UNIVERSITY OF TRENTO


DOCTORAL THESIS

# Modern Anomaly Detection: Benchmarking, Scalability and a Novel Approach

*Author:*
Sivam PASUPATHIPILLAI

*Advisor:*
Prof. Emanuele DELLA VALLE
*Co-advisor:*
Prof. Yannis VELEGRAKIS

*Industrial advisors:*
Mattia PASOLLI
Michele VESCOVI

ICT International Doctoral School
Department of Information Engineering and Computer Science

November 17, 2020

*"Manners maketh man.*
*Practice makes perfect."*

English aphorisms

*To my family*

UNIVERSITY OF TRENTO

# *Abstract*

ICT International Doctoral School
Department of Information Engineering and Computer Science

Doctor of Philosophy

**Modern Anomaly Detection:
Benchmarking, Scalability and
a Novel Approach**

by Sivam PASUPATHIPILLAI

Anomaly detection consists in automatically detecting the most unusual elements in a data set. Anomaly detection applications emerge in domains such as computer security, system monitoring, fault detection, and wireless sensor networks. The strategic importance of detecting anomalies in these domains makes anomaly detection a critical data analysis task. Moreover, the contextual nature of anomalies, among other issues, makes anomaly detection a particularly challenging problem. Anomaly detection has received significant research attention in the last two decades. Much effort has been invested in the development of novel algorithms for anomaly detection. However, several open challenges still exist in the field.

This thesis presents our contributions toward solving these challenges. These contributions include: a methodological survey of the recent literature, a novel benchmarking framework for anomaly detection algorithms, an approach for scaling anomaly detection techniques to massive data sets, and a novel anomaly detection algorithm inspired by the law of universal gravitation. Our methodological survey highlights open challenges in the field, and it provides some motivation for our other contributions. Our benchmarking framework, named BAD, tackles the problem of reliably assess the accuracy of unsupervised anomaly detection algorithms. BAD leverages parallel and distributed computing to enable massive comparison studies and hyperparameter tuning tasks. The challenge of scaling unsupervised anomaly detection techniques to massive data sets is well-known in the literature. In this context, our contributions are twofold: we investigate the trade-offs between a single-threaded implementation and a distributed approach considering price-performance metrics, and we propose a scalable approach for anomaly detection algorithms to arbitrary data volumes. Our results show that, when high scalability is required, our approach can handle arbitrarily large data sets without significantly compromising detection accuracy. We conclude our contributions by proposing a novel algorithm for anomaly detection, named Gravity. Gravity identifies anomalies by considering the attraction forces among massive data elements. Our evaluation shows that Gravity is competitive with other popular anomaly detection techniques on several benchmark data sets. Additionally, the properties of Gravity makes it preferable in cases where hyperparameter tuning is challenging or unfeasible.

# *Acknowledgements*

This thesis was developed to the best of my abilities over several years of research, and it represents my contribution to the anomaly detection field. In this brief note, I would like to thank all the people who contributed to this work, either directly or indirectly.

The first "Thank you!" goes to my doctoral advisors: professor Emanuele Della Valle and professor Yannis Velegrakis. Without your help I would not have been able to overcome the challenges and to navigate the world of academia. To you goes my most sincere gratitude for assisting me with both your time and experience.

Another "Thank you!" goes to my industrial advisors Mattia Pasolli and Michele Vescovi. Thank you for your friendship, your language classes, and in general for making daily life at the office more enjoyable.

Of course, a big "Thank you!" goes to my family for supporting me and contributing to make me the man I am today.

Finally, I would like to thank all of the people, both friendly and not, I have met in this journey, who have contributed to the achievement of this wonderful goal, by either assisting me, supporting me, or teaching me something new.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# List of Symbols

| | | |
|---|---|---|
| $D$ | data set, data matrix | $D \in \mathbb{R}^{n \times d}$ |
| $n$ | cardinality of $D$ | $n \in \mathbb{N}$ |
| $d$ | dimensionality of $D$ | $d \in \mathbb{N}$ |
| $x_i$ | $i$-th data element (row vector) | $x_i \in D, i \in [0, n)$ |
| $O$ | outlier set | $O \subset D$, $O = D \setminus I$ |
| $I$ | inlier set | $I \subset D$, $I = D \setminus O$ |
| $s$ | outlier scoring function | $s : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times (d+1)}$ |
| $g$ | ground truth function | $g : \mathbb{R}^{n \times d} \to \{0.0, 1.0\}^n$ |

# Chapter 1

# Introduction

As human beings, our curiosity is naturally aroused by rare or unexpected events. Although these events are unexpected "by definition", it is difficult to imagine a physical world phenomenon that cannot produce unexpected results. Even when tossing a balanced coin, an example of a simple and well-understood phenomenon, there exists a small chance for the coin to land on its edge. Naturally, increasing the phenomenon complexity increases the number and complexity of its unexpected outcomes.

Considering the way we perceive our environment, one might argue that all that is unexpected is, in fact, interesting. The rarity of an event makes it worthwhile to experience. The rarity of a substance makes it valuable. The unexpectedness of a situation makes it interesting. Although in this thesis we do not delve into the philosophical, it is nonetheless interesting to reason about the definition of *interesting* and *unexpected*, and how they relate to human intuition.

Instead of dealing with these problems, this thesis deals with the *ability* to recognize the unexpected. This ability is valuable in countless domains. Unexpected medical symptoms might indicate problematic medical conditions [116]. Unexpected Internet traffic might indicate malicious intrusions [63]. Unexpected credit card transactions might indicate fraud attempts [32]. Unexpected sensor readings might indicate the presence of new elementary particles [1].

In data analysis, the task of recognizing unusual behavior is known as *anomaly detection*, or *outlier detection* [3]. More precisely, anomaly detection consists in identifying the most anomalous data elements in a data set. A data set represents a phenomenon in the real world, and data analysis provides a way of understanding the phenomenon by analyzing its data. This process can be applied to any phenomenon as long as there are ways to collect its data.

Anomaly detection represents the binding thread for this thesis. In this thesis, we present our contributions to the anomaly detection field. These contributions are mainly related to three aspects: evaluating anomaly detection technique, or *benchmarking*, scaling anomaly detection techniques to massive data sets, and proposing a novel approach for anomaly detection. The description of these contributions constitutes a large part of the contents of this thesis.

In this introductory chapter, we present the motivation for the work developed in the thesis, formulated as a set of research questions (Section 1.1). In Section 1.2, we introduce the mathematical notation used throughout the thesis, and we formally define the anomaly detection problem. Section 1.3 presents the thesis outline as well as briefly introducing each chapter. Finally, Section 1.4 concludes the chapter by listing the scientific publications related to this thesis.

