and software platform for Enterprise applications delivered as pre-assembled building blocks in order to be easy to deploy and operate. It is an assemblage of storage, compute, network, operating system, and software products which is not proprietary so that one could integrate it with other services at will. [15]. Each Exalogic Virtual Machine is identical and is assembled with four processors which characteristics are listed in Table 3.1 along with 16 GB RAM and is running Red Hat Enterprise Linux Server release 6.5 (Santiago) operating system.

These four Virtual Machines are connected together by Oracle load balancer called Oracle Traffic Director, this allows the virtualized cluster to be seen as a unique machine from the application point of view as shown in Figure 3.2.

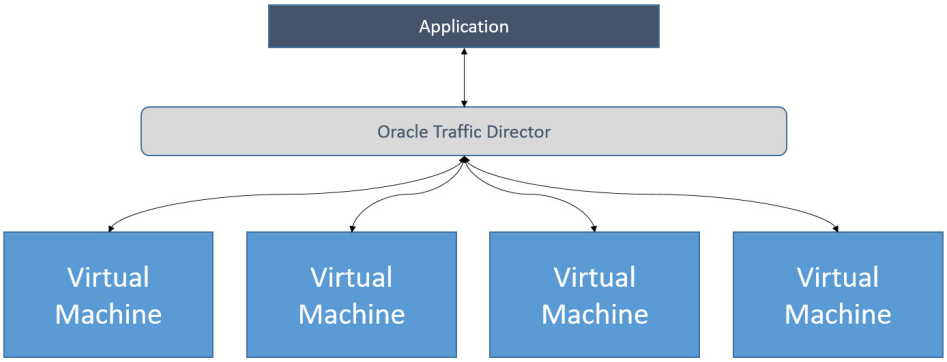

Figure 3.2: Relation between the application and the VMs.

3.3 Log Collection

Each virtual machine of the cluster hosts the same application which produces logs that are collected by Splunk.
Splunk is a log collector used by technical experts to collect, monitor in real time and analyze machine data from any input [16]. It is a very versatile tool, able to be integrated in different kind of systems with ease, used by many companies to collect and monitor their application. Splunk allows extensive, efficient research on the whole history of the application logs since its reports are very detailed and well structured, allowing to browse logs filtering by application, time and host.
The application is installed on top of the above described cluster and is integrated with Splunk as shown in Figure 3.3 which shows the abstract information flow. The single arrow indicates that Splunk sees the application as a single entity producing logs continuously while it stays in polling. Whenever the application produces a log it be almost immediately visible from the Splunk console.

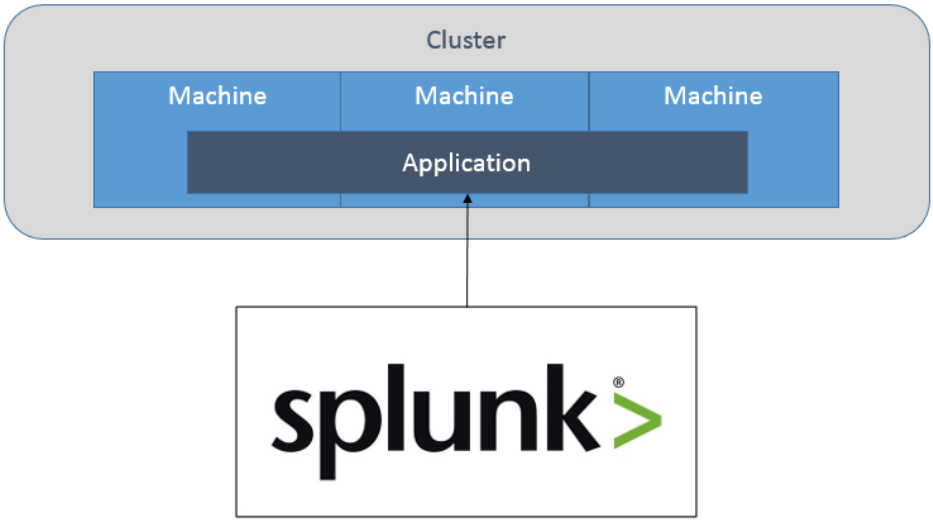

Figure 3.3: Abstract view of the information flow between the application and Splunk.

Underhand, instead, the actual information flow between the application and Splunk is a bit different: since the application is distributed on the cluster it produces separate logs for each machine it is hosted in. Each VM hosts a Splunk agent which reads the logs produced by the application and sends it to the Splunk Server that merges them and stores it into an index database. This flow is shown in Figure 3.4.
Since Splunk is very versatile, it is able to gather any kind of log from the machine, not only the applicative ones anyway, since the cluster is composed by Oracle machines, in our problem Splunk is set up to collect only the applicative logs and does not consider the infrastructural metrics like CPU usage, amount of stored RAM and so on. This information is collected by a proprietary software called Oracle Enterprise Manager(OEM) 12c [17] which is used by Oracle to monitor all its products. Differently from Splunk, OEM sees every machine of the cluster as a separate entity since the family banker portal which is running on them is not an Oracle software. Each machine has a health monitoring software installed, called Oracle Enterprise Manager Cloud Control, which collects infrastructural metrics information and pushes them to OEM, from here they are stored in an Oracle Database table. Figure 3.5 shows the logical view of this information flow.

It should be now clear that the cluster is monitored on two different levels,

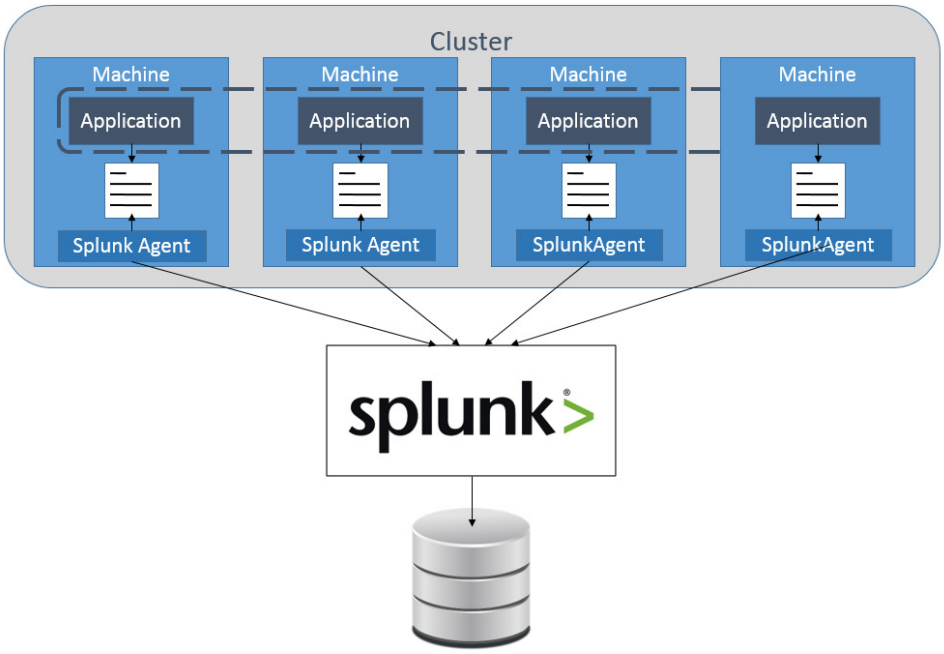

Figure 3.4: Information flow between the application and Splunk.

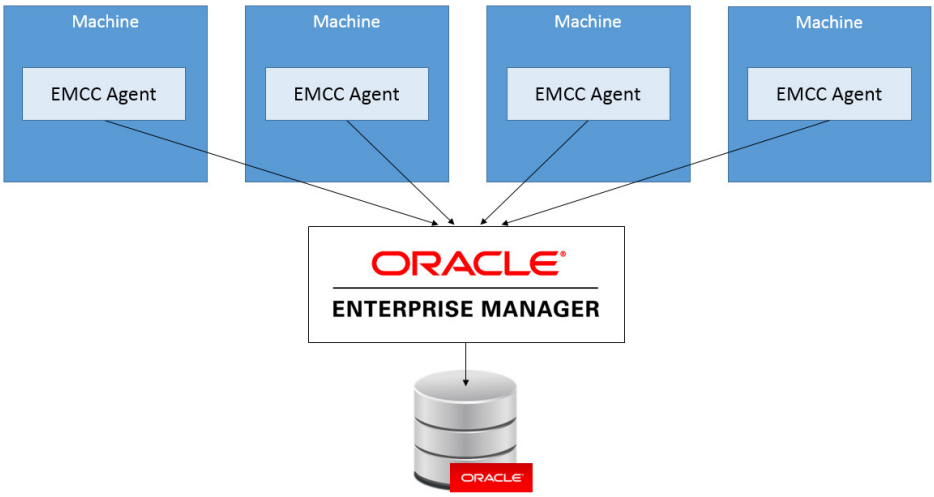

Figure 3.5: Infrastructural information flow.

infrastructural and applicative, and that the information is collected into two different storages which do not communicate between them.

3.4 Log overview

Now that we have described which kind of information we are dealing with and its flow, we are going to explore it in details, starting by Splunk applicative logs and then exploring infrastructural logs collected by Oracle Enterprise Manager. In this chapter we are going to focus on the format of those kinds of logs by showing data from sample machine which run another Oracle application, instead of directly showing the one from which we extracted data. We prefer this approach because we had to work agnostically with respect to the final target machines and applications on which our algorithm will run. This reasoning is valid even if we focused only on the family banker portal since we were given just a test environment which, even if it run theoretically the same application, is guaranteed to produce different logs in terms of level of details and even information gathered.

3.4.1 Application log format

We start by describing the format of application logs which come from a simple extraction on a CSV file of all the logs of the target application. This file is composed by attributes which give us a wide range of information in a fine grained fashioned way since in Splunk logs are generated in the order of milliseconds. These fields are automatically filled by Splunk from the applicative log using a custom parsing logic which has been predefined by the technical experts monitoring the family bank portal. Depending on how it has been set up, Splunk could produce reports composed by hundreds of fields, some of which are often nulls and thus any log analysis algorithm has to take in consideration this characteristic. The percentage of not-null values per attribute on a sample application is illustrated in Figure 3.6 representing on the how many fields contains which percentages of not-null values using bins of 0.1 percentage.

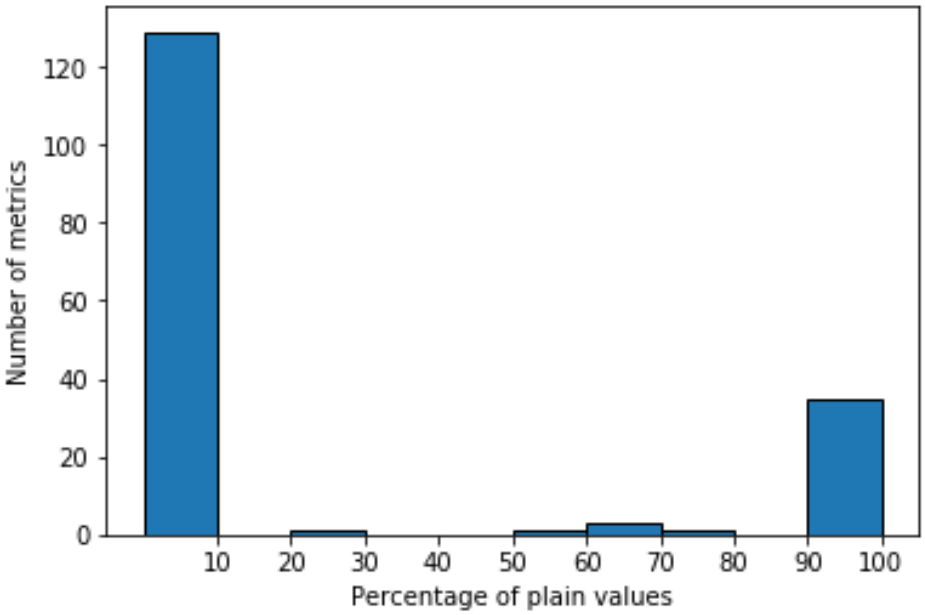

Figure 3.6: Distribution of plain values into the fields of the applicative logs.

In this case more than 120 fields are almost always set to null while less than 40 are filled at least 90% of the time.
This information suggests that developing an algorithm focused on the single fields could lead to problem since there is no guarantee of how many nulls they contain or, even, if they are recorded the same way in another application or, worse, recorded at all. Fortunately, Splunk is set in such a way that it keeps the original unparsed applicative log in one of the fields called "_raw" that, if used properly, could be able to fill this apparent lack of information. This field, by nature, can never be null, unless the log itself is empty and thus is the focus of our analysis as better explained in chapter 4. An Example of "_raw" log is shown in Figure 3.7.
The entire log is not reported because it would not give any additional insight. The main information it contains is about the timestamp the log occurred, the environment in which it was running along with its host, the application this log refers to (remember that Splunk is able to monitor different applications at the same time), the position in the code that generated the logs and a lot of minor and situational information.

Figure 3.7: Example of applicative log.

The "Info"tag suggests that the family banker portal is an application that uses log4j [18] to implement its logging system. Log4j is a part of the Apache Foundation Project and is a standardized Java library which allow access to an optimal logging system to monitor a Java application. One of its characteristic is that it associates to every log a severity level, here listed in Table 3.2 in decreasing order of severity. Since these levels given by the developers of the application, they constitute a fundamental piece of information because they already express the importance of the log giving us a strong prior information.

3.4.2 Infrastructural log format

Infrastructural logs are collected by querying the Oracle database table in which they are progressively stored. Oracle Enterprise Manager offers several monitoring views depending on the level of details required by the user. There are mainly two kind of views:

- History of hourly or daily aggregation of every metric from the beginning
- History of most recent metric showing all records for each of them

Overall, since machines can be of different types, the number and the kind of infrastructural metric differs from one machine to another. To give an example of the level of details that OEM is able to give, for the machine on which the sample application mentioned before run 82 metrics are reported. They go from the amount of allocated memory (both in bytes and in percentage), the CPU load, information about I/O activity and much more.

Level	Description
OFF	The highest possible rank and is intended to turn off logging.
FATAL	Severe errors that cause premature termination. Expect these to be immediately visible on a status console.
ERROR	Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
WARN	Use of deprecated APIs, poor use of API, 'almost' errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong". Expect these to be immediately visible on a status console.
INFO	Interesting runtime events (startup/shutdown). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
DEBUG	Detailed information on the flow through the system. Expect these to be written to logs only. Generally speaking, most lines logged by your application should be written as DEBUG.
TRACE	Most detailed information. Expect these to be written to logs only.

Table 3.2: Log4j severity levels

Differently from Splunk, however, infrastructural metrics are emitted asynchronously one with respect to the others so Oracle Enterprise manager does not produce each time a log containing all of them at once but rather produces a log containing the name of the metric and its value each time it is emitted.

On a deeper analysis we found out that metrics are collected in discrete, predefined, intervals. Figure 2.8 shows this characteristic, in this case most metrics are collected in a relatively low amount of minutes, up to 15, while others are collected up to once a day or more. The most collected metrics are the one regarding memory, CPU and disk usage while the least collected regards system flags which are changed by developers only when needed. This asynchrony is motivated by storage issues: since Oracle Enterprise Manager has been created to manage huge clusters of machines, collecting at high rate metrics which change only once in a while would be a waste of disk memory.