## 1.1 Motivation

The anomaly detection field has received a lot of attention in the last two decades [25, 80, 39, 164, 75, 3]. This can be related to several causes. The inception of the data analysis field [67], and the increasing availability of data sets, commonly known as *data deluge*, at the beginning of the 21st century increased the interest on all data analysis tasks. Additionally, the large number of applications of anomaly detection and the challenging nature of the problem, sparked the interest of both researchers and practitioners. However, unlike in other data management communities, e.g. the database community[1], these efforts were not coordinated explicitly. This resulted in a large corpus of research following different methodologies and approaches.

This huge corpus motivated the first contribution presented in this thesis, namely a methodological survey of the anomaly detection literature. Several anomaly detection surveys exist [39, 164, 75]. However, few studies focus on methodological aspects. Our initial contribution can be formulated as the following research question:

**Question 1 (Q1)** *What are the most widespread issues in the anomaly detection literature with respect to methodology, evaluation and reproducibility?*

To answer this question, we analyzed a large sample of the anomaly detection literature with respect to four methodological aspects. Our approach, analysis and results are described in details in Chapter 4.

One of the issues highlighted in our survey is the inconsistent use of benchmarks in the literature. This is particularly relevant when algorithms are compared across studies, since different experimental conditions might induce bias in the results.

The second contribution of this thesis focuses on this problem. In particular, we developed a benchmarking framework for anomaly detection, the BAD framework. BAD enables users to execute reproducible anomaly detection experiments on the most widely used benchmark data sets from the literature. BAD was developed to answer our second research question:

**Question 2 (Q2)** *How can we improve the reproducibility of anomaly detection evaluation experiments across different studies?*

The design and implementation of the BAD framework, as well as a comparison study highlighting the advantages of BAD, are described in details in Chapter 5.

The development of this thesis lead us to work on several industrial anomaly detection use cases. These works were developed at the Semantics & Knowledge Innovation Lab (SKIL) of TIM, as well as during an internship at the Artificial Intelligence Center of Excellence (AICE) at F-Secure[2]. One of the main issues we found when developing production-ready anomaly detection systems is the problem of scaling anomaly detection techniques to massive data volumes.

One well-known approach to scaling algorithms is to parallelize or distribute the computation on a cluster of machines [13, 31]. However, in the industrial context a distributed solution might not be optimal. This is due to the fact that in this context the feasibility of a project is related to its monetary cost, and distributed solutions are usually more expensive than single-threaded applications with respect to operational costs. The third contribution of this thesis analyzes this trade-off. In particular,

[1] http://www.tpc.org/
[2] https://www.f-secure.com/

we take an industrial anomaly detection application as a case study and analyze the price/performance trade-off between a distributed solution and a single-threaded application. Our third contribution can be summarized by the following research question:

**Question 3 (Q3)** *At which data volume is a distributed approach more cost-effective than a single-threaded application with respect to an anomaly detection use case?*

Toward answering this question, we compared several single-threaded and distributed application deployments. This work is described in details in Chapter 6.

The problem of scaling anomaly detection techniques to huge data volumes is analyzed more directly in Chapter 7. In this chapter, we propose a distributed formulation for one of the most popular unsupervised anomaly detection techniques, the $k$-nearest neighbors algorithm. Our formulation improves scalability by an order of magnitude without significantly penalizing performance. We also discuss a possible theoretical explanation for this behavior. The contribution described in Chapter 7 revolves around the following question:

**Question 4 (Q4)** *Is it possible to scale the k-nearest neighbors anomaly detection algorithm to arbitrarily large data sets without significant losses in detection accuracy?*

The final contribution of the thesis is a novel anomaly detection algorithm, named Gravity. Gravity relies on a novel definition of outlier. This definition takes inspiration from the law of universal gravitation by defining attractive and repulsive forces between data elements based on their distance and a definition of mass.

The Gravity algorithm is the result of investigating the following question:

**Question 5 (Q5)** *Is it possible to detect outliers in a data set by considering massive data elements and using attractive and repulsive forces as an outlying criterion?*

The Gravity algorithm, as well as a comparison with the current state-of-the-art, are presented in Chapter 8.

## 1.2 Notation and problem statement

This section introduces the mathematical notation used throughout the thesis. All terms introduced here will be detailed in Chapter 2.

The anomaly detection task consists in detecting outliers in a data set. A data set is represented in relational form as a data matrix

$$D \in \mathbb{R}^{n \times d} \tag{1.1}$$

where $n$ and $d$ are the data set *cardinality* and *dimensionality*, respectively. The cardinality refers to the number of data elements, while the dimensionality refers to the number of features of each data element. Data elements are represented as row vectors in $D$. We denote with $x_i$ the $i$-th data elements, $i \in [0, n)$. Unless otherwise stated, we always assume numerical features.

An anomaly detection algorithm can be modeled as a scoring function

$$s : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times (d+1)} \tag{1.2}$$

where the additional feature corresponds to the *outlier score* for each element, as computed by the algorithm. We denote with $s_i$ the outlier score for element $x_i$. We

assume that higher outlier scores correspond to more outlying data elements. Notice that in this formulation, a scoring function produces a ranking instead of a classification. In order to produce a classification from a score ranking, it is possible to introduce an arbitrary threshold $\sigma$, and assume that the outlier set $O$ is

$$O = \{x_i \in D \mid s_i > \sigma\} \tag{1.3}$$

The inlier set $I$ is defined as

$$I = D \setminus O \tag{1.4}$$

A benchmark data set is a data set containing class label information, i.e. each element is classified as either normal or anomalous. We refer to this information as the *ground truth* of the data set. We represent the ground truth $g$ as a function

$$g : \mathbb{R}^{n \times d} \to \{0.0, 1.0\}^n \tag{1.5}$$

where 1.0 represents true outliers and 0.0 represents true inliers.

The correctness of an outlier detection algorithm can be established by comparing the results of the algorithm on a benchmark data set with its ground truth. Performance metrics provide a quantitative measure of correctness.

We represent a performance metric as a function

$$m : \{0.0, 1.0\}^n \times \mathbb{R}^{n \times (d+1)} \to \mathbb{R} \tag{1.6}$$

Intuitively, given a ground truth vector $g(D)$, and a score matrix $s(D)$, the metric $m(g(D), s(D))$ provides a quantifiable measure representing the quality of the outlier scores on data set $D$.

We can now formally define the anomaly detection problem. We provide the statement for unsupervised anomaly detection. The unsupervised approach is the most common approach, since it does not require a training data set with ground truth to be available. The unsupervised anomaly detection problem can be stated as follows:

**Problem 1 (Unsupervised Anomaly Detection)** *Given a benchmark data set $D \in \mathbb{R}^{n \times d}$ with ground truth $g : \mathbb{R}^{n \times d} \to \{0.0, 1.0\}^n$, and a metric $m : \{0.0, 1.0\}^n \times \mathbb{R}^{n \times (d+1)} \to \mathbb{R}$, find a scoring function $s^*$ from the family $\mathcal{F} = \{f \mid f : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times (d+1)}\}$ such that*

$$s^* = \arg\max_{s \in \mathcal{F}} \{m(g(D), s(D))\} \tag{1.7}$$

We refer to Problem 1 throughout the thesis. Notice that this formulation does not assume any particular algorithm, modeled by the scoring function $s(\cdot)$. Thus, it is applicable to any unsupervised anomaly detection technique.

## 1.3 Contributions and thesis outline

The rest of the thesis is organized as follows:

- **Chapter 2: Anomaly Detection -** In this chapter we introduce the anomaly detection field. We analyze anomaly detection with respect to other data analysis tasks, i.e. classification and clustering. We present the anomaly detection approaches that have been investigated in the literature with respect to learning paradigm, anomaly definitions and evaluation. Finally, we conclude the chapter by describing the most successful applications of anomaly detection.

- **Chapter 3: State of the Art -** In this chapter we present the most seminal works in anomaly detection. We begin the chapter by presenting a brief history of the anomaly detection field. In particular, we highlight a set of key milestones and paradigm shifts in chronological order.

  We continue by describing some of the most seminal papers in the field, which contributed to shape the current state of the anomaly detection field. Then, we present some of the resources and tools available to anomaly detection practitioners and researchers. We conclude the chapter by presenting some of the most important open research challenges in the field.

- **Chapter 4: A Methodological Survey of Anomaly Detection [Q1] -** In this chapter we describe our first original contribution to anomaly detection, namely our methodological survey of the state of the art.

  Contrary to other surveys [25, 80, 39, 164], we focus on the research methodology instead of focusing on the algorithms proposed. Our aim is to highlight potential issues with the way modern anomaly detection research is carried out. We select *impactful* papers published from 2007 to 2017, where the impact is based on the number of citations. Around 760 papers were analyzed in total. For each surveyed paper, we analyze four aspects: application domain, formal problem statement, data representation, and evaluation methodology.

  Our results provide several insights. Although the anomaly detection task is very general, it has been investigated in only a limited number of application domains. Several formal problem statements exists. In fact, it is common for each novel approach to propose a new outlier definition [89, 33, 132, 11, 108]. We propose a novel classification of anomaly definitions that provides some insight and improves comparison studies in the field. We also find several issues with the evaluation methodology of anomaly detection techniques.

- **Chapter 5: BAD: Benchmarking for Anomaly Detection [Q2] -** In this chapter we present BAD, our Benchmarking framework for Anomaly Detection. The BAD framework was motivated by some of the challenges highlighted in our survey. BAD is a distributed framework for benchmarking and hyperparameter tuning of anomaly detection algorithms.

  BAD enables the execution of extensive hyperparameter searches, as well as comparison studies between different algorithms. BAD relies on a selected collection of benchmark data sets, as well as on plain text interfaces. This makes it easy to produce, share and replicate results.

  This chapter provides a thorough description of BAD, as well as an analysis on the issues related with the evaluation of anomaly detection techniques. We also present an exemplar comparison study using some of the most popular anomaly detection algorithms.

- **Chapter 6: Cost-aware Data Analysis [Q3] -** This chapter describes our first contribution toward the scalability of anomaly detection techniques. It is well known that some of the most popular anomaly detection algorithms have a worst-case quadratic complexity [33, 132]. Thus, they do not scale to massive data volumes. This is particularly problematic for industrial applications, where it is not uncommon to produce large amount of data even from modestly-sized applications. One generally accepted solution to scaling up algorithms

is to partition the problem into smaller instances and to distribute the computation on several machines. We analyze this approach with respect to anomaly detection along two perspectives.

In this chapter, we analyze the trade-offs between a single-threaded application and a distributed solution for an anomaly detection task. We base our analysis on price-performance metrics [72, 114, 31]. We adopt this approach since cost is an important factor when selecting the best solution for an industrial use case. Our results show that a distributed approach, while speeding up the computation, is not always the best choice in term of price performance. In fact, for smaller data volumes a single-threaded application is more cost effective, if we can afford to wait a longer time for a solution.

- **Chapter 7: Scalable Unsupervised Anomaly Detection [Q4] -** This chapter continues the analysis on the scalability of anomaly detection algorithms. Here, we propose a scalable formulation of the famous *k*-nearest neighbors algorithm. Our experiments on synthetic and real-world benchmark data sets show how our approach can scale up to massive data volumes without significant losses in accuracy.

- **Chapter 8: Gravity-based Anomaly Detection [Q5] -** This chapter presents the final contribution of this thesis, the Gravity algorithm for unsupervised anomaly detection. This approach is inspired by the law of universal gravitation. In particular, Gravity detects outliers by considering attraction forces between data elements whose expression is derived from the expression for the gravitational force.

  In this chapter, we describe the Gravity algorithm and we present an extensive experimental comparison with popular anomaly detection algorithms. Our results show that our proposed algorithm is competitive with the state-of-the-art. Additionally, Gravity properties makes it a good fit for situations where hyperparameter tuning is unfeasible, or where a low false positive rate is required.

- **Chapter 9: Conclusions -** This chapter concludes the thesis. Here we summarize our main contributions and results. Finally, we present the limiting assumptions of our contributions and present interesting future directions.

## 1.4 List of publications

The work described in this thesis relates to the following peer-reviewed publications:

- Published - Marco Balduini, Sivam Pasupathipillai, and Emanuele Della Valle. "Cost- aware streaming data analysis: Distributed vs single-thread". *In: Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. 2018, pp. 160–170 [22].

- Accepted for publication - Sivam Pasupathipillai, and Emanuele Della Valle. "BAD: a Benchmarking Framework for Unsupervised Anomaly Detection". *In: Big Data Research - Special issue on Benchmarking, Performance Tuning and Optimization for Big Data Analytics*.

- Accepted for publication - Sivam Pasupathipillai, and Emanuele Della Valle. "Approximate Distance-based Anomaly Detection at Massive Scale". *In: Proceedings of the 9th Workshop on Scalable Cloud Data Management*.