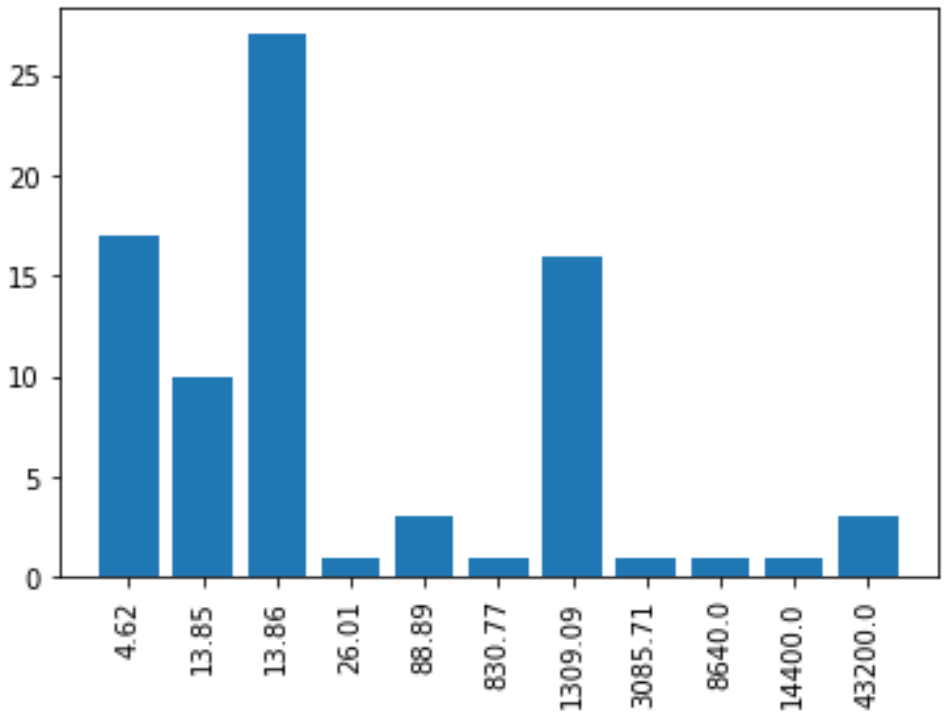

Figure 3.8: Sample rate distribution of the metrics for a machine.

Another thing that Oracle Enterprise Manager does to preserve storage is to aggregate old metrics as mentioned above. This is done by keeping a view which is a sliding windows of the records of the previous 30 days, eliminating from the view the ones that become too old. Obviously, this information is not completely lost but it is aggregated in a daily report which shows for every metric the following indicators explained in Table 3.3.

These indicators are useful for having an insight of the machine health trend during a certain day but are of little use for implementing an anomaly prediction streaming algorithm.

Metric	Description
ROLLUP_TIMESTAMP	The start and end period of the collection.
SAMPLE_COUNT	How many metrics were used for the aggregation.
AVERAGE	Average of the metric during the collecting period.
MINIMUM	Minimum of the metric during the collecting period.
MAXIMUM	Maximum of the metric during the collecting period.
STANDARD_DEVIATION	Standard deviation of the metric during the collecting period.

Table 3.3: Attributes of the aggregation Table

Chapter 4

Definition of the problem

In this chapter we are finally going to present our solution for building an anomaly prediction model which could forecast infrastructural anomalies from applicative logs. Our algorithm is split in three parts: Anomaly detection, log clustering and anomaly prediction. Anomaly detection and log clustering are completely independent one from the other and can be considered the data preparation phase of our work while in the last part, anomaly prediction, the outputs of the previous steps are merged in order to build the model. In this chapter we present our work starting by describing the behavior of the test environment given to us in terms of hardware setup and then passing to a description of the specific applicative and infrastructural logs, in the last paragraph the final solution is proposed.

4.1 Setup

Before starting analyzing our solution we have to spend some time explaining our hardware setup. As above mentioned, for policy reasons we could not access directly data from the production environment, formed by the cluster of four Virtual Machines described in chapter 3.2 but we were given a single machine which is part of the test-environment. This environment is composed by only two identical Exalogic Virtual Machines, each equipped with two processors which characteristics are shown in Table 4.1, note how the test processors are identical to the one in Table 3.1 but have half the cores. As in the production environment, each Virtual Machine has 16 GB RAM and is running Red Hat Enterprise Linux Server release 6.5 (Santiago) operating system. The test cluster follows the same logic as the production one regarding how machine are connected between them and how it relates to the application.

Model Name	Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz
N. cores	2
Standard CPU Frequency	2.70 GHz
Max CPU Frequency	3.50 GHz
Cache	30 MB Smart Cache
Bogomips	5387.21
Address size physical	46 bits
Address size virtual	48 bits

Table 4.1: Test processor characteristics

The family banker portal application installed of course is not used by the bank employees but it is used for other activities like training and testing purposes.

4.2 Anomaly detection

The first part of our algorithm is Anomaly Detection. Since our goal is to build an anomaly prediction model we first need to define what is an anomaly. Up to now there was no such concept from the customer side so we had to find an unsupervised anomaly detection technique that allowed us to model anomalies in this particular context.
Remembering that the anomalies we want to predict are the infrastructural ones and that we had access to data regarding only one of the two machines composing the cluster used for testing, we focused our analysis on the database logs regarding the infrastructural metrics of this specific VM. For this test Virtual Machine, Oracle Enterprise Manager collects 65 metrics, please note how the number of metrics collected is different with respect to the sample machine mentioned in 3.4.2 which were 82. The plot shows the distribution of the sample rate of the metrics in our test machine, shown in Figure 4.1.

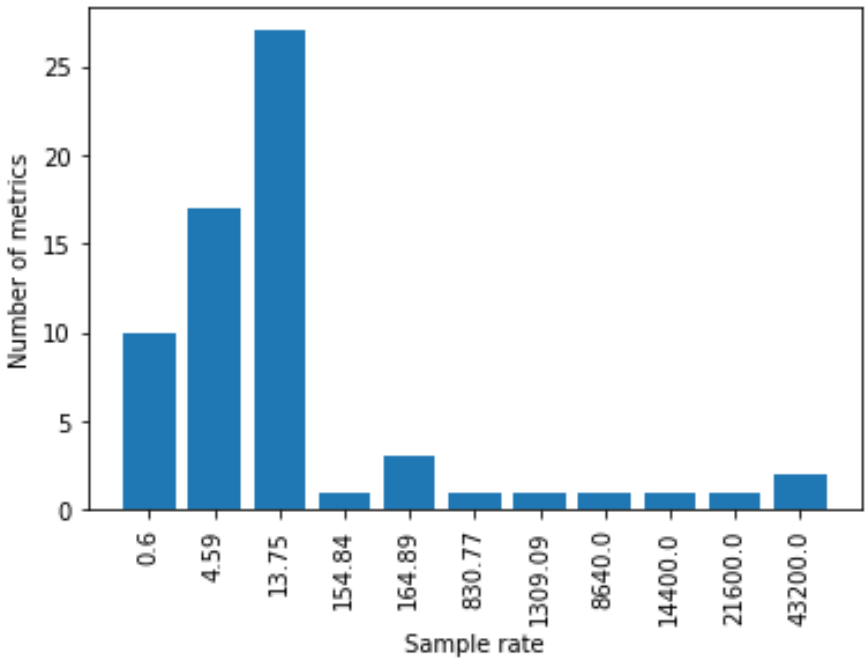

Figure 4.1: Distribution of the collection rate of the metrics in the test machine.

Note how the majority of metrics are sampled relatively frequently while others are less important metrics or system flags that changes once in a while, or did not change at all. The difference between the charts in Figure 3.8 and 4.1 has the following motivations:

- The difference in the number of metrics is motivated by the fact that for different situations it is necessary to sample different metrics.
- The differences in the sample rates is due to the fact that probably our test machine is more important than the one used for Figure 2.8 and thus metrics in it are samples more often.

This demonstrates that Oracle Enterprise Manager's output is different from machine to machine, thus reinforcing our need to find a generic approach to this problem since there is no guarantees that we will find the same metrics sampled at the same rate of the test machine on the production machine.

4.2.1 Local outlier factor

For detecting anomalies, we decided to use a method called Local Outlier Factor, simply called LOF which theoretical background is given in this paragraph. The motivation of our choice were the following:

- It is an unsupervised technique for anomaly detection.
- It is proven to work well even in the range of dimension that is the number of metrics Oracle Enterprise Manager usually collects so we could use it also in production environment where the number of metrics and their collection rate is unknown.
- It works agnostically with respect to the definition of the metrics.
- Unlike most of anomaly detection, it does not output a label "Anomalous" or "Not anomalous" but rather assigns at each point an anomaly score that can be seen as the degree of being an outlier.

To better understand this algorithm, we have to go through some definitions [8].

Definition 1 - $DB(pct, dmin)$-$Outlier$: An object p in a dataset D is a $DB(pct, dmin)$-$Outlier$ if at least percentage pct of the objects in D lies greater than distance $dmin$ from p, i.e., the cardinality of the set $\{q \in D \mid d(p, q) \leq dmin\}$ is less than or equal to $(100 - pct)\%$ of the size of D.

The above definition considers a global view of the dataset therefore the outliers fitting this definition are called global outliers. In real world there could be also objects that are outlying relatively to their local neighborhoods, particularly with respect to the densities of the neighborhoods. These outliers are regarded as "local" outliers. To illustrate this concept, consider the example given in 4.2 which pictures a simple 2-dimensional dataset.

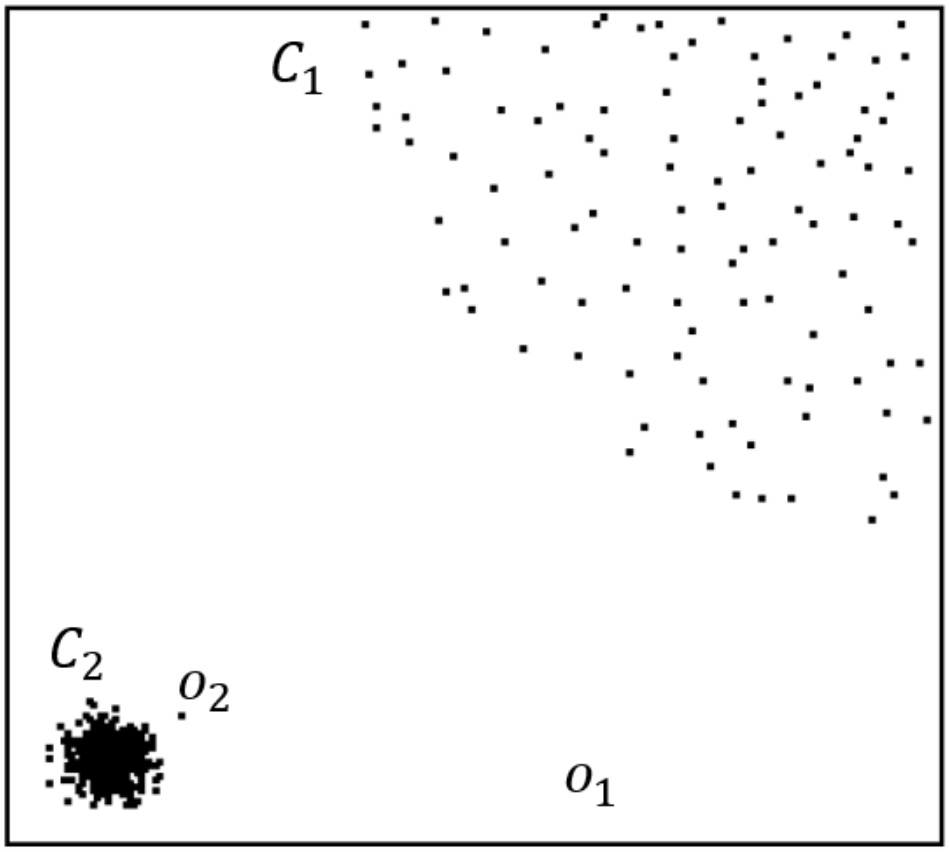

Figure 4.2: Local Outliers example.

In this example, C_2 forms a denser cluster than C_1. According to the definition of outlier both o_1 and o_2 are, whereas objects in C_1 and C_2 should not be. With the notion of a "local" outlier, we wish to label both o_2 and o_1 as outliers, this would not be possible by using just the outlier concept in Definition 1. It is in fact possible to demonstrate that there is no appropriate value of *pct* and *dmin* such that o_2 is a $DB(pct, dmin)$-*Outlier* but the objects in C_1 are not. To introduce the concept of locality we must introduce the concept of the k-nearest neighborhood as follows.

Definition 2 - k-distance of an object p: For any positive integer k, the k-distance of object p, denoted as k-distance(p), is defined as the distance $d(p, o)$ between p and an object $o \in D$ such that:

1. for at least k objects $o^{'} \in D \setminus \{p\}$ it holds that $d(p, o^{'}) \leq d(p, o)$ and
2. for at most $k - 1$ objects $o^{'} \in D \setminus \{p\}$ it holds that $d(p, o^{'}) < d(p, o)$

Definition 3 - k-distance neighborhood of an object p: Given the k- distance of p, the k-distance neighborhood of p contains every object whose distance from p is not greater that the k-distance, i.e:

$$N_{(k-distance(p)}(p) = \{q \in D \setminus \{p\} \mid d(p,q) \in k - distance(p)\} \tag{4.1}$$

These objects are called the k- nearest neighbors of p. To arrive at the final definition used in our algorithm we need to define also what is the reachability distance.

Definition 4 - reachability distance of an object p w.r.t. object o: Let k be a natural number. The reachability distance of object p with respect to object o is defined as:

$$reach - dist_k(p,o) = max\{k - distance(o), d(p,o)\} \tag{4.2}$$

This concept is illustrated in Figure 4.3. Intuitively, if object p is far away from o (e.g. p_2 in the figure), then the reachability distance between the two is simply their actual distance. However, if they are "sufficiently" close (e.g., p_2 in the figure), the actual distance is replaced by the k-distance of o. The reason is that in so doing, the statistical fluctuations of $d(p,o)$ for all the p close to o can be significantly reduced. The strength of this smoothing effect can be controlled by the parameter k. The higher the value of k, the more similar the reachability distances for objects within the same neighborhood.

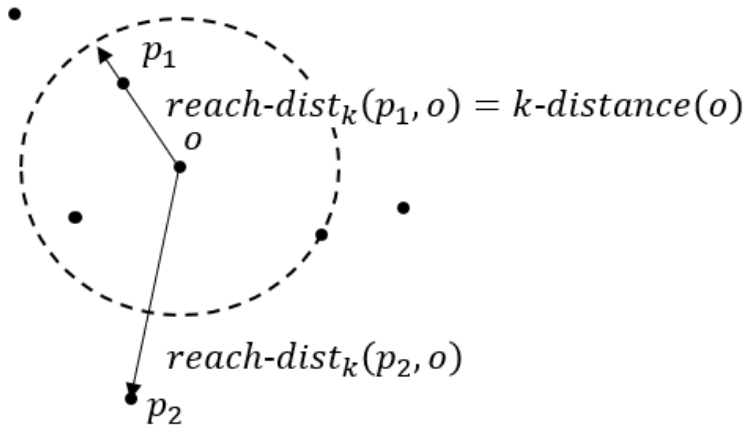

Figure 4.3: $reach\text{-}dist(p_1, o)$ *and* $reach\text{-}dist(p_2, o)$*, for* $k = 4$

The next step of this reasoning is that being an outlier is not a binary property but we could also assign an outlier factor which is the degree of outlying of a point. Usual density based cluster algorithms leverage one two parameters that define the notion of density: a parameter MinPts specifying a minimum number of objects and a parameter specifying a volume. These two parameters determine a density threshold for the clustering algorithms to operate. That is, objects or regions are connected if their neighborhood densities exceed the given density threshold.
To detect density based outliers however is necessary to compare the densities of different sets of objects, which means that we have to determine the density of sets of objects dynamically. Therefore, we keep $MinPts$ as the only parameter and use the values $reach\text{-}distMinPts(p, o)$ for $o \in MinPts(p)$, as a measure of the volume to determine the density in the neighborhood of an object p.

Definition 5 - local reachability density of an object p: The local reachability density of p is defined as:

$$lrd_{MinPts}(p) = \frac{1}{\frac{\sum_{o \in N_{MinPts}(p)} reach - dist_{MinPts}(p, o)}{\mid N_{MinPts}(p) \mid}} \tag{4.3}$$

Intuitively, the local reachability density of an object p is the inverse of the average reachability distance based on the $MinPts$-nearest neighbors of p. We have now all the elements to define our anomaly score. The outlier factor of object p captures the degree to which we call p an outlier. It is the average of the ratio of the local reachability density of p and those of p's $MinPts$-nearest neighbors. It is easy to see that the lower p's local reachability density is, and the higher the local reachability densities of p's MinPts-nearest neighbors are, the higher is the LOF value of p.

Definition 6 - local outlier factor of an object p The local outlier factor of p is defined as:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{\mid N_{MinPts}(p) \mid} \tag{4.4}$$

Local outlier factor captures the degree to which we call p an outlier, It is the average of the ratio of the local reachability density of p and those of p's $MinPts$-nearest neighbors. Basically, a point has an high LOF score if its distance from its k neighbors is far greater than the average distance between those neighbors and the points in their respective k-neighborhood. This concept is explained in Figure 4.4 using $k = 3$.

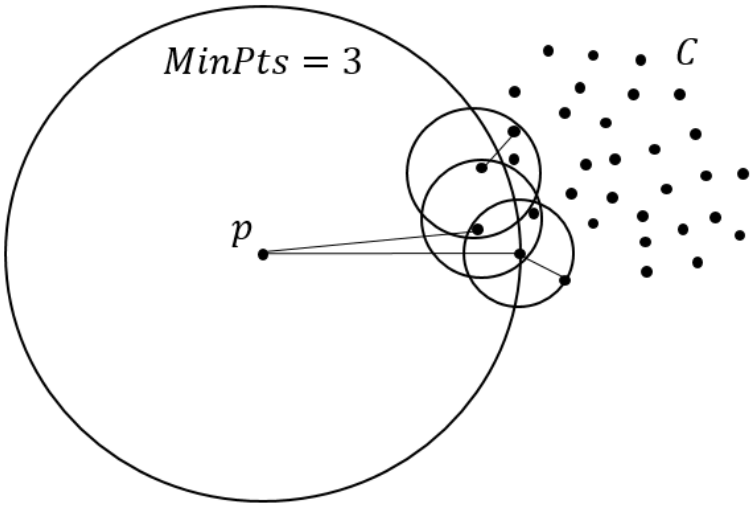

Figure 4.4: Visual representation of distances considered to calculate Local Outlier Factor.

A more critical situation presents when we have many smaller clusters: choosing k too high would lead to consider points from different clusters in the calculation of the score while choosing k too low would lead to a bad modeling of the closest cluster to the outlier which could be erroneously considered as a point of it. Usually, one choses a lower bound and an upper bound for k, then runs the model for all k in this interval and aggregates the results. For the lower bound, a general literature guideline is to consider at least 10 points in the neighborhood to remove any unwanted statistical fluctuation. Another guideline comes from the fact that k can be regarded as the minimum number of objects a cluster has to contain, so that other objects can be local outliers relative to this cluster. This value could be application-dependent. The literature shows that picking 10 to 20 appears to work well in general for most dataset. Now that we set some guidelines for the lower bound, let's see how to choose the upper bound. As the lower bound, also the upper bound carries a significate: let C be a set/cluster of close by objects, then the upper bound can be regarded as the maximum cardinality of C for all objects in C to potentially be local outliers.
There is no single way to select lower and upper bound for k, anyway after having chosen them, one proceeds by calculating the LOF score of every point in the dataset for every k in the chosen interval. As aggregation method some propose the heuristic of ranking every object with their respective maximum LOF value within the specified range, other suggests instead to use mode, mean or minimum as long as this last one is could be inappropriate since it may erase the outlying nature of the object completely.

4.2.2 Implementation of anomaly detection algorithm

To train the anomaly detection model the only dataset available was the database table hosting the last 30 days of records. This table contains 968831 records made in the following way:

- Collection timestamp
- Metric name
- Metric value
- Other parameters

This format was not appropriate for LOF algorithm which expects in input a set of 65-five dimensional points. In order to obtain this set of points we reshaped the dataset in such a way that each row contained the collection

Collection Timestamp	Metric	Value
14 nov 2017 14:54:06	cpuIOWait	0.163
14 nov 2017 14:54:06	cpuLoad	0.57
14 nov 2017 14:54:42	MemUsedPct	0.23

Collection Timestamp	cpuIOWait	cpuLoad	MemUsedPct
14 nov 2017 14:54:06	0.163	0.57	nan
14 nov 2017 14:54:42	nan	nan	0.23

Figure 4.5: Reshaping of the metric dataset.

timestamp and a column for each metric which would be filled with its value if the metric was collected in that precise timestamp and null otherwise. This concept is more easily explained in Figure 4.5 in which we make an example using just 3 metrics. After this step however we still do not have full 65-dimensional points since many dimensions are null values, for this reason we need to find a way to replace them. We assumed that in the period between two subsequent collections of the same metric it can be considered constant. This assumption well fits both the case of metrics with high sample rate and the ones with a low one: in the first case we have a high amount of different values so that developing a model imputing its change over time would be excessively complicated, in the second case we can assume that a low sample rate implies a scarcely important metric or one which changes only once in a while and thus remains constant meanwhile.
Since we are dealing with more than 65 metrics, it is unlikely that they all share the same scale. For this reason, we perform feature scaling by centering each metric and by dividing it by its standard deviation so that a variation in any metric carries the same importance.
Once prepared the dataset, we had to decide which range of k we wanted to use for running the LOF algorithm and which quantity to use for aggregation. In the previous paragraph we analyzed the importance of this choice and the consequences that a wrong decision could have. We set as

lower bound 10 as suggested by literature and we experimented on the upper bound. Since we had no idea about how many clusters of not-anomalous situation there could be and their density we tried by setting k to 20, 35, 50 and the aggregation metric first to maximum and then to median. Figure 4.6 shows the distribution of the anomaly score calculated for all possible combinations of the above parameters. The first thing to notice is how anomaly score are distributed: by nature, they cannot be less than 0.9 and they orbit around 1 while the more they get far from this value, the fewer they become.
One of the most evident property is a natural consequence of choosing the maximum: it would shift the scores to the right with respect to the median. This, would bring many points out of the "safe interval" which is generically identified as $[0.9 - 1.1]$. Another factor that seems to shift anomaly scores to the right for both aggregation quantity is k: the larger it gets the more the values are shifted. This is because the algorithm would look for the distance between more points which can happen to be in another cluster or even outlier themselves and thus they contribute to increase the anomaly score of the selected point.

Please notice that, in some cases, there were points with anomaly score reaching up to 20 but they have been dropped in the representation to make it easier to read.
To clarify the behavior of the distribution we discretized the anomaly scores in anomaly levels the following way, which results are shown in Figure 4.7.

It is clear how, overall, the median tends to label fewer anomalies than the maximum for different values of k and that it is more stable over time. This characteristic has lead us to choose the median as aggregation metric, while for the number of nearest neighbors to use as an upper bound we decided to set k=20 because there were no substantial differences with respect to other larger values of k and it requires few resources to compute.
What is left now is to validate the model. Unfortunately, we could not find a methodology to do it with our current data. A common approach for testing anomaly detection algorithms is giving in input to the trained model objects which were labeled as not anomalous for another situation (that could be in our case a different machine running another application). This could not have been done because of the lack of other data sources. For this reason, we must content to check some of the point labeled as anomalous with others who are not. In Figure 4.8 we perform a manual check to see if the results of the anomaly detection algorithm.

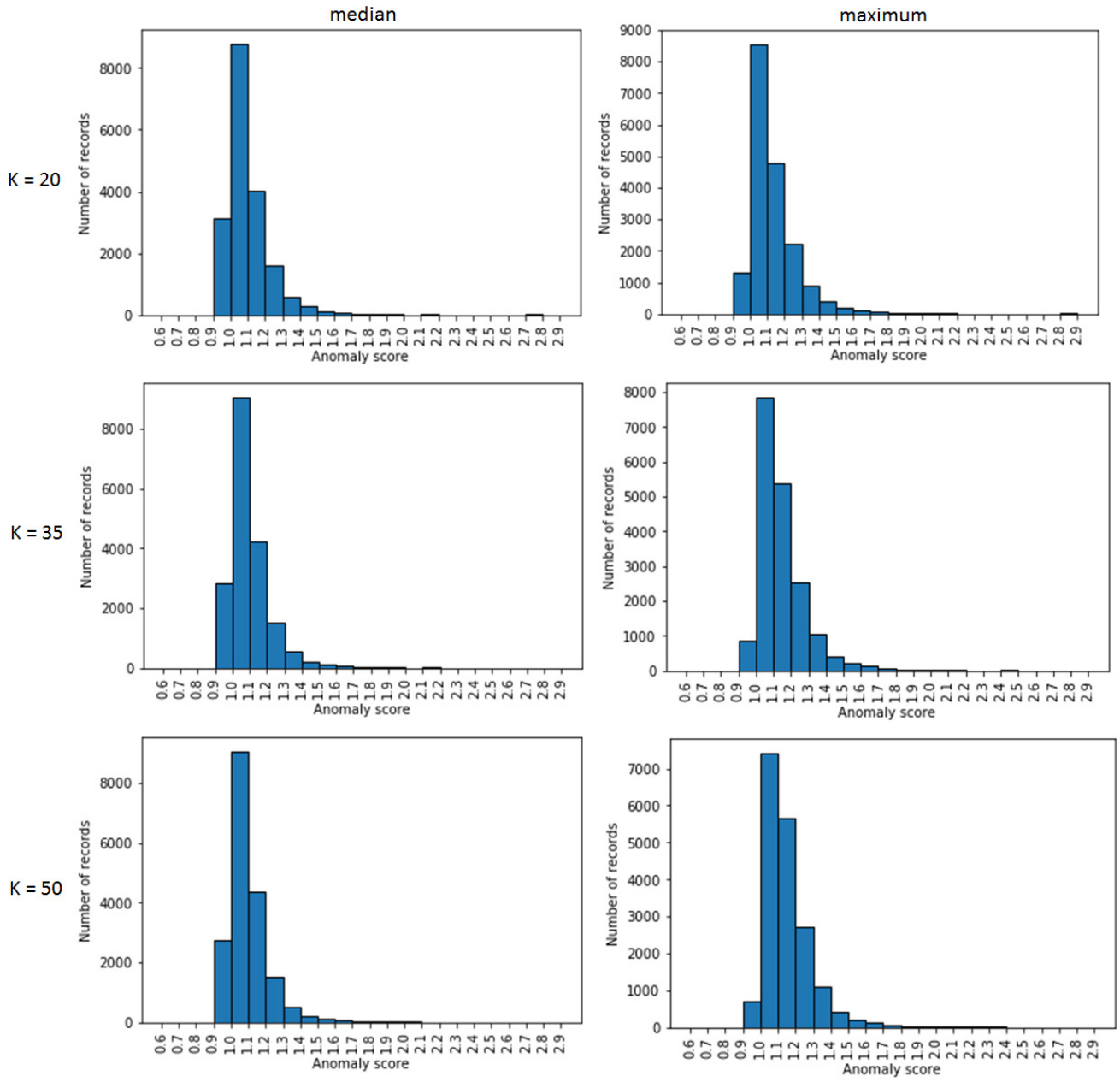

Figure 4.6: Distribution of anomaly scores for different parameters.

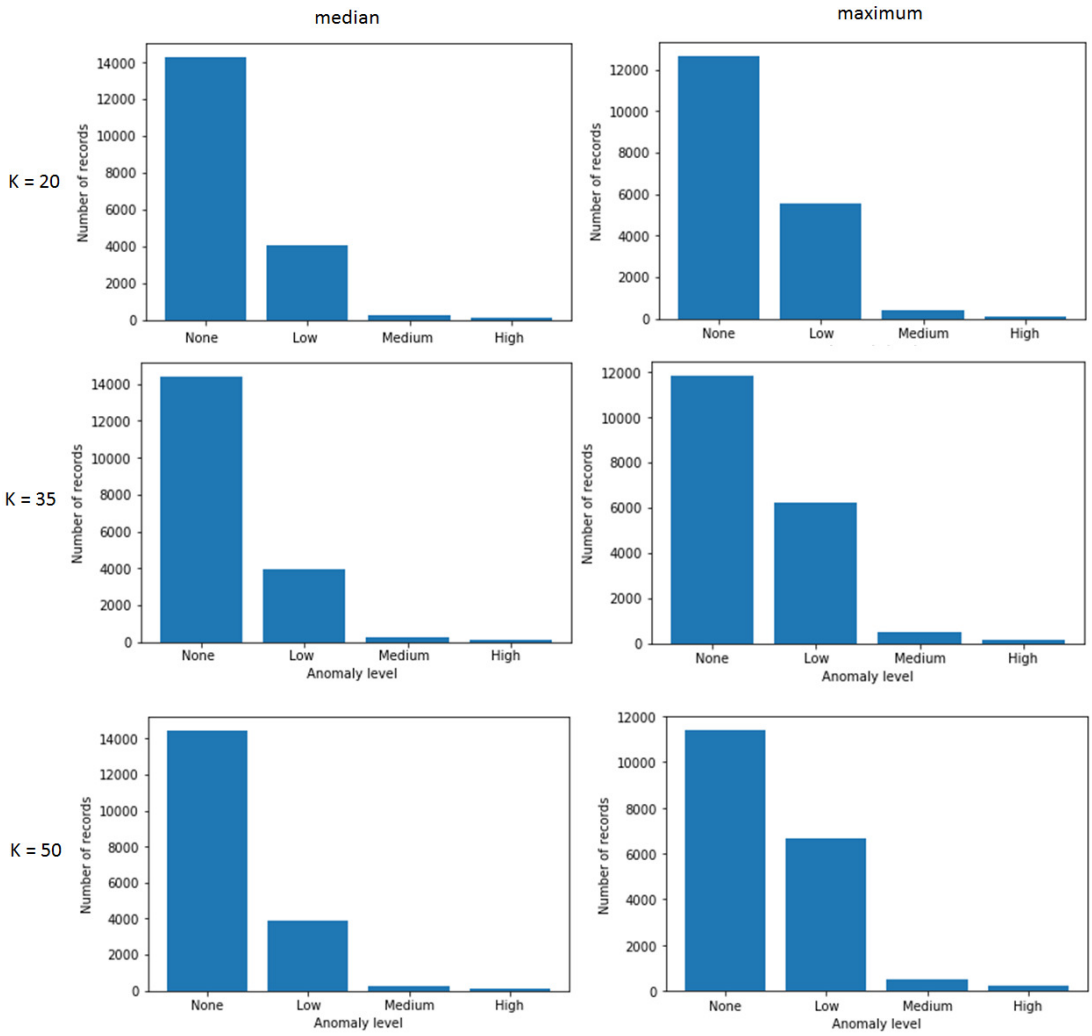

Figure 4.7: Distribution of the discretization of the anomaly scores.

Collection Timestamp	cpuIOWait	cpuKernel	cpuUser	cpuUtil	logicMemfreePct	score
2017-10-16 03:39:37	-0,29	0.43	-1.08	-0.81	2.06	0.98
2017-10-16 03:41:43	0.61	6.52	0.95	2.67	2.06	2.73
2017-10-16 03:41:43	0.61	6.52	0.95	2.67	2.06	2.71
2017-10-16 03:44:37	0.61	6.52	0.95	2.67	2.06	2.74
2017-10-16 03:49:29	0.61	6.52	0.95	2.67	2.06	2.74
2017-10-16 03:49:31	-0.25	0.38	-0.91	-0.65	2.06	1.42

Figure 4.8: An example of anomaly on normalized data.