# Part I

# Background

# Chapter 2

# Anomaly Detection

In this chapter, we introduce the anomaly detection field. Section 2.1 introduces the problem, its fundamental concepts, and its relation with other data analysis tasks. In Section 2.2, we present some of the most successful applications of anomaly detection techniques.

## 2.1 Introduction

Anomaly detection, or outlier detection, consists in automatically identifying abnormal data elements in a data set. These abnormal elements are commonly referred to as *anomalies* or *outliers*. In the following, we will use these terms interchangeably.

Detecting outliers is valuable in several domains. In some cases, outliers might represent measurement errors, faults, or other critical conditions. In these cases, early outlier detection can help to prevent system failures and economical losses. In other cases, outlier analysis can provide insights on the phenomenon under consideration.

As humans, we are particularly skilled at detecting abnormal patterns. Unfortunately, our attention span does not scale to large amounts of information. For this reason, anomaly detection techniques are used to automatically detect unusual patterns, or to filter the data and provide the analyst with a more manageable data set to further analyze.

Notice that our ability to detect abnormal patterns relies on intuition. As an example, consider the data set depicted in Figure 2.1. The figure depicts a simple 2-dimensional data set. Assume that we want to detect outliers from this data set. A quick visual inspection shows that elements **a1** and **a2** are isolated with respect to the majority of the data elements, and therefore, intuitively, anomalous. Notice that we have not defined formally what "anomalous" means, or what characteristics outliers should satisfy.

Now consider a slightly different example, depicted in Figure 2.2. The figure represents the same data set with the addition of element **b1**. Notice that in this case, our intuition alone cannot provide a crisp classification. Element **a2** is still an outstanding outlier. However, the outlierness of **a1** is not so obvious anymore, and element **b1** could either be considered an outlier or an inlier.

In fact, if we consider **b1** as an outlier, we would also have to reconsider other elements at the frontier of the cluster (e.g. **c1** and **c2**). This problem relates to the following question:

*How far from the center of the cluster would an element have to be to be considered an outlier?*

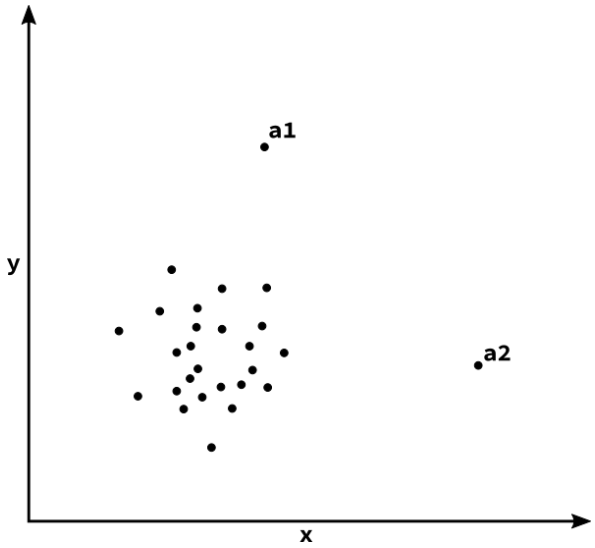


FIGURE 2.1: Simple 2-dimensional data set consisting of a cluster and two outstanding outliers.

Of course, the answer to this question can only rely on intuition. Assume we select a real-valued distance threshold $\bar{d}$ as the answer. This choice would be completely arbitrary, in the sense that even slight changes in the data set might make us question our decision. Since outliers represent abnormality, and our distinction between normal and abnormal behavior is fuzzy, outliers can only be defined arbitrarily.

This example illustrates two fundamental concepts in anomaly detection: i) the abnormality of a data element is *contextual* to the other elements in the data set, ii) the definition of outliers is a fuzzy concept, thus formally defining outliers requires arbitrary choices. These concepts characterize anomaly detection with respect to other data analysis task. We will describe these relations in Sections 2.1.1 and 2.1.2. We return to the problem of formally defining outliers in Section 2.1.4.

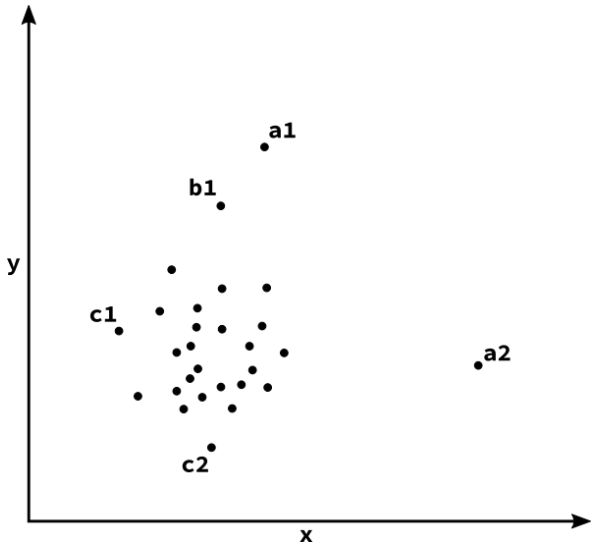


FIGURE 2.2: Simple 2-dimensional data set consisting of a cluster and one outstanding outlier. Classification of other elements as outliers is arbitrary.

### 2.1.1 Relation with classification

Anomaly detection is closely related to binary classification [54]. Binary classification consists in assigning data elements to two distinct classes, based on their features. The similarity with anomaly detection holds when the two classes can be thought of representing normal versus abnormal behavior. For instance, classifying patient medical conditions, or classifying spam emails, are applications where both binary classifications and anomaly detection techniques can be applied [145].

The difference between anomaly detection and binary classification can be described through the different assumptions of the two approaches. Binary classification is a supervised learning task. It relies on a data sample, i.e. the *training set*, consisting of examples of data elements from each class. By analyzing the training set, a binary classification algorithm builds a model that can be used to classify unseen data elements. The richer the training set, the more accurate the model can be. There are no assumptions on the frequency of each class in the training set. Ideally, both classes are represented equally.

On the other hand, anomaly detection has been investigated using different approaches. These include statistical methods, supervised, as well as unsupervised learning approaches. The reason for this is that in many anomaly detection domains, anomalies are either rare or difficult to define. In these domains, it is easier to define anomalies as "What is abnormal?", instead of providing a representative data set. For these reasons, most anomaly detection techniques rely on the following assumptions:

- Anomalies are rare. It is assumed that anomalies represent the minority of the data set.
- If a training set is available, it either contains few anomalies, or none at all.