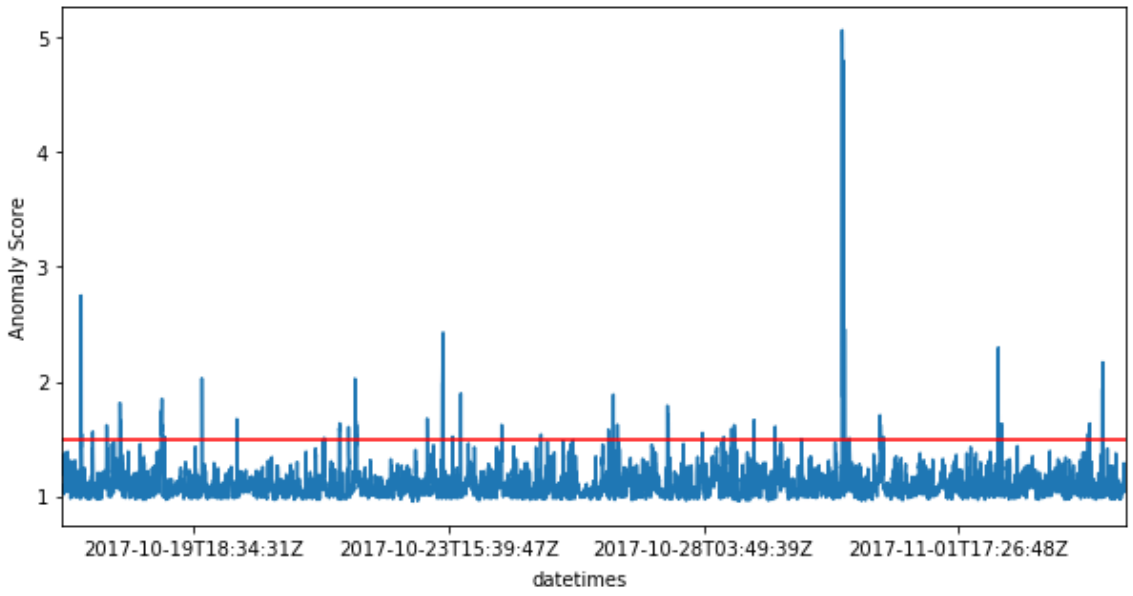

Figure 4.9: Anomaly trend of the analyzed machine with anomaly alert set to 1.5.

We see that the anomaly scored marked as a severe anomaly a problem generated by the cpu which time of response raised a lot along with other indicators. In the figure we showed the result scaled with the above mentioned technique to highlight the sudden changes. Note how the anomaly regarded only the CPU as the measure regarding other piece of the infrastructure like the percentage of free memory shown in the Figure were left unchanged.

This anomaly detection algorithm is propaedeutic to the realization of the final anomaly prediction algorithm but is itself a valid tool. If set up with a proper visualization tool in fact, it could offer a dashboard on which it would be possible to monitor the trend of the health of the machine by plotting the anomaly score on a temporal graph with possibly a threshold indicating when a sample of metric could be considered dangerous, like the one shown in Figure 4.9 which is set to 1.5, so that the reactive response to the anomaly can be quicker. Another advantage of this algorithm is that it could be possible to find the metric, or the metrics, that caused an anomaly by looking at the most divergent dimension with respect to the others among the neighbors.

To conclude, this anomaly detection step respects the secondary goal of being reusable for other machines since it works despite the number of metrics and their kind.

4.3 Feature extraction from applicative logs

The second part of our work was to find a way to deal with application logs coming from Splunk. The final goal of this stage is to find a way to extract a set of feature in a proper format to be ingested by the anomaly prediction algorithm. For this step we could not take a whole months of data as we did in anomaly detection but we had to reduce at only one week. This is because for this test instance of the Family banker portal Splunk outputs almost 1.5 million logs corresponding to a file of 1.5 GB. Normally, this would not have been a problem but to develop this algorithm we had to work on our laptops which were not powerful enough to deal with an entire month of data.

Our approach is divided into the following steps:

1. Logs exploration, to understand their nature.
2. Log cleansing and keyword extraction
3. Feature extraction, to transform the history log into a proper format for a machine learning algorithm, that in our case is be a set of vectors.

To perform step 3, we applied two techniques: Tf-idf statistics to give each keyword a proper weight and then K-means clustering, to group together similar logs. The output of this latter algorithm is be the input for the anomaly prediction algorithm.

Before describing how we performed the above mentioned steps, we give an explanation of the techniques used in Step 3.

4.3.1 Tf-idf (term frequency-inverse document frequency)

Tf-idf is a numerical statistic approach that intends to reflect how important a word is to a document in a corpus which is often used as a weighting factor in information retrieval, like in our case. The tf-idf value increases proportionally to the number of times a word appears in the document but inversely proportional to the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Before calculating tf-idf for every word in our logs we need to perform a preparation step that is projecting the logs into a *bag-of-word model* also known as *vector space model*. In a bag-of-words model a text, such as a sentence or a document (in our case, the applicative logs) is represented by a vector of its words by keeping multiplicity. In our case then, each applicative log will be represented as a vector of n dimensions, where n is the number of distinct words in all logs, in which every dimension represent a word and its value corresponds to the number of occurrences of that word in that particular log. After this step is concluded, our collection of keywords of logs will be transformed into a set of integer array of n dimensions thus being ready to calculate tf-idf statistic.
As the name suggests, this approach is the product of two statistics: term frequency and inverse document frequency. Term frequency leverage on the idea that the more a word appears in a single document, the more this document is related to the word. In order to calculate this statistic, the simplest way is to perform a raw count of the word in each document. In this case, the quantity is expressed as $tf(t, d)$ where t is the term we are analyzing. Anyway, using just the raw count could lead to a bias towards bigger documents therefore the document length is introduced a normalization term. If we denote $f_{(t,d)}$ as the raw count of a word in a document we can define the term frequency statistic as:

$$tf(t, d) = \frac{f_{t,d}}{\sum\limits_{t' \in d} f_{t',d}} \tag{4.5}$$

The problem with term frequency is that it tends to emphasize common words which appear frequently in all documents by nature, giving them excessive importance. Inverse document frequency acts as a regularization term for this phenomenon by acting as a measure of how much information the word provides, that is, whether the term is common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient, calculated as follows.

$$idf(t, D) = log \frac{N}{1 + \mid \{d \in D : t \in d\} \mid} \tag{4.6}$$

Where:

- N is the total number of the documents in the corpus, that is N=—D—
- $\{d \in D : t \in d\}$ | is the number of documents of the corpus in which t appears. If the term is not in the corpus this would lead to a division by zero which is commonly adjusted with the summation by 1

Now the statistics are combined by a simple product to obtain the final tf-idf formulation:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{4.7}$$

4.3.2 K-means clustering

K-means clustering is a clustering algorithm which aims to partition n-observation into k clusters. Each observation belongs to the cluster with the nearest mean which serves as a prototype of the cluster.
Given a set of observations $(x_1, x_2, \ldots, x_n)$, where each observation is a real vector of d dimensions, k-means clustering aims to partition n observations into $k(\leq n)$ sets $S = \{S_1, S_2, \ldots, S_n\}$ so as to minimize the within-cluster sum of squares (WSS). Formally the objective is to find:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \tag{4.8}$$

Where μ_i is the mean of the points in S_i
The most common approach is an iterative technique. Given an initial set of k-means $m_1^{(1)}, m_2^{(1)}, \ldots, m_n^{(1)}$ the algorithm proceed by alternating between two steps:

Assignment step: Assign each observation to the cluster whose mean has the least squared Euclidean distance, this is intuitively the "nearest" mean.

$$s_i^{(t)} = \{x_p : \|x_p - m_i^t\|^2 \leq \{x_p : \|x_p - m_j^t\|^2 \forall j, 1 \leq j \leq k\} \tag{4.9}$$

Where x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.
Update step: Calculate the new means centroids of the observations in the new clusters:

$$m_i^{t+1} = \frac{1}{\mid S_i^{(t)} \mid} \sum_{x_j \in S_i^{(t)}} x_j \tag{4.10}$$

The algorithm has converged when the assignments no longer change. There is no guarantee that the optimum is found using this algorithm.
The algorithm is often presented as assigning objects to the nearest cluster by distance. Using a different distance function other than (squared) Euclidean distance may stop the algorithm from converging whereas various modifications of k-means have been proposed to allow using other distance measures. The results may be influenced by the initialization methods, depending from the dataset: initial prototype can be set as random points or to be as distant one from another as possible, other methods are possible but further explanation would be no use for the goal of our thesis. Usually, to decide the value of k, one plots the value of WSS for each k and then identifies the "elbow" of the graph, such as the k for which the derivative of WSS starts decreasing becoming more and more flat. This is done because the minimum for the WSS function is found when $k = n$ and thus making the whole algorithm useless. This procedure has also the side effect of reducing the number of clusters so that data would be less fragmented.

4.3.3 Proposed Solution

We can now start to show how we performed feature extraction on the application logs. As we did for the sample application, we analyze the distribution of plain values per fields of the test application which is shown in Figure 4.10.
If we compare it to the distribution in Figure 3.6. we can notice two things. The first one is that the number of fields collected for this application is far greater than the previous one, this time each log is composed by 275 fields while before we had 161. The second thing to notice is that even in this case there is a predominance of fields that are almost always left blank, these two factors are a consequence of the high customizability of Splunk. Even if these two application may seem different in terms of fields collected, as anticipated in Chapter 3 they share at least one common field, "_raw",

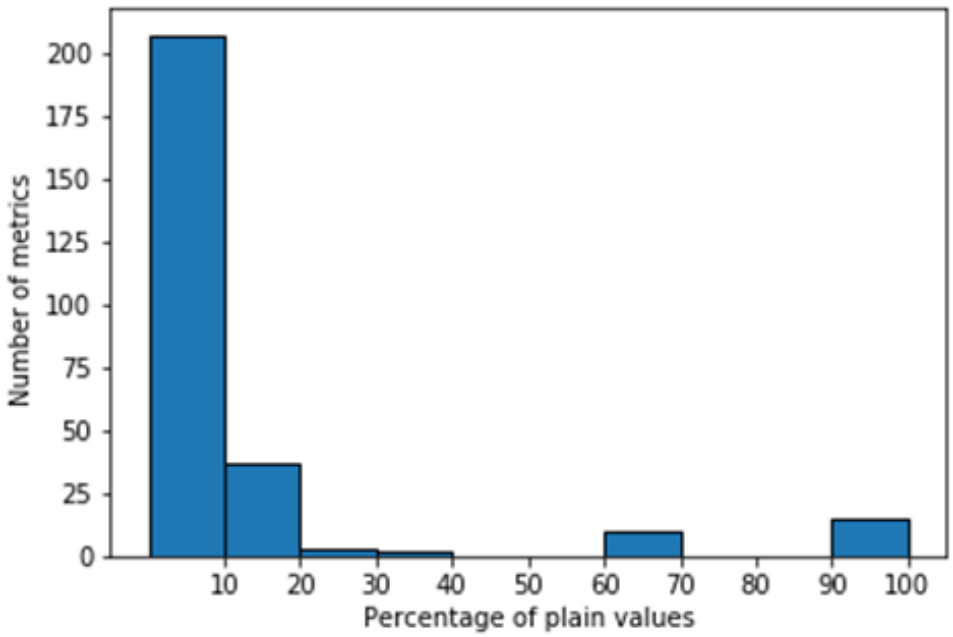

Figure 4.10: Distribution of plain values in the logs of the family banker portal.

which contains the original log and therefore all possible information. It is clear now that in order to extract any information from the application log, the only viable option is to find a way of parsing this field. This approach also perfectly fits to the underlying request of the customer: use the most general approach possible. Figure 4.11 shows an example of how the "_raw" field of the log is composed, using a random log as an example.
It is easy to understand that this particular log has the same format as the one in in Figure 3.7 while it is carrying information about a different application. This is possible thanks to the log4j standard and thus, finding a way to perform feature extraction on this entity would mean being able to perform in on every application following this standard. Differently from Anomaly detection in which we were able to find a reliable methodology which has been proved to be valid by the literature, in this case we had to create our own algorithm. In fact, it is hard to find some open-source log parsers which could fit our solution since some were created for a very specific situation while others had some prerequisites that were not satisfied in our situation like knowing beforehand the format of the logs. Our idea is to group these logs into a finite number of clusters in such way that we could tell how many logs of each group have been produced by the application in

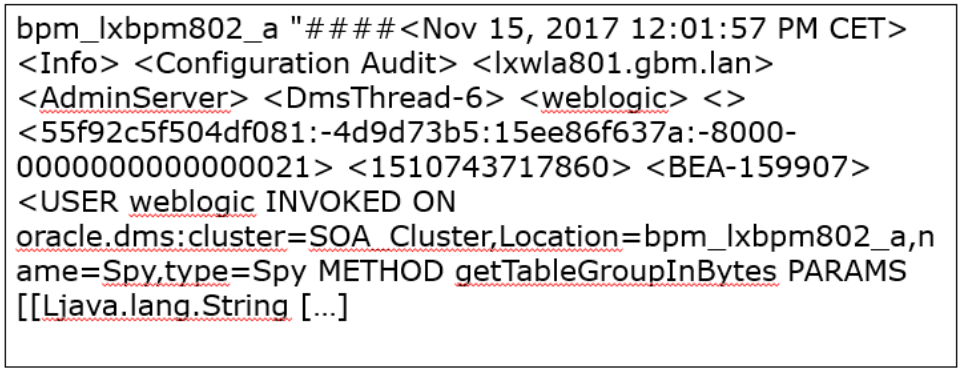

Figure 4.11: A Log from the family banker portal.

a given period of time. To do this we need to extract keywords from logs and then use a clustering algorithm that could group them based on these words, the entire process is explained ahead.

By exploring the logs, we realized that depending on the error message they carried, some of them were much longer than the others, often because this message reported the Java class which triggered the error which in a complex application could mean up to hundreds of recursive calls. To give an insight of the complexity and heterogeneity of the logs in Figure 4.12 we plot using a logarithmic scale on both axes the distribution of the number of words in each log, obtained by splitting them using as a separator every possible punctuation symbol. It is clear how logs could take any form, from very few words up to ten thousand, the maximum recorded is a log with 13305 words.
The first step is to clean the logs; this is done not just by splitting the words using any possible punctuation but we need to distinguish the parts carrying information with the one who do not.
Our log cleansing approach could be divided mainly in the following steps:

1. Eliminate every single variable part in the logs, these may include numbers in an alphanumeric word. As already mentioned in Chapter 2, logs are composed by a fixed part, set by developers, that indicates the type of event and a variable part, set by the application in real time.

2. Eliminate all queries: some logs were caused by some queries which generated an error. For this level of analysis, we can't go deep into analyzing the weight of each query but we content about knowing

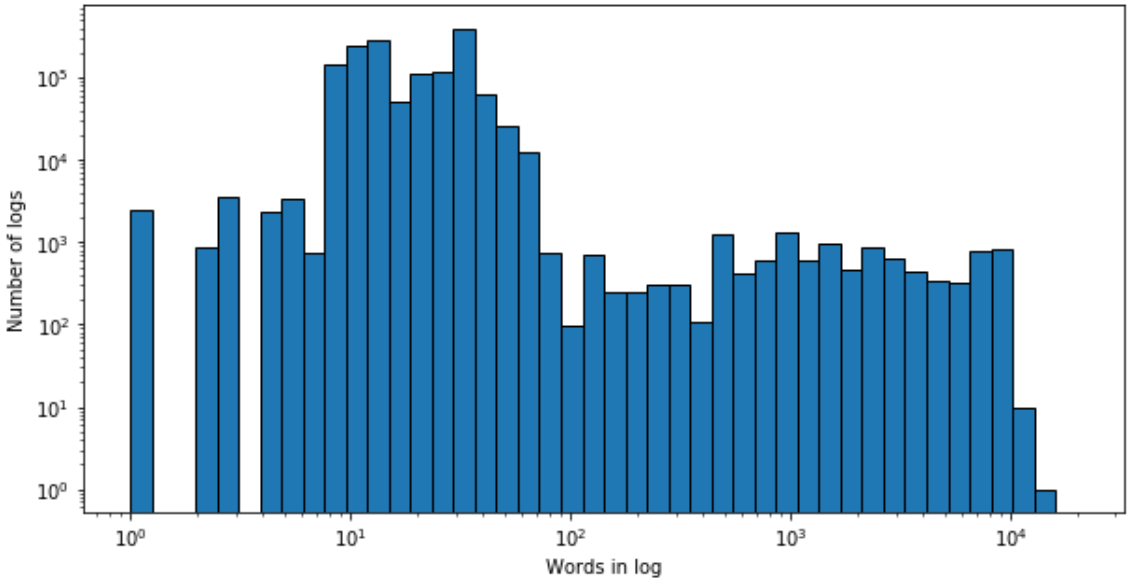

Figure 4.12: Distribution of words in the logs.

which error a query generated.

3. Eliminate extremely verbose descriptions. As mentioned before Some logs reported the class generating the error with the consequences previously described and shown. We decided just to keep the Java error reported by dropping other useless information.

4. Eliminate stopwords.

An example of the result of this process can be seen in 4.13.
It is evident how for every log only keywords are kept while unintelligible parts are discarded. Another achievement of this step having reduced the number of total log: from 1484090 logs as initial data to 670156 after the cleansing process as shown in Figure 4.14. This is due to the fact that many logs contained just punctuation or unintelligible alphanumeric sequences like "\uxa782" which do not bring any information. Our hypothesis has been confirmed by expert technicians which work on the machine every day, in fact they revealed that those kind of logs are remnants of an old logging system used before which explains why there are so many of them.
Now that we have a list of keywords for each log, to cluster them we need to find the most relevant ones. First, we explore the distribution of their frequencies as shown in Figure 4.15. We can see that while some words appear more than 130000 times, a large quantity of them are appearing very

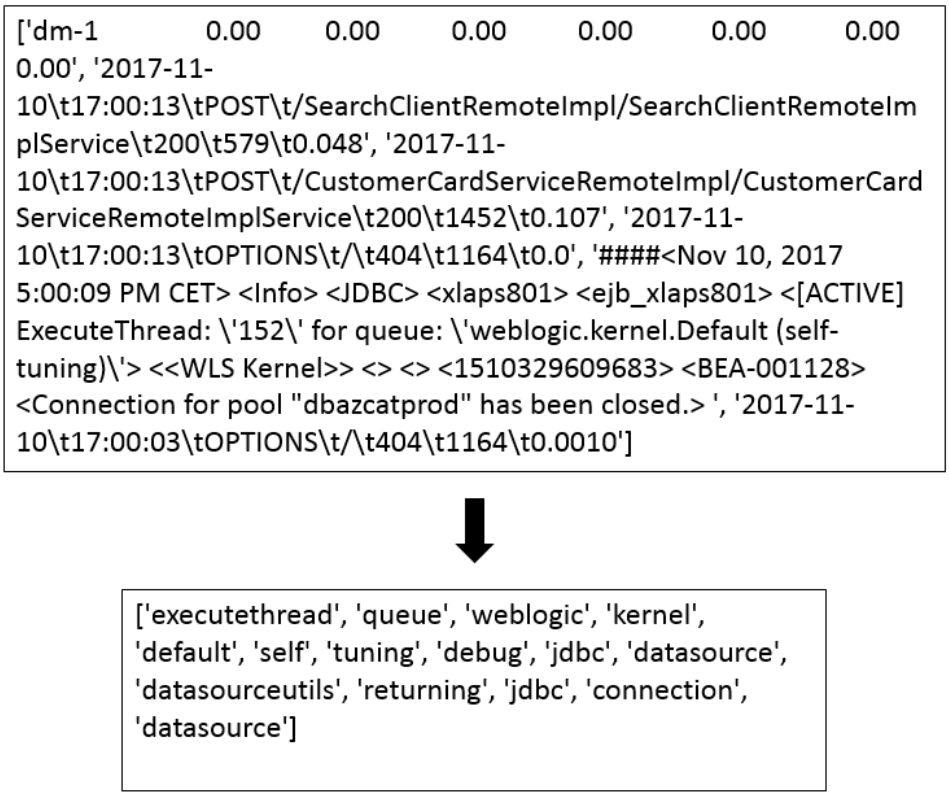

Figure 4.13: Cleansing process on a log.

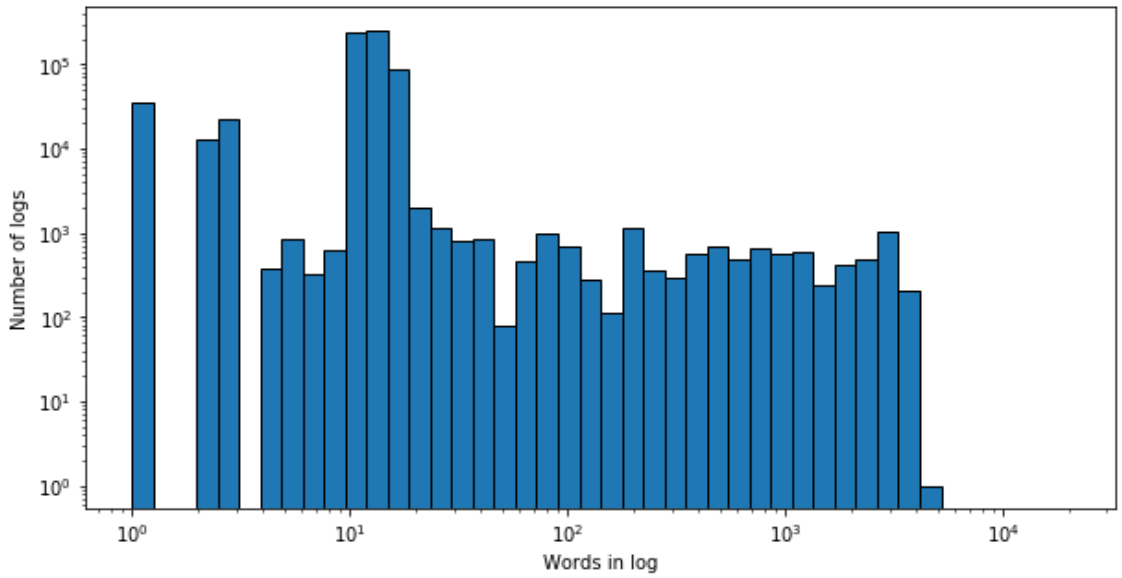

Figure 4.14: Distribution of words in the logs after cleansing process.

rarely, some lower than 10 times. For this reason, we made the following assumption: uncommon words are more meaningful than the most common ones for identifying a cluster of log.

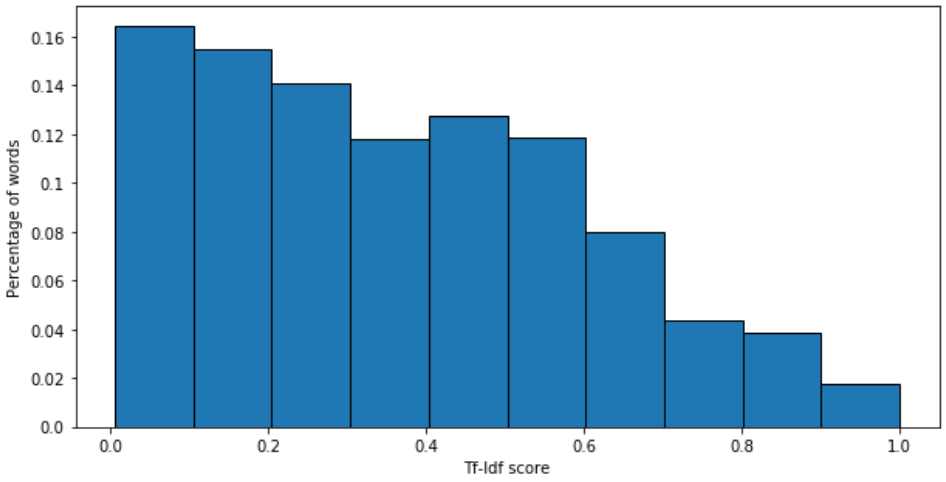

Figure 4.16: Distribution of tf-idf model scores.

To bring this concept in our algorithm we used calculated tf-idf function for

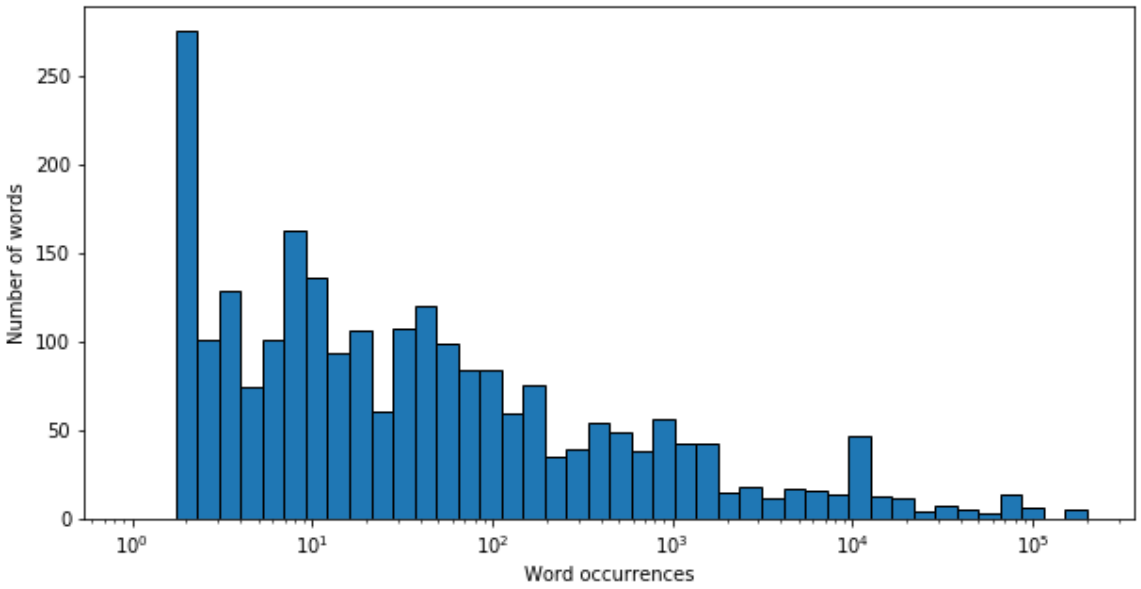

Figure 4.15: Distribution of occurrences of keywords.

every word in log history which is composed by 2443 distinct words spread in 670156 logs for a total of almost 17 million total words. The goal of this step is having a dataset composed by a n-dimensional vector, where n is the number of distinct word, for every log in the dataset so that we could use it as an input for k-means clustering algorithm. Since k-means is computationally expensive and we had to use only our laptops, we could not continue the algorithm using all dataset, therefore we split the entire corpus using just 10% of the logs obtained by random sampling. On this slice we calculated tf-idf statistics, this mean that it is very unlikely that all words are considered in this model It would be therefore misleading to show the most relevant words according to the calculated model, for this reason in Figure 4.16 we show just the distribution of the scores per word with respect to the percentage to help the reader.
Having now the tf-idf statistics, we can calculate k-means clustering. Figure 4.17 shows the value of WSS we obtained for the following values of k:

a We first run the algorithm using an evenly spaced array of k, from 20 to 150 with step 10, in order to have a generic idea of the trend of WSS function. We considered the elbow to be around k=80.

b We run the algorithm again for every k between 70 and 90 in order to focus the analysis around the elbow. Unfortunately, the trend becomes

linear in this interval meaning that there is no proper elbow.

The consequence of Figure 4.17 (b) is that we have no proper indication on the value of k to set. We decided to set k=80 since it looks like a proper elbow in Figure 4.17 (a) and choosing any larger k would bring few additional information while forcing us to deal with a larger value of dimensions in input to the final Anomaly Prediction model.

Remembering that we used only 10% of logs for performance issues, we need to include the remaining 90%. To do that we simply use the newly created model to "predict" the unused records. To test if the clustering model was correct we analyzed the distribution of logs per cluster for both sample logs and the history logs, here shown in Figure 4.18. From the distribution we see that the proportion between clusters are equal both in the sample and the history, therefore we can safely assume that we would have obtained a similar result if we run k-means algorithm using all dataset.
By looking at the distribution, it appears that there are two clusters, there named as C_1 and C_2 which alone contains roughly 35% of the total logs. For the sake of completeness, we show an example of log belonging to those clusters and compare them. C1 contains logs that, once cleaned, appear like this:

- *[executethread, queue, weblogic, kernel, default, self, tuning, info, getcolumnlabel]*

These two clusters may seem very similar but they differ from the last word. As long as these words could appear irrelevant, they actually have both a high tf-idf value, around 0.96 each, which casts them very far one from the other in the vector space. If we explore C_3 instead we see that is starts diverging more to the previous ones, in fact a typical log is composed as follows:

- *[executethread, queue, weblogic, kernel, default, self, tuning, info, gri, start, getindex, service, method]*

To complete the exploration on the composition of the clusters we show two logs of C_{79}:

- *[executethread, queue, weblogic, kernel, default, self, tuning, info, business, prodottocartabo, chiamata, prodottocartabo, loadfeatures, con, input]*

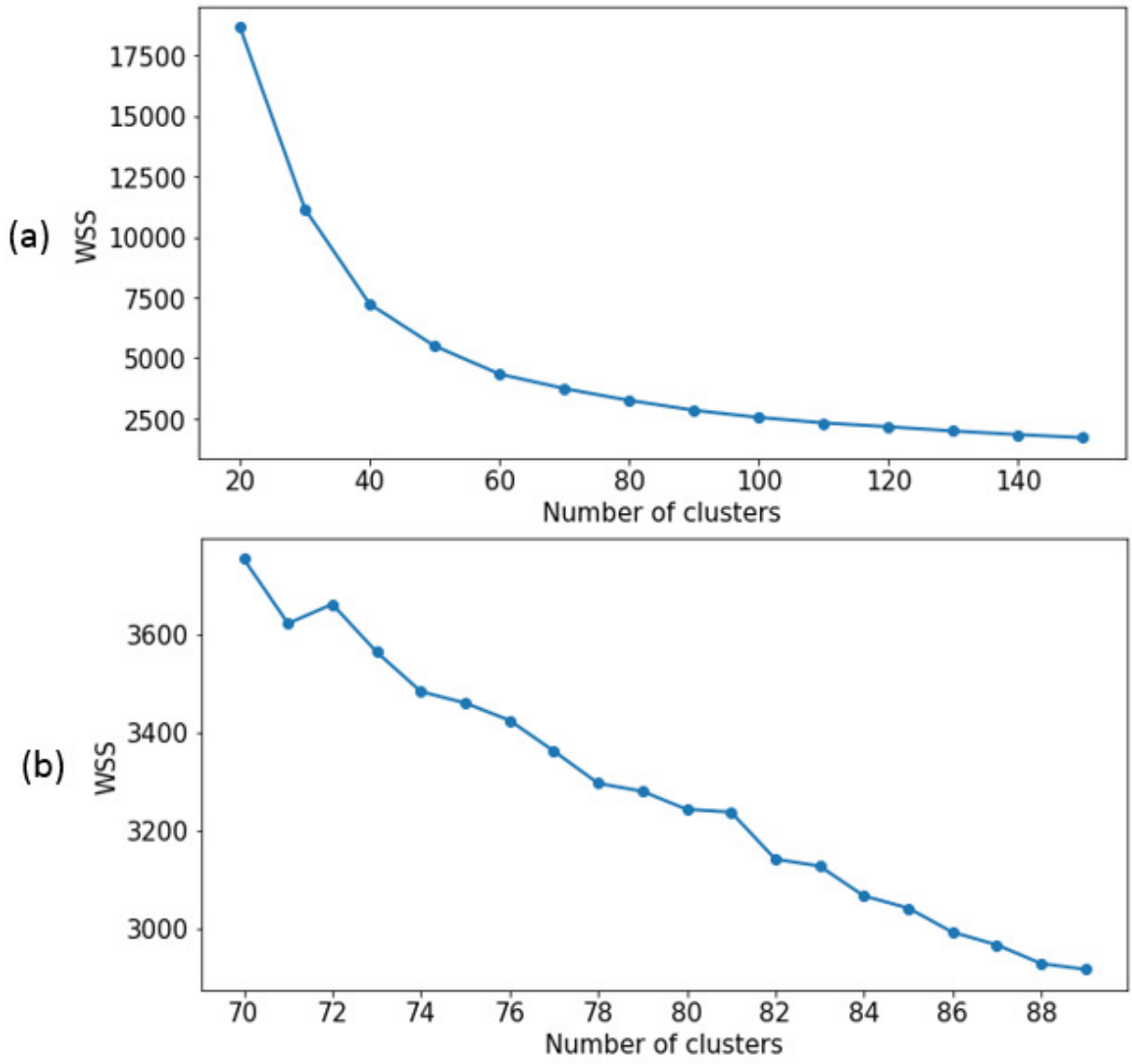

Figure 4.17: Within cluster sum of squares for different k.