In several contexts, a training set is not available, thus unsupervised learning techniques must be adopted. We discuss anomaly detection approaches in greater details in Section 2.1.3.

One last important connection between anomaly detection an binary classification relates to evaluation and benchmarking. Since anomaly detection data sets are rare, it is common in the literature to use binary classification data sets to evaluate anomaly detection algorithms [147, 59, 37, 145]. These data sets contain the actual classification for each element. This information can be used to assess the quality of a outlier detection solution on the data set. In some cases, classification data sets must be preprocessed in order to account for the different assumptions of anomaly detection methods, e.g. class imbalance [59, 37]. We describe the evaluation of anomaly detection techniques in Section 2.1.5.

### 2.1.2 Relation with clustering

Data clustering consists in identifying groups of similar elements (*clusters*) in a data set [85]. Identifying clusters and identifying outliers can be seen as two sides of the same coin, since outliers can be defined as elements which do not belong to any cluster, i.e. isolated elements [64].

Both clustering and anomaly detection follow an unsupervised learning approach. This means that clusters are not defined by examples in a training set, but by a cluster definition. This is similar to the definition of anomalies, and it relates to the fact that there is no unique way of defining clusters, since our understanding of what constitutes a group is fuzzy and relies on intuition [65].

Clustering algorithms have been applied to anomaly detection in several studies [64, 63, 53, 129]. However, clustering algorithms are optimized for clustering. This causes issues when, for example, outliers are not completely isolated, but form small outlying groups. Also, the choice of clustering algorithm affects which elements are considered outliers.

### 2.1.3 Anomaly detection approaches

In this section, we discuss the most common approaches to anomaly detection. Each approach rely on different assumptions. In some contexts, we can choose between different approaches, while in others, we might be forced to use one, depending on the available data.

**Supervised learning** Supervised learning approaches for anomaly detection are somewhat rare. This is due to the rarity of supervised data sets containing examples of anomalous behavior. In fact, if we had access to such data, a better approach would consist in using binary classification techniques. In case the training data set is unbalanced, these techniques can be improved by implementing a data balancing scheme [40].

One disadvantage of this approach is that the model specializes in detecting the anomalies represented in the sample data, thus losing the ability to detect previously unseen abnormal behavior. This is a key strength of other anomaly detection approaches. Some studies have also investigated the reduction of anomaly detection to a supervised learning problem [70, 126].

**Semi-supervised learning** In the context of anomaly detection, semi-supervised learning approaches consist in building an anomaly detection model using a data sample containing only normal data [120]. This approach is convenient since in many applications normal data can be obtained inexpensively, while producing example of anomalies requires a large effort.

One drawback of the semi-supervised approach is that, in some contexts, it might be unfeasible to produce a sample containing exclusively normal data. Without this guarantee, outliers in the training set might compromise the model's ability to discriminate between normal and abnormal behavior. Artificial neural networks algorithms have been applied to anomaly detection following the semi-supervised approach [158, 9].

**Unsupervised learning** Unsupervised learning is the most common approach to anomaly detection [33, 161, 63, 37, 69]. This approach does not rely on the availability of information on the actual classes in the training set. Without this information, an unsupervised anomaly detection algorithm must rely on the structural patterns in the data set to provide a classification (e.g., classify as outliers those elements that are isolated with respect to the majority of the data set, see Figures 2.1 and 2.2).

This approach is very popular since it can be applied to any data set. It is also challenging, since, as in any unsupervised learning task, the output must rely on an outlier definition instead of on examples of abnormal behavior in the training set. Whether outliers satisfying the outlier definition are also interesting outliers with respect to the application domain is not always obvious.

Most of the outlier detection literature deals with unsupervised learning. Some example of this are statistical methods [161], where outliers are classified as elements

lying in low probability regions in the feature space, distance-based methods [89, 132], where outliers are classified as elements far away from others, and clustering-based methods [129, 63], where outliers are elements which do not belong to any clusters. All contributions presented in this thesis refer to unsupervised outlier detection. For this reason, in the following we focus exclusively on the unsupervised learning paradigm.

### 2.1.4 Anomaly definitions

Unsupervised outlier detection algorithms rely on a formal anomaly definition to identify outliers in a data set. Several definitions have been proposed in the literature [89, 33, 108]. This variety of definitions stems from the fact that there does not exist one unique way of formally defining outliers. This is due to the fact that our intuition of what constitutes an outlier is contextual and fuzzy. This can be related to the problem of defining clusters in cluster analysis [65].

One important thing to notice is that data elements satisfying an anomaly definition do not necessarily correspond to outliers. For example, consider a data set in which outliers form a small cluster. In this case, the definition "outliers are elements which do not belong to any cluster" [64], and the algorithm implementing this definition, would not be useful in discriminating outliers and inliers. Unfortunately, it is impossible to know a priori whether a given definition of outlier is useful for a particular data set. We discuss evaluating outlier detection algorithms, as well as definitions, in Section 2.1.5.

The most widely quoted informal anomaly definition is the one by Hawkins [78]. The definition is the following:

> An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.

Notice that this definition is intuitive, as it refers to human *suspicion*, and it does not provide any quantitative measure of outlierness.

Since machines cannot rely on intuition, a formal definition is needed to design an algorithm for automatic anomaly detection. In the following, we discuss the most successful formal anomaly definitions proposed in the literature. Notice that all these definitions are parametric, in the sense that they depend on one or more user-defined parameters. These parameters model the fuzzy nature of anomalies.

**Statistical outliers.** Statistical outliers are the most intuitive and well-known definition of outliers. Unsurprisingly, the definition of statistical outliers originated in Statistics, much earlier than the development of the data analysis field [73, 78].

Statistical outliers are defined with respect to the data distribution of the data set. In this context, a data set is considered as a sample generated by the true underlying data distribution. This distribution can be either known (parametric statistical methods), or unknown (non-parametric statistical methods).

A simple example of a parametric statistical outlier definition [23] is:

**Definition 1 ($z_{score}$-outlier)** *An element $x$ in data set $D$ is a $z_{score}$-outlier if its $z_{score}$ is larger than three.*

where the $z$-score of sample $x$ is defined as

$$z_{score}(x) = \frac{|x - \mu|}{\sigma} \tag{2.1}$$

and where $\mu$ and $\sigma$ are the sample mean and standard deviation of the data distribution. Definition 1 assumes that the data follows a Gaussian distribution.