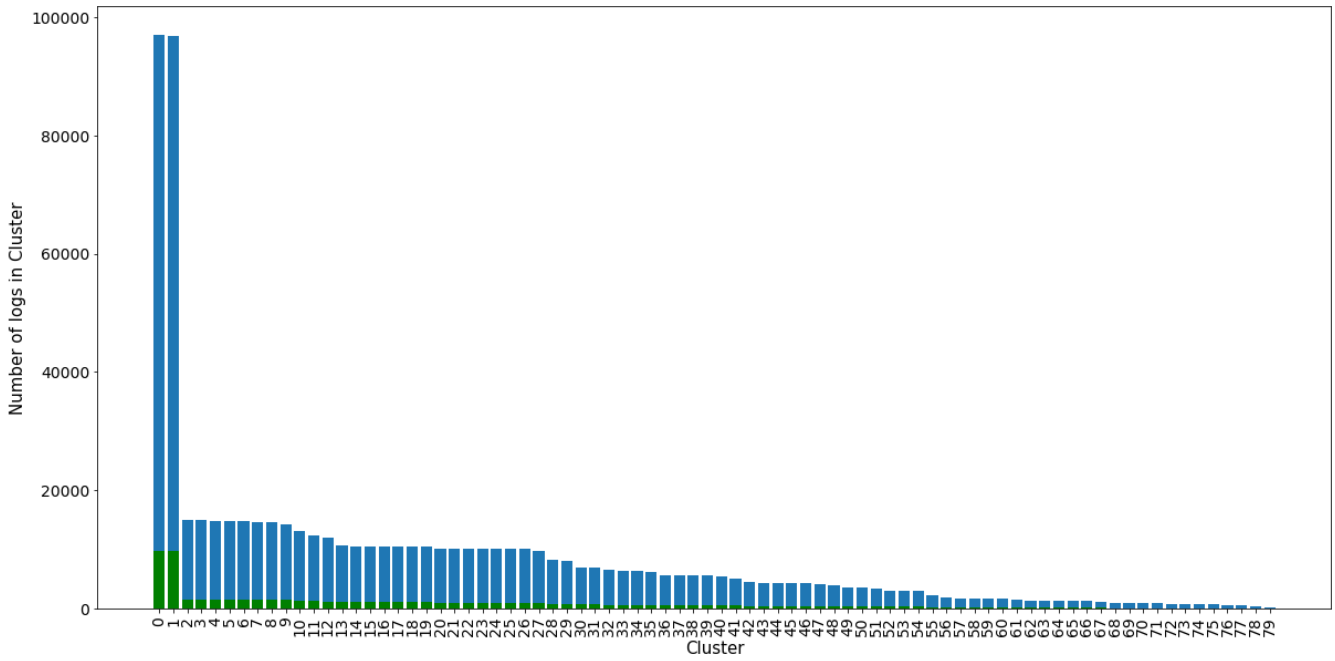

Figure 4.18: Distribution of logs into clusters for both sample and history logs.

- *[executethread, queue, weblogic, kernel, default, self, tuning, info, eseguita, query, per, popolare, una, lista, entityrecuperata, bmed, dao, entity, dettbenef, con, executethread, queue, weblogic, kernel, default, self, tuning, info, business, prodottocartabo, chiamata, prodottocartabo, loadfeatures, con, input]*

It is clear now that there are words, the initial ones, which are common in almost all logs and therefore influences less the calculation of the distances between them. Moreover, by exploring clusters with lower cardinality it is easier to find heterogeneous logs since this is where are grouped logs that are not so distant one from another because they carry more unimportant words. As last observation, we noted that there is no cluster in which appear two different log4j severity level, mentioned in Table 3.2, meaning that they were of great help in clustering logs. The second step of the anomaly prediction algorithm is now concluded, it took in input a raw log and it returns as output a number, representing its cluster, along with its timestamp. The entire flow is resumed in Figure 4.19.

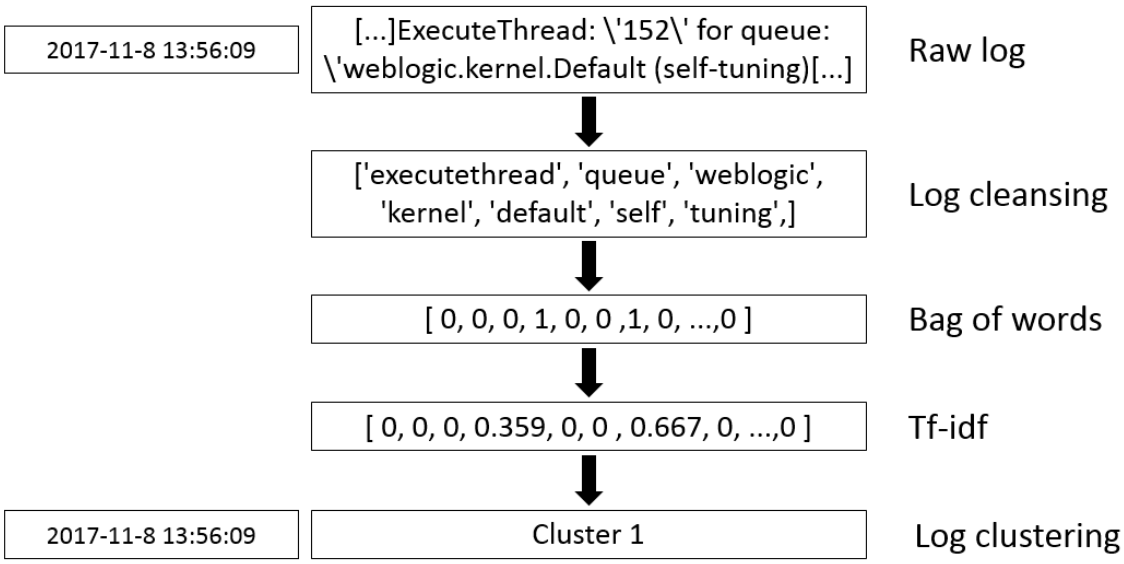

Figure 4.19: Log clustering process schema step-by-step.

4.4 Anomaly prediction

This is the final step of our algorithm. The goal is to use applicative logs to predict the next infrastructural anomaly. The previous two steps can be considered as a preparation step for this goal: with anomaly detection we built a dataset of target variable using anomaly score while with log parsing we transformed raw logs into real vectors so that we can use them as input for the model.

The main problem in combining those two steps together is that they are independent one from another such as they sampled data asynchronously thus meaning that we had to resort to some kind of aggregate measure. Before seeing how we managed to do this we must recall that data coming from Oracle Enterprise Manager had a time span of 30 days while the ones coming from Splunk over 7 days. To build a dataset we could therefore use only the span dictated by Splunk thus discarding 23 days of infrastructural metrics which, anyway, were useful to build a more consistent anomaly scoring. This reduction caused the metrics log dataset to fall from 18765 entries to 3824. Figure 4.20 shows the different distribution of infrastructural and applicative logs in this period.

The lower amount of sample in November 10 is due to the fact that we started gathering data from the afternoon of that day. We can see that

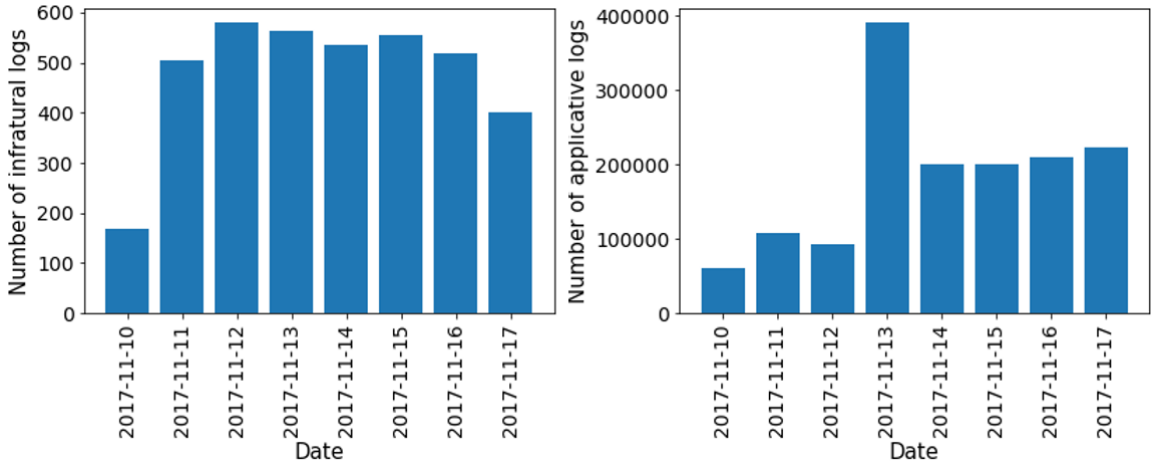

Figure 4.20: Distribution of logs during analyzed week

if the number of collection for infrastructural metrics remains more or less the same while for applicative logs it is less homogeneous. This is due to the fact that infrastructural metrics are sampled at a constant rate while applicative logs are produced in response to an event, in fact we can see that for November 11 and November 12 we have lot less collection with respect to the other days because they were respectively Saturday and Sunday and thus they corresponded to a period in which the portal was poorly used.
To aggregate such different data, we used a multiple sliding windows technique which composed by two hyperparameter: the first one is Δt which is the width of our sliding window and the second one n, representing the number of sliding window we consider at the same time. This second parameter could sound tricky but it allows us to consider a wider time span composed by smaller ones, the reason for this choice will be soon presented. To better understand this we start by showing how we aggregated both kind of logs. Applicative logs are the simpler ones, in fact it is enough to tell how many logs of a certain cluster were produced during Δt. This is done by summing all occurrences of different clusters thus populating a count event matrix, this is shown in Figure 4.21.
Infrastructural logs require a more sophisticated procedure since anomaly score is a continuous attribute. First, we calculate for each time window of width Δt the mean of the anomaly scores that resides into it and then we calculate the mean of the means of n consequent sliding windows. Formally: given a set of m anomaly scores $(a_1^k, a_2^k, \ldots, a_m^k)$ produced during time window t_k, the mean anomaly score of this window is calculated as:

Collection Timestamp	Cluster
2017-11-08 13:56:09	1
2017-11-08 13:56:31	3
2017-11-08 13:56:44	3
2017-11-08 13:57:03	2
2017-11-08 13:57:59	2
2017-11-08 13:58:01	4

Sliding Window	Cluster 1	Cluster 2	Cluster 3
2017-11-08 13:56:00	1	0	2
2017-11-08 13:57:00	0	2	0

. . .

Figure 4.21: Applicative log aggregation example with Δt of 1 minute

$$\mu_k = \sum i = 1^m \frac{a_i^k}{m} \tag{4.11}$$

Then, given g_k the group $(t_k, t_{k+1}, \ldots, t_{k+n})$, of n consecutive time windows to consider, we calculate the mean anomaly score of g_k as:

$$\Theta_k = \sum_{i=k}^{k+n} \frac{\mu_i}{n} \tag{4.12}$$

Figure 4.22 explains graphically these quantities.

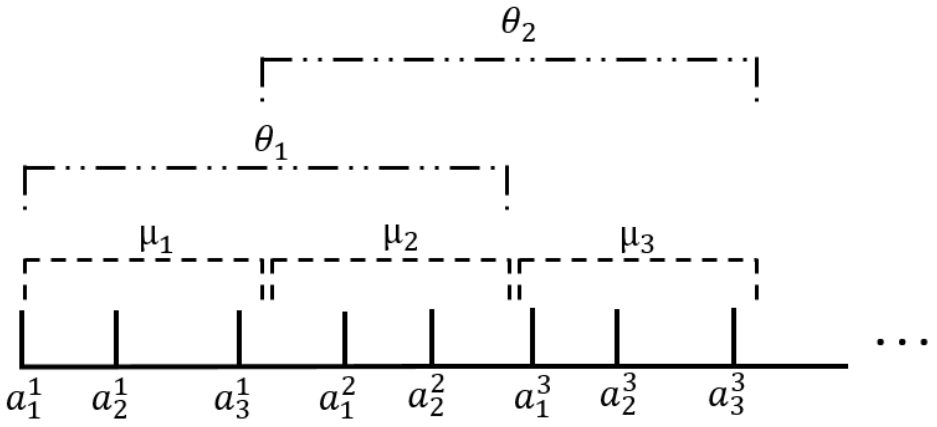

Figure 4.22: Example of multiple sliding windows technique using $n = 2$

Note that calculating Θ_k as a mean of means is different from calculating as the mean of all records of n consecutive time windows, in fact Θ_k gives equal weight to the anomalies reported during a window regardless of the amount of records it contained.
A point can therefore belong to one and only one time window but belongs to n sliding windows. This new quantity, Θ_k , will become the target variable of the anomaly prediction algorithm. Our choice is led by the fact that Θ_k allows to consider a wider time span at the same time, thus using more information for each prediction, while allowing us to still predict the anomaly score for the next Δt. Given k the next time window for which we would like to know the mean of anomaly scores, Θ_{k-n+1} would be the quantity

we predict. By knowing it, it is possible to calculate μ_k from the newly predicted in the following way:

$$\mu_k = n\Theta_{k-n+1} - \sum_{i=k-n}^{k-1} \mu_i \tag{4.13}$$

To bind applicative log collections with Θ_k we have to make the following assumption: application logs do not immediately influence anomaly score but their effect becomes visible in the near future. This assumption has been discussed along with the experts currently monitoring the system and has been considered valid because single errors in the application are often managed by internal procedures like the load balancers and cause issues only when they occur in groups. Moreover, since the anomaly prediction will be running continuously ingesting data in streaming, we cannot predict Θ_k using applicative logs that incurred during this time period because it hasn not finished yet: we have to wait until Θ_k is finished to group the applicative logs and thus it would be like predicting an anomaly score from the past, thus making the algorithm useless. This sliding window mechanism would also bring in the model the concept of time without recurring to computationally expensive neural networks, in fact by predicting Θ_{k+1} we use also some of the applicative logs that appeared in Θ_k thus making our model learn consequentiality.
The goal of the algorithm is then to estimate μ_k, the mean of anomaly scores in the next time window, by predicting Θ_k using the aggregation of the applicative logs incurred during $(t_{k-n}, t_{k-n+1}, \ldots, t_{k-1})$. The models used and the results are discussed in the next chapter.