Notice that the value three in the definition is arbitrary. This value is selected following the empirical rule, stating that for Normally distributed data, 99.7 percent of the data distribution falls within three standard deviations from the mean. However, there could exist interesting outliers closer to the mean which would go undetected by this definition. Statistical definitions of outliers have been investigated in several studies [62, 161, 137, 123].

**Distance-based outliers.** The distance-based outlier definition was originally proposed by Knorr and Ng [89]. The definition is:

**Definition 2 ($DB(\rho, dist)$-outlier)** *An element $x$ in data set $D$ is a $DB(\rho, dist)$-outlier if at least a fraction $\rho$ of the elements in $D$ lies at distance greater that $dist$ from $x$.*

This is historically the first definition explicitly considering the distance between two data elements as an outlying criterion. However, Definition 2 is not widely used, due to the difficulty in setting the $\rho$ and $dist$ parameters for arbitrary data sets.

A more widely used distance-based definition is the following [132].

**Definition 3 (KNN-outlier)** *An element $x$ in data set $D$ is a KNN-outlier if there are no more than $n-1$ other elements in $D$ with a larger distance to their $k$-nearest neighbor.*

this definition is more easily applicable with respect to Definition 2, since it only requires to set the $k$ parameter.

**Density-based outliers.** The concept of density-based outliers was originally proposed by Breunig *et al.* [33]. Density-based outliers are generally associated with the Local Outlier Factor (LOF) algorithm [33]. Density-based outliers are formally defined as follows:

**Definition 4 (LOF-outlier)** *An element $x$ in data set $D$ is a LOF-outlier if there are no more than $n-1$ other elements in $D$ with a larger LOF value.*

where the LOF value of a data element is defined as in [33].

Similarly to distance-based techniques, density-based algorithms also consider distances between data elements. Contrary to distance-based methods, outliers are detected considering local properties, where locality is defined with respect to the $k$-nearest neighborhood of each data element [141]. The LOF algorithm is probably the most popular algorithm for unsupervised anomaly detection.

**Isolation-based outliers.** More recently, the isolation-based outlier definition was proposed by Liu *et al.* [108]. Intuitively, an element is an isolation-outlier if it can be easily isolated by the rest of the data set. In the IForest algorithm [108], elements are isolated by performing random, axis-parallel cuts in the feature space. Each element is assigned a score representing the difficulty of isolating it from other elements. Isolation-based outliers can be defined as follows:

**Definition 5 (Isolation-outlier)** *An element $x$ in data set $D$ is an Isolation-outlier if there are no more than $n-1$ other elements in $D$ that can be isolated more easily.*

Isolation-based methods have the advantage of not having to compute directly all pairwise distances between data elements. However, in higher dimensional space, the procedure of isolating an element still suffers from the curse of dimensionality [28, 5].

**Other outlier definitions.** Several other less popular outlier definitions have been proposed. In the context of artificial neural networks, outliers can be defined as data elements producing high residuals when fed into an Autoencoder network [79, 159, 41]. This approach consists in training the network on a data set consisting of mostly normal data elements. The Autoencoder architecture specializes in reproducing its input as output. This has applications in dimensionality reduction, since the hidden layers can provide a compressed representation of the input data.

The key assumption of these approaches is that the representation of outliers and normal elements is significantly different. Therefore, if the network is trained to represent normal elements as accurately as possible, it will produce large residuals when representing outliers. These residuals can then be used as a measure of outlierness.

Other algorithms look for outliers in subspaces of the full feature space [101, 119, 139]. This approach has gained popularity with the development of ensemble methods for outlier detection [7, 6]. The intuition behind this approach is that the feature space might contain features that are irrelevant, or worse, that introduce noise to the anomaly detection task. Thus, one approach consists in randomly subsetting the feature space and consider several anomaly detection tasks in each resulting subspace [101]. One open challenge with this approach is how to meaningfully split the feature set, and how to aggregate the subspace results to maximize detection accuracy [101, 86, 135].

### 2.1.5 Evaluation

Most anomaly detection techniques follow an unsupervised learning approach. This poses several challenges to the experimental evaluation of proposed algorithms. The evaluation of supervised learning approaches rely on the availability of a labeled data set, containing, for each data element, information on whether it is normal or anomalous. This information is commonly known as the *ground truth* of the data. In the unsupervised paradigm, we usually do not possess this information. Thus, in the general case, we cannot assess whether an algorithm correctly detects outliers.

The most well-established approach to solve this issue consists in leveraging binary classification data sets. These data sets contain the ground truth, since binary classification algorithms rely on this information to be present in order to build a supervised classification model. These data sets, referred to as *benchmark* data sets or simply *benchmarks*, have been extensively used in the anomaly detection literature to

TABLE 2.1: Classifier confusion matrix.

| | **Positive (Actual)** | **Negative (Actual)** |
|---|---|---|
| **Positive (Pred.)** | TP | FP |
| **Negative (Pred.)** | FN | TN |
| **Total** | P | N |

assess the performance of anomaly detection algorithms [37, 69, 147]. An interesting approach that does not require benchmark data sets has also been proposed [111].

Apart from benchmark data sets, the evaluation of outlier detection techniques also requires appropriate performance metrics [72]. Most metrics for anomaly detection are defined on the algorithm confusion matrix. An example confusion matrix is presented in Table 2.1. The confusion matrix depicts a classifier performance with respect to the ground truth. Its components are:

- **True positives (TP) -** number of anomalous data elements correctly identified as outliers.
- **False positives (TP) -** number of normal data elements incorrectly identified as outliers.
- **False negatives (FN) -** number of anomalous data elements incorrectly identified as inliers.
- **True negatives (TN) -** number of normal data elements correctly identified as inliers.
- **Outliers (P) -** total number of actual outliers.
- **Inliers (N) -** total number of actual inliers.

Several popular metrics in binary classification are not appropriate for anomaly detection. For example, the *accuracy* metric, defined as:

$$accuracy = \frac{TP + TN}{P + N} \tag{2.2}$$

is not appropriate for anomaly detection, since $N \gg P$. Thus, an algorithm assigning the majority class ("inlier") to all elements, would obtain an high accuracy while incorrectly classifying all outliers.