Chapter 5

Results and considerations

In this chapter we are going to present the models we used and the results we obtained. We anticipate that the kind of data we had, in terms of quantity and quality, made basically impossible to achieve an acceptable result, that is why we preferred to focus on all the motivation that led to this poor conclusion rather than obsessively trying to find a way to fit a model.
Before showing the results, we remember that independently from the machine learning model one wants to use, our anomaly prediction approach has two hyperparameters: the width in minutes of the sliding window, Δt, and the number of sliding window to consider to calculate which we indicate as n. For this reason, we will plot the results obtained while changing these 2 parameters.
For this task we decided to try three different models: Random Forest regression, Support Vector Machine regression and Neural Networks. For the first two model we show, for each hyperparameter configuration, the one giving the best results according to a grid search parameter tuning method while for neural networks we use a fully connected neural network made by 2 layers of 20 neurons each. The choice of these model was led by the fact that we are not looking for the perfect algorithm but we rather prefer to identify which model could fit data best in order to give additional insight on its behavior.
For each model we performed the following preprocessing steps: the dataset is first split in training set and test with a 80/20 proportion using the last 20% of the entries as test set. This kind of split is done for two reasons: the first is that we have very few data so we have to use as much instances as we can to train the model, the second reason is that we want to test the capacity of the model to predict future anomalies based on the past ones and thus random sampling the occurrences would be less meaningful.

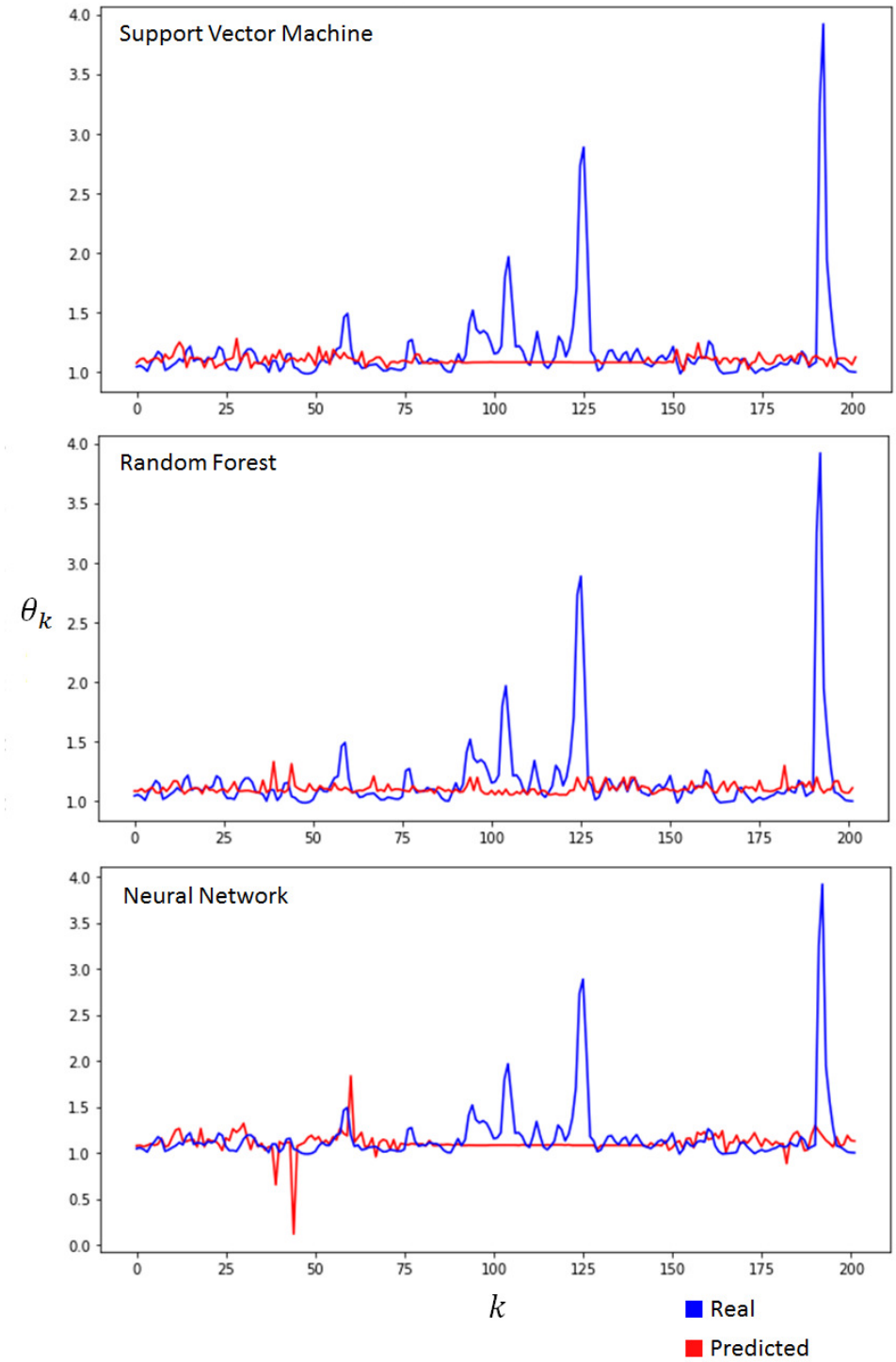

Figure 5.1: Results for $n = 2$ and $\Delta t = 10$ on the test set.

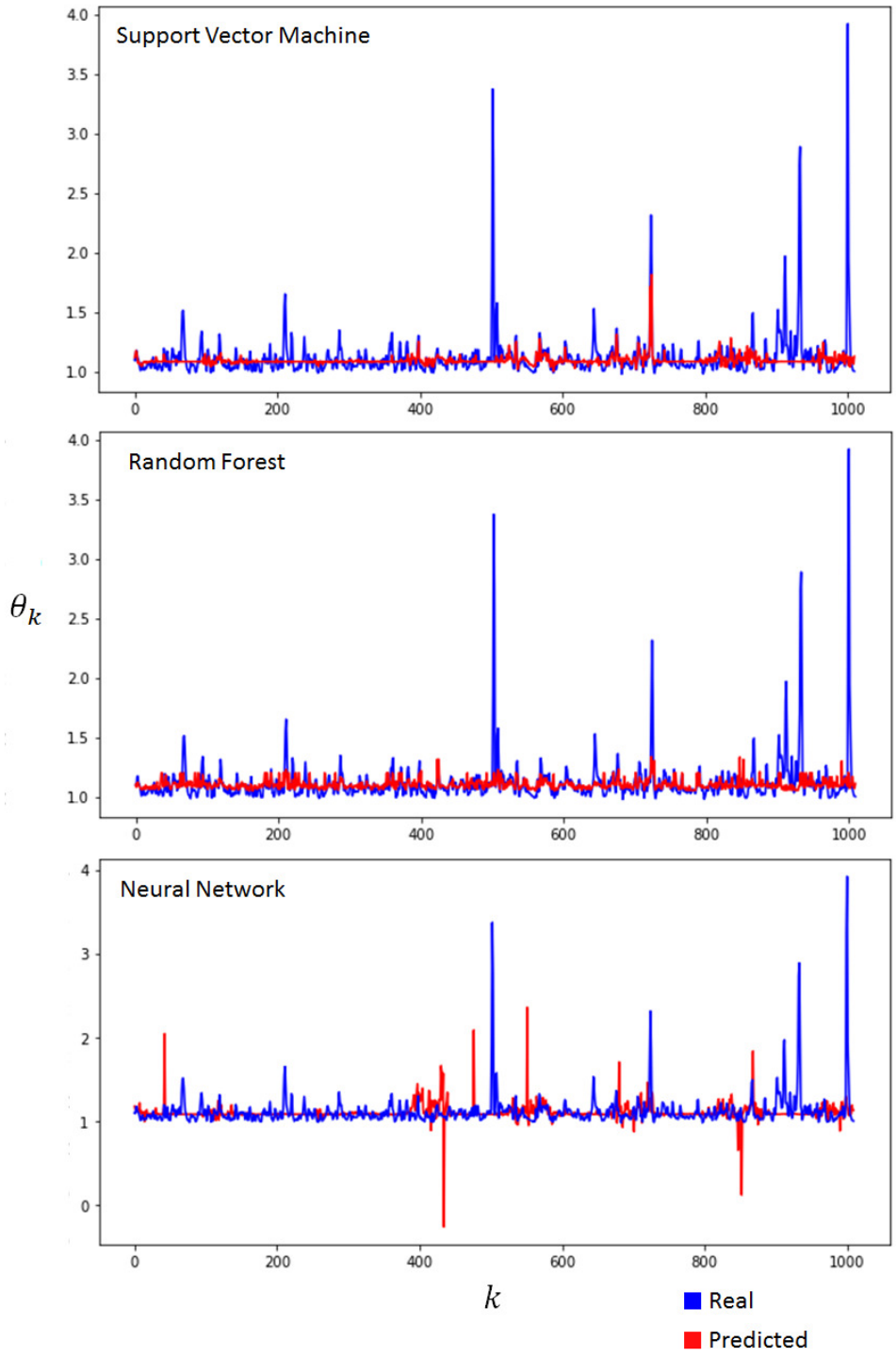

Figure 5.2: Results for $n = 2$ and $\Delta t = 10$ for all dataset.

After having done that, we performed Principal Component Analysis on the whole dataset by fitting it in such a way that it could explain 99% of the variance on the training set. This resulted in a dimensionality reduction of the dataset from 80 dimension to less than 30, depending on the current choice of the hyperparametrs. We can now proceed by showing the results. We started by taking $n = 2$ and $\Delta t = 10$ for which the above mentioned PCA returned 24 dimensions.

Figure 5.1 shows for each model the difference between its predictions and the actual test set. On the x-axis, we use the index of the test samples which are taken in temporal order while on the y-axis we have the corresponding value of Θ_k.
It can be easily seen that each model fails in correctly predicting the anomaly score, our guesses about the reason of this behavior are explained later on. Before diving deep into our speculation about this topic, we first check that the problem does not resides into the models themselves. To do this we used the models to predict the very same instances they have been trained on, the results of this test are shown in Figure 5.2. We can see that for SVM and Random Forest the model tried to fit data, without actually managing to fit it. Any attempt of fitting by modifying the parameters, even by trying to overfit, has resulted in failure. For the Neural Network instead the model did not even manage to fit the training set by even showing peaks where there was none. Any attempt done by changing the network layout, batch size, epoch or any other parameter did not end in better results.

Our major hypothesis is that the cause of this behavior resides in data, which in fact have a lot of characteristic that can harm the final result. First of all, we are in possession of few data, we recall that, due to the fact that we lacked of computational power, we could use just a single week of application logs with which we had to build our dataset, ending up with few thousands of data points. Another important aspect of data is that they come from a single machine of a test environment, this has two important consequences. The first is that we are considering just one machine of a cluster composed by two, thus we are forced to completely ignore the fact that this cluster acts as a single entity thanks to its protocol of load balancing and other features. There is no guarantee that this factor actually harms the quality of data but due to the nature of our goal this hypothesis is strongly suggested. The second consequence is that test environments are used, by a matter of fact, for testing purposes meaning that they do not experience the same kind of load as the production one. This includes

also using them to check how the application could react to certain circumstances by simulating, for example, infrastructural faults or overflowing it with requests and thus overloading it in a very brief, artificial time. This means that there is a high possibility that the nature of our data was too artificial to be actually predicted by a machine learning model.

Other motivations for such poor results, not excluding the precedent given one, could also reside in our approach. The final dataset in fact comes from the output different steps which altered original data, this includes for example the number of cluster we used for application logs and more specifically the very definition of Θ_k. We recall that it comes from a function accepting two hyperparameters: n and Δt that has to be chosen manually. To check whether this was the reason of the failure of the machine learning models we tried different combinations of them. An example is given in Figure 5.3 and Figure 5.4 which show the results we obtained by choosing $n = 20$ and $\Delta t = 1$: a diametrically different approach with respect to the previous one. Please note that the choice of using again a total of 20 minutes for the second plot is purely casual. It can be easily seen that the behavior is almost identical to the preceding one: on the test set none of the model manages to correctly predict Θ_k and using the same model for predicting the very same sample on which they have been trained on suggests that Support Vector Machine and Random Forest are trying to fit data without success while Neural Network manages to mispredict even them.
Different combinations of n and Δt were tried without giving different results in terms of quality. It has to be noted that by decreasing Δt we increase the number of total records in the dataset, normally this should give a boost to our performances but as we just seen this did not happened. The main reasons for this could still reside in the nature of data discussed above or, worse, in our approach which is based on various assumption, listed during this thesis, thus there is the possibility that one or more of the steps composing this algorithm actually harms data.
To analyze this possibility, we now recap all steps by listing what factors could have caused this behavior.

The first thing to consider has been already discussed at the beginning of Chapter 3: we are taking in consideration infrastructural anomalies considering just one cause for them such as applicative faults. There are other odds to consider: infrastructural anomalies could be caused by other minor infrastructural faults incurring for a prolonged period or even some external factor for which we are blind.

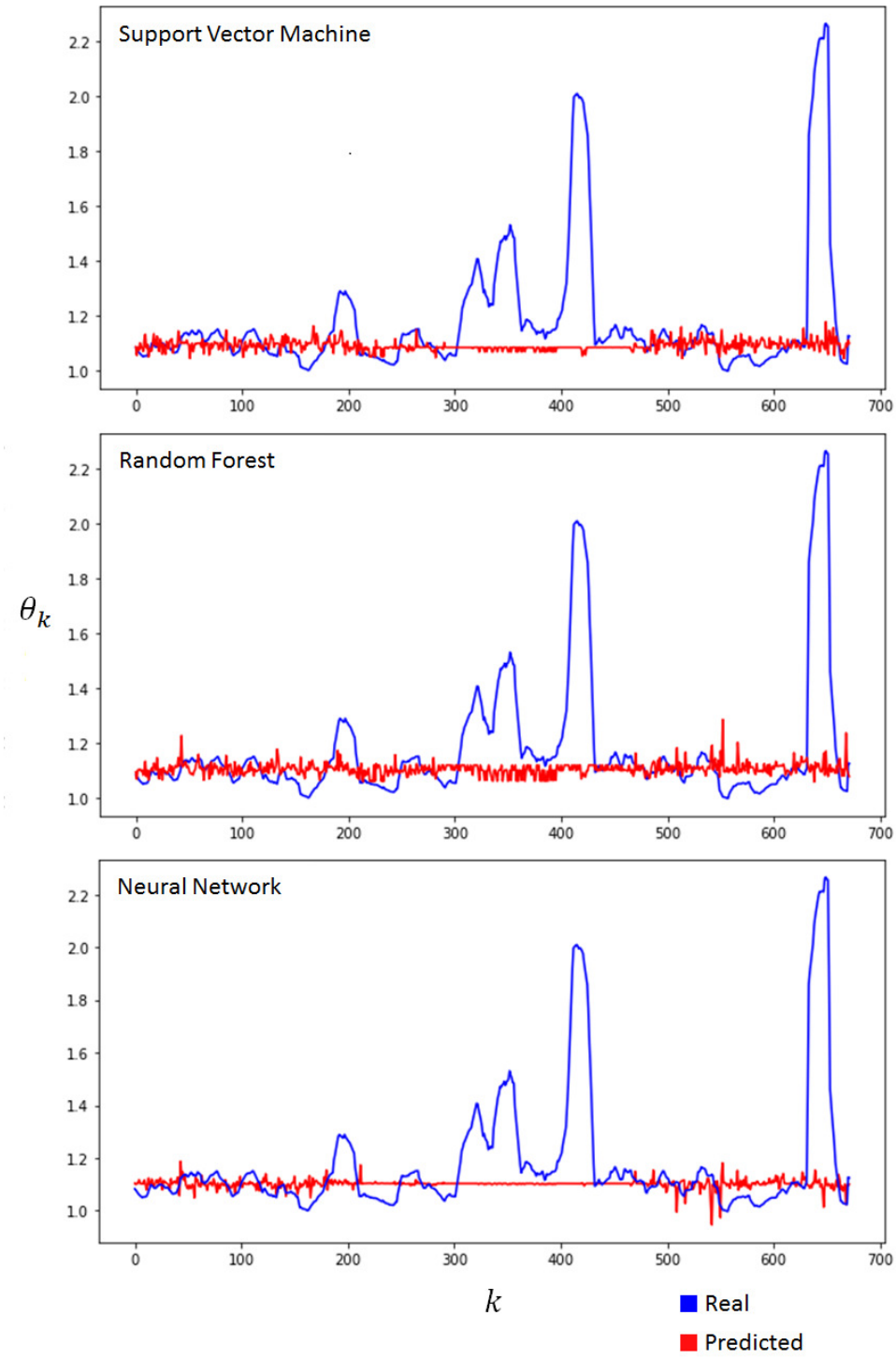

Figure 5.3: Results for $n = 20$ and $\Delta t = 1$ on the test set.