More appropriate metrics are the *precision* and *recall* [49]. Precision is defined as:

$$precision = \frac{TP}{TP + FP} \tag{2.3}$$

Precision refers to the ability of a detector to minimize false positives, i.e. false alarms. Recall (or *true positive rate*) is defined as:

$$recall = TPR = \frac{TP}{P} \tag{2.4}$$

Recall refers to the ability of a detector to minimize false negatives, i.e. undetected outliers. We can also define the *false positive rate* as:

$$FPR = \frac{FP}{N} \tag{2.5}$$

All of these metrics are defined for a binary classifier output. However, as already discussed, the definition of anomaly is fuzzy, since it relies on intuition. Therefore, outlier detection algorithms usually provide a ranking for each element based on its *outlier score* [33, 108]. This score represents the abnormality of an element with respect to the data set. This outlier score ranking can be converted into a binary classification by introducing a threshold value $\sigma$, so that outliers are elements whose outlier scores is larger than $\sigma$. However, notice that selecting a given value for $\sigma$ is arbitrary.

One approach to solve this issue is to use metrics that are defined directly on the outlier ranking, and whose value is averaged over all possible $\sigma$ values. The most popular metric for anomaly detection is the area under the ROC curve (ROC AUC) [66]. The ROC curve is a curve in the FPR-TPR plane. Each point in the curve corresponds to setting $\sigma$ to a value in the range from the minimum outlier score to the maximum. The curve is obtained by interpolating the obtained points. Thus, the ROC AUC is an aggregated metric representing the average performance of the classifier. An example of ROC curve is presented in Figure 2.3.

One drawback of the ROC AUC metric is the overestimation of the area under the curve due to the interpolation of its points. Thus, large values of the ROC AUC should be taken with consideration. Another drawback is that the curve only considers the recall as a measure of positive performance, without considering the classifier's precision.

A complementary metric is the *average precision* score [49]. This can be defined as the area under the Precision-Recall (PR) curve in the precision-recall plane. One advantage of the average precision score is that it can be computed directly, without interpolating the points of a curve. It also takes into consideration the precision of an algorithm, thus it is more reliable when evaluating algorithms with respect to the number of false positives. An example of precision-recall curve is presented in Figure 2.4.

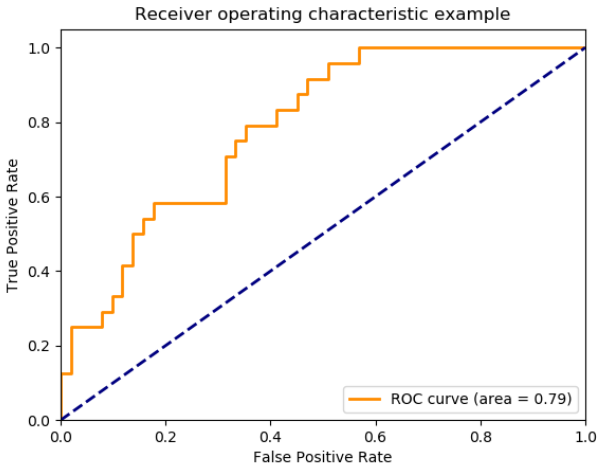


FIGURE 2.3: Example of Receiver Operating Characteristic (ROC) curve.

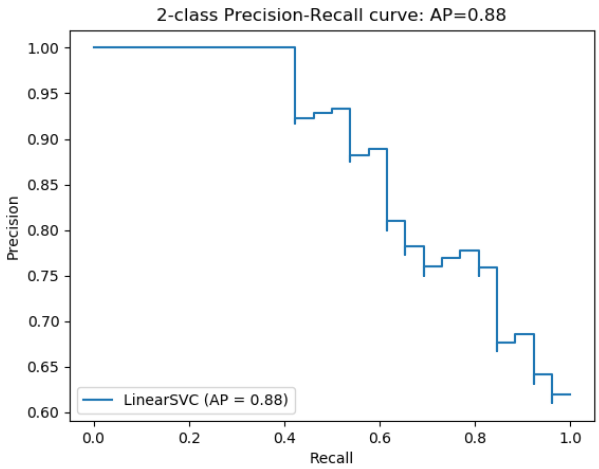


FIGURE 2.4: Example of Precision-Recall (PR) curve.

## 2.2 Applications of anomaly detection

Applications for anomaly detection techniques are countless. In fact, anomaly detection can be applied to any scenario in which there is interest to automatically detect unusual events or behavior from data. In the following, we discuss some of the most widespread applications in the literature.

### 2.2.1 Intrusion detection

Anomaly detection has been extensively studied in the context of intrusion detection [144, 102, 147]. In this context, anomalies represent malicious attempts to infiltrate a system or network. Each intrusion might cause great damage, thus detecting intrusions in a timely manner is of great value.

Another reason for the popularity of this application domain is the availability of the DARPA intrusion detection data set [106]. This data set is one of the few examples of a data set specifically developed for evaluating anomaly detection systems. Although this data set presents several flaws [113, 148], it is one of the most widely used evaluation data set in the literature.

### 2.2.2 Sensor networks

Sensor networks have gained popularity with the spread of interconnected sensor devices. These sensors provide a stream of data that can be analyzed in real-time to gather insight on the monitored environment. Detecting abnormal events in the monitored environment is a natural application of anomaly detection [19]. Anomaly detection in sensor networks present unique challenges with respect to other domains. For example, wireless sensors are constrained with respect to network bandwidth and energy consumption, as well as having relatively high failure rates. For these reasons, one of the most common use cases for anomaly detection in sensor networks is to detect measurement errors or misbehavior or faulty sensors [123, 131, 57, 19].

### 2.2.3 Fault detection and system monitoring

Complex systems, both hardware and software, require extensive monitoring and maintenance to guarantee an adequate quality of service. The attention level of human beings cannot scale to the huge number of metrics that need to be monitored in a complex system. Therefore, anomaly detection methods have been investigated to provide reliable monitoring and fault detection.

One advantage of anomaly detection techniques is that, in some cases, it is possible to detect a fault before it becomes critical and it causes damage to the system. For example, it is common for hardware components to misbehave for a given period of time before failing. Thus, detecting misbehavior can identify components that needs to be substituted before a system-compromising failure happens.

Some example applications of anomaly detection techniques applied to fault detection include aircrafts maintenance [47], cloud monitoring [152], autonomous robots [125], and hard disk drives monitoring [154].

# Chapter 3

# State of the Art

waIn this chapter, we present the current state of the art of anomaly detection. Section 3.1 begins the chapter by presenting some historical milestones in the field. In Section 3.2, we present some of the most seminal papers. Section 3.3 presents some of the most popular resources and tools available to anomaly detection researchers and practitioners. Finally, Section 3.4 concludes the chapter by presenting some of the most relevant open challenges. As already mentioned in Chapter 2, in the following we focus on unsupervised anomaly detection.

## 3.1 A brief history of anomaly detection

The anomaly detection problem can be traced back to the problem of detecting outliers in statistical samples [73, 78, 25]. The term outlier was coined in the 17th century, when it was used to refer to foreigners, outsiders or nonconformist people. The statistical use of the term is more recent and can be found from the first half of the 20th century [149].