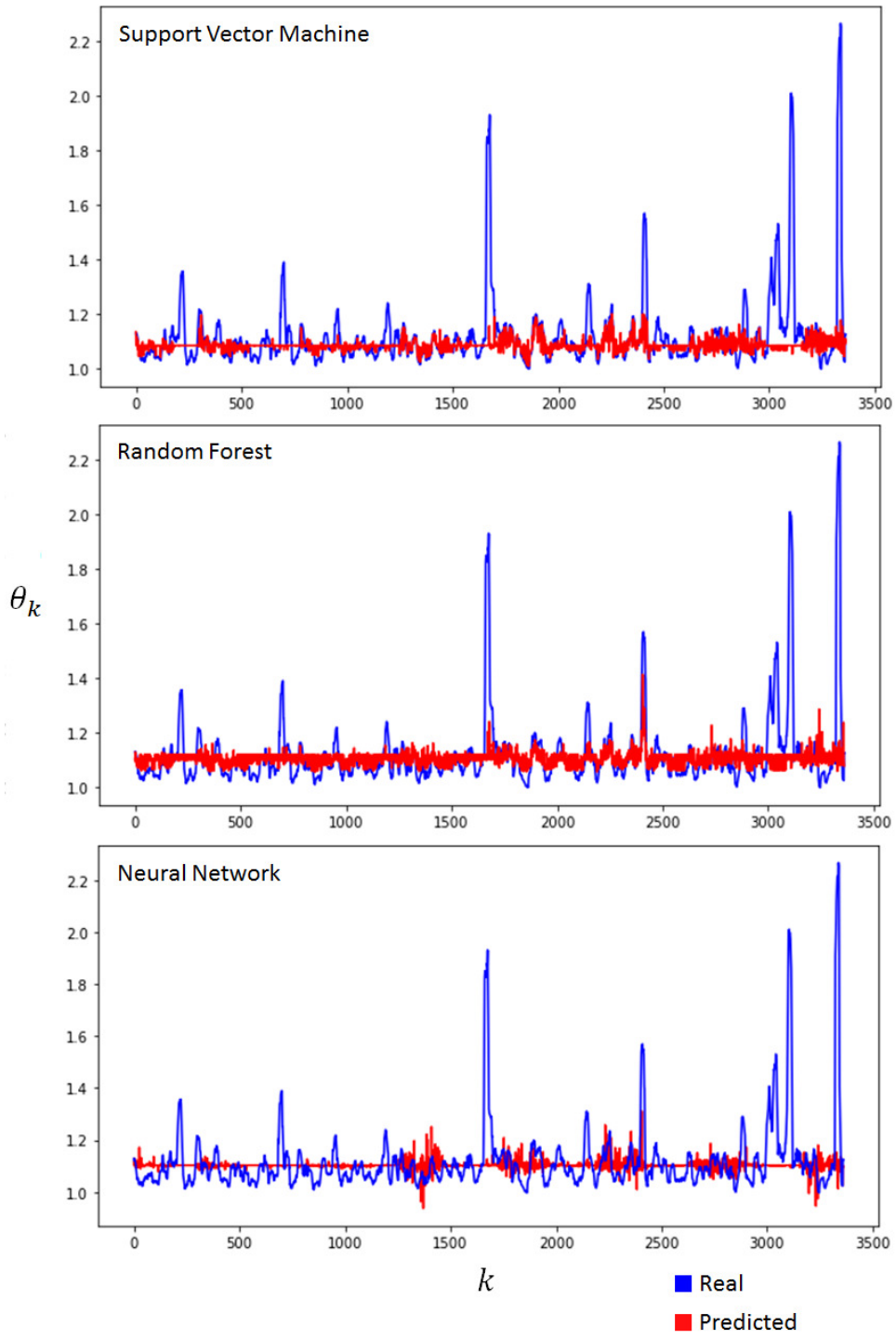

Figure 5.4: Results for $n = 20$ and $\Delta t = 1$ for all dataset.

With the limited resources and information, we had at disposal we acted under the hypothesis that the main cause of infrastructural anomalies was to be searched into applicative logs but, looking at the results, there is no actual guarantee in this assumption. To build an anomaly prediction algorithm one should examine deeply the causes of such anomalies in order to take every of them in consideration during the prediction phase.

The first step of the algorithm the anomaly detection which actually seems to be performing quite well. One thing to be noticed about this algorithm is that it tends to identify as anomaly any situation deviating from one considered normal. This could include also situations in which a deviation is not actually a bad behavior: for example, if the cpu-usage percentage always orbits around 60% but in a certain situation drops to 20% we could not be sure whether this comes from a malicious cause, for example a sudden failure in reaching data to elaborate leading to an idle CPU, or simply comes from a moment in which the application is not used. Our hypothesis is that any normal behavior has already been recorded with enough sample to be able to form a cluster big enough to be recognized by Local Outlier Factor algorithm for different k. To fine-tune this algorithm one should dive deep into the nature of every metrics understanding for every of them when a behavior must be considered malicious or not thus modifying the algorithm by consequence.

The second step of the algorithm is log parsing. Here we acted on data on two major ways: by cleaning the logs and by clustering them. One chance could be that during the cleansing process we dropped part of the log which we considered irrelevant but in reality is not. The heterogeneity of the logs coming from this step anyway let us think that this possibility is very unlikely, yet it has to be considered anyway. About the clustering process instead, one thing that could raise an alarm is the graph of the WSS measure, which had no clearly defined elbow. This could be caused by several reasons: first, as previously mentioned, the log cleansing process could have dropped some important information. Secondly, the vectorization process of each log done with the tf-idf algorithm could actually be harming data, this anyway looks very unlikely to us since we could see that important words, like the severity levels of log4j, are very well distinguished in different clusters of logs. Another motivation could reside into the clustering algorithm which could not be the optimal one for this task. For this reasons different type of clustering, like spectral clustering, have been tried with no different results. The last, and maybe the simplest reason is that

the nature of data simply could not never lead to a clear division between logs thus never resulting in to an evident elbow in the WSS graph.

The last step is anomaly prediction in which we correlate applicative logs to a custom metric, Θ_k, derived from the single anomaly scores calculated during the first step. It is possible that the way we calculated this value could have brought to a loss of information about the real anomaly situation into infrastructural metrics. A cause for this could be the usage of the mean instead of another metric like median or max or, worse, the entire concept revolving around Θ_k is completely wrong and we should have used a completely different approach.

As we have seen, there are many possible reason explaining such poor results. Our strongest hypothesis is that the strongest factor is the fact that data comes from a single machine of a test environment in a cluster of two. We believe that speculating even more than what has already been done on what step could have cause the failure of the anomaly prediction is no use. The first thing to do should be, instead, testing the algorithm using a reliable dataset coming from a production environment from a sufficiently long period, only after having seen these results one could really start making considerations.

Chapter 6

Conclusion

In this thesis we presented our approach to implement an anomaly prediction algorithm by correlating applicative and infrastructural logs. Despite the results were not satisfying we believe that, since we respected the underlying requirement of implementing a generic approach, the steps this work is composed of and the methodologies used could inspire and guide the reader into implementing his own anomaly prediction system. Anomaly detection works in fact agnostically with respect to the metrics recorded by the system while application log parsing performs log cleaning and keyword extraction on the raw text of the logs thus ignoring the protocol they have been collected with. We have therefore implemented and explained an anomaly prediction approach that could be extended at least to any Oracle application running on a machine, since they all follow the same paradigm in terms of information collected, but could also to any other system replicating the same situation.

6.1 Future developments

The most urgent task is to gather enough data from the production environment to test the algorithm on a real case environment. If these new results were satisfying the next step would be deploying the algorithm to predict anomalies for every machine in a cluster. Figure 6.1 shows the architecture we designed.
There would be a separate anomaly prediction model for every machine in the cluster, the whole anomaly prediction system would be hosted and ran on a separate machine. This machine which would gather all information needed by using Splunk API, this includes also infrastructural metrics, which are collected by Oracle Enterprise Manager as explained in Chapter 3 and

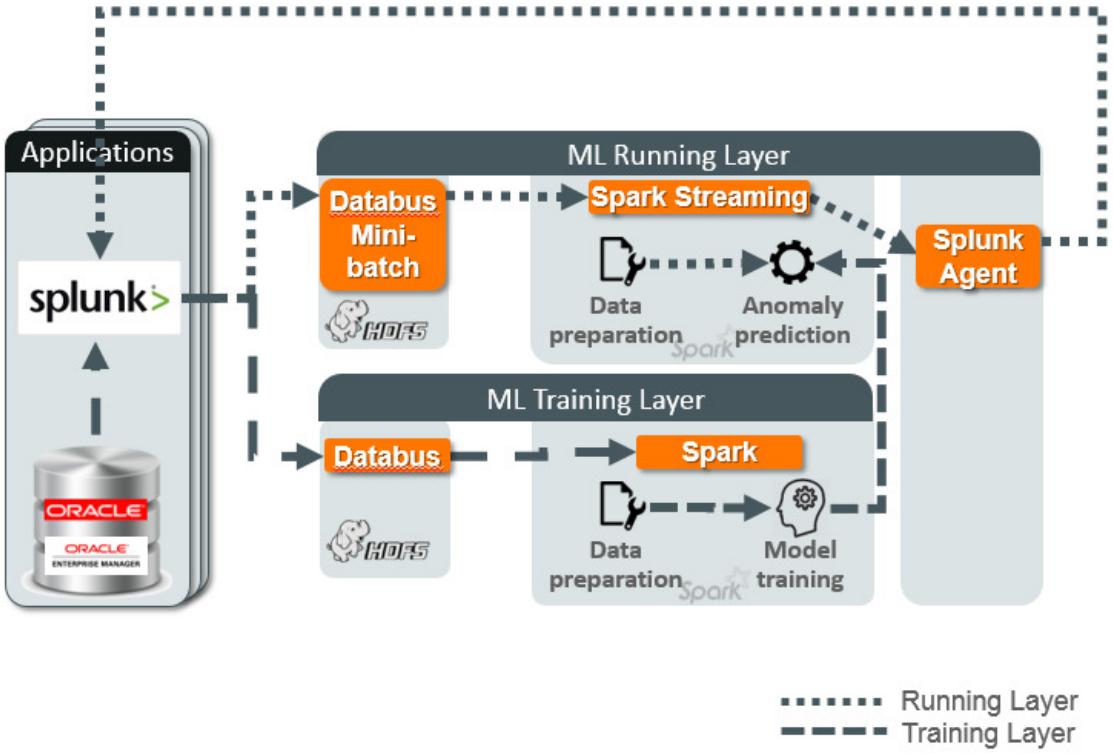

Figure 6.1: Architecture of the anomaly prediction for an enterprise system.

then forwarded to Splunk. This is done because it would be more efficient and easy for the algorithm to have a single point of input instead of several ones. The idea is to perform online anomaly prediction, to achieve this we split the architecture into two parts: a Running Layer and a Training Layer. On the training layer data are read from Splunk and stored into a Hadoop File System from which they are manipulated by Spark to perform the various steps of data preparation listed in this thesis. Once data is prepared we train the different anomaly prediction models by using Spark Regressors from the Spark ML package [21] and then we deploy them.
On the Running Layer instead new data is ingested in a mini-batch of Δt minutes of data and stored again in HDFS, then it is prepared in a similar way to the training layer, sent as input to one of the previously trained anomaly prediction model (depending on the machine the data belongs to) and then the predicted anomaly scores are sent to Splunk again by using a Splunk agent like the one already installed on the machines we analyze. We send data back to Splunk because it can also be used as a dashboard

to visualize current anomaly levels and can be set to send alarms if needed. The two extraction of data, the one from the training layer and the mini-batch of the running layer are separated to not to create excessive overhead while reading data in this last mentioned layer since we want the anomaly prediction to be as immediate as possible.
In order to keep track of every possible changes in the system behavior and of new, unseen events, the idea is to train each anomaly prediction model once in a while, for example once in three weeks. Since it is possible that with the amount of data stored in a production environment the training of the algorithms could require up to hours, this task has to be performed during the night where anomaly prediction is far less crucial. If needed, it is also possible not to train all models together but to distribute the jobs during several nights in order to keep the system running as much as possible.

Bibliography

[1] Liang Tang, Tao Li, Chang-Shing Perng. *LogSig: Generating System Events from Raw Textual Logs*, IBM T.J. Watson Research Center in collaboration with International University of Florida

[2] Qiang FU, Jian-Guang LOU, Yi WANG, Jiang LI1, *Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis*, Beijing University of Posts and Telecommunications.

[3] Risto Vaarandi, *A Data Clustering Algorithm for Mining Patterns From Event Logs*, Tallinn Technical University

[4] Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios *Clustering Event Logs Using Iterative Partitioning*, Dalhousie University

[5] Risto Vaarandi *Mining Event Logs with SLCT and LogHound* Cooperative Cyber Defence Centre of Excellence Tallin

[6] *https://nlp.stanford.edu/IR-book/pdf/06vect.pdf* Standford University open book reference for nlp

[7] Song, X., Wu, M., Jermaine C., and Ranka, S. 2007, *Conditional anomaly detection.* IEEE Trans. Knowl. Data Eng. 19, 5, 631.

[8] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, Jorg Sander, *LOF: Identifying Density-Based Local Outliers*, University of Munich

[9] Jaeshin Lee, Bokyoung Kang, Suk-Ho Kang *Integrating independent component analysis and local outlier factor for plant-wide process monitoring.* Department of Industrial Engineering, Seoul National University, Seoul, 742, Republic of Korea

[10] Yongmin Tan, Xiaohui Gu, Haixun Wang *Adaptive System Anomaly Prediction for Large-Scale Hosting Infrastructures*, Microsoft Research Asia

[11] Xiaohui Gu, Haixun Wang, *Online Anomaly Prediction for Robust Cluster Systems* North Carolina State University

[12] Yongmin Tan, Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Chitra Venkatramani, Deepak Rajan *PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems*, IBM T. J. Watson Research

[13] *https://docs.oracle.com/middleware/12213/wls/index.html* Oracle WebLogic Server manual

[14] *https://docs.oracle.com/cd/E1322201/wls.html* Weblogic Server domains guide

[15] *https://www.oracle.com/engineered-systems/exalogic/index.html* Oracle Exalogic manual

[16] *https://www.splunk.com/* Homepage of Splunk, a log collector.

[17] *http://www.oracle.com/technetwork/oem/enterprise-manager/overview/index.html* Oracle Enterprise Manager Homepage

[18] *https://logging.apache.org/log4j/2.0/* Log4j descriptive page, part of Apache project

[19] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu, *Experience Report: System Log Analysis for Anomaly Detection*, The Chinese University of Hong Kong

[20] Varun Chandola, Arindam Banerjee and Vipin Kumar ,*Anomaly Detection: A Survey* University of Minnesota

[21] *https://spark.apache.org/docs/latest/api/python/pyspark.ml.html* Pyspark machine learning library.

[22] *https://kafka.apache.org/* Kafka, a distributed streaming platform used to build streaming application.