In Statistics, outliers are often considered as spurious data samples, e.g. measurement errors. Thus, the problem of outlier classification was generally associated with data cleaning [149, 73]. This is particularly important as it is well-known that several statistical techniques are not robust to the presence of outliers in the data sample [38].

The meaning and value of outliers were reconsidered at the beginning of the 21st century [89, 33]. Thanks to the development of information technology and the wide availability of data sets, the considerations of outliers shifted from data errors to be removed, to valuable nuggets of information to be mined. This paradigm shift originated in application domains such as intrusion detection [106], and fraud detection [32], where outliers represent critical and potentially catastrophic events. In these contexts, the identification of outliers is obviously of great value.

Anomaly detection evolved from Statistics also with respect to methodologies. Initially, anomaly detection techniques were relying on knowing [73] or estimating [161] the data distribution. This statistical approach was challenged by Knorr and Ng [89], which introduced a novel definition of outliers based on distances between elements in the data set.

This breakthrough was shortly followed by an explosion of different anomaly definitions, which did not rely on explicitly modeling the true data distribution [33, 132, 8, 11, 124]. These approaches were conceived within the newly born field of data mining [67].

The development of novel algorithms and anomaly definitions did not stop. Among these, two of the most notable contributions are: the work of Hawkins *et al.* [79], which originally proposed the definitions of outliers based on residuals of an autoencoder neural network, and the work of Liu *et al.* [108], which introduced

the concept of *isolation* to avoid the burden of computing all pairwise distances in the data set, typical of distance and density-based techniques [33, 132]. The work of Hawkins *et al.* [79] paved the way for the more recent research on deep anomaly detection [158, 168].

Apart from the proposal of novel anomaly definitions, the most disruptive happening in the field from the beginning of the century was probably the development of a corpus on ensemble methods for anomaly detection [101, 119, 4, 169, 139, 6]. Ensemble methods are particularly well-suited for anomaly detection, since the randomized nature of ensemble components is able to better capture the fuzzy nature of outliers with respect to deterministic algorithms.

## 3.2 Seminal papers

This section describes seminal papers on unsupervised anomaly detection. Each paper either presented one of the most popular algorithms in the current state of the art, or introduced a paradigm shift in the literature.

**Breunig *et al.* [33], Identifying density-based local outliers.** The Local Outlier Factor (LOF) algorithm [33] is one of the most popular algorithms for unsupervised anomaly detection. LOF is a ranking method which assign an outlier score to each data element. The LOF score depends on the relative $k$-nearest neighborhood density between the elements and its $k$ nearest neighbors. The LOF value is close to one for elements in a dense cluster and it is higher for isolated elements [33]. Since it considers relative densities with respect to a $k$-nearest neighborhood, LOF can be classified as a local density-based algorithm.

Although it was proposed almost two decades ago, LOF has stood the test of time and it is still one of the most popular baseline algorithms used in the literature. One drawback of LOF is that it requires the user to select the value of the $k$ hyperparameter. The optimal value of $k$ depends on the data set under consideration, and an appropriate setting can only be discovered empirically.

With respect to algorithm comparison, the most extensive comparison study in outlier detection by Campos *et al.* [37] presents LOF as being one of the best performing methods on the considered benchmarks, along with KNN [132] and its variants. In particular, these algorithms stand out both when performance is averaged over different values of $k$, and when $k$ is selected to be optimal in a given range of values.

Another comparison study by Goldstein and Uchida [69] found that LOF performance is often superior or comparable with other local nearest-neighbors based algorithms, however it fails to accurately detect global outliers.

A more recent evaluation by Domingues *et al.* [52] show that LOF underperforms with respect to IForest [108], and other methods on the considered benchmarks. However, these results might be influenced by the arbitrary choice of the $k$ parameter used in the analysis. LOF has also been found to outperform other algorithms on benchmarks specific to the intrusion detection domain [102].

**Ramaswamy *et al.* [132], Efficient algorithms for mining outliers from large data sets.** The most popular distance-based algorithm is the KNN algorithm by Ramaswamy *et al.* [132]. Instead of following the original distance-based definition proposed by Knorr and Ng [89], the authors propose a simpler definition. The outlier score in their formulation is based on the distance between an element and its $k$-th nearest neighbor.

This simple definition is extremely useful in practice. In fact, the KNN algorithm has been found to outperform other techniques in different comparison studies [37, 69]. Additionally, KNN has been found to be more robust with respect to the $k$ hyperparameter setting with respect to other algorithms [37]. Notice that other variants of the original distance-based approach have been proposed [11, 26]. However, the proposed methods have not been found to consistently outperform the original KNN algorithm.

**Hawkins *et al.* [79], Outlier detection using replicator neural networks.** The idea of using replicator neural network for anomaly detection was pioneered by Hawkins *et al.* [79]. This approach consist in training an autoencoder architecture to represent as closely as possible the data elements, and then, detect outliers as elements who have large average residuals with respect to their reconstructed representation.

Considering its differences with respect to other unsupervised anomaly detection techniques, this algorithm is not usually included in comparison studies. Thus, we cannot report its performance with respect to the current state of the art.

**Lazarevic and Kumar [101], Feature bagging for outlier detection.** The feature bagging method [101] is a meta-algorithm for anomaly detection. Feature bagging is one of the first approaches investigating the idea that outliers might be easier to find in subspaces of the full feature space. This is especially true when the data set presents a large number of noisy features. As a meta-algorithm, feature bagging can be applied to other algorithms to improve their detection accuracy. This procedure can be related to ensemble methods, since each subspace can be considered as an independent ensemble component [6].

**Liu *et al.* [108], Isolation forest.** The isolation forest (IForest) algorithm by Liu *et al.* [108] is one of the most recent milestones in unsupervised anomaly detection. IForest follows the ensemble paradigm [6]. Each ensemble component builds a space partitioning tree data structure, called an *isolation tree*, from a sample of the data set. The isolation tree is constructed by successively cutting the feature space via axis-parallel random cuts. Each cut isolates a portion of the data set from the rest. The procedure terminates when all elements are isolated. Then, the outlier score for each element is computed as a function of the depth (i.e. the number of cuts) of the element in the tree.

Intuitively, the number of cuts required to isolate an element is inversely proportional to its outlierness, i.e. outliers are more easy to isolate. IForest has been proven more accurate and efficient than LOF and ORCA [26, 108]. These results have also been confirmed by a few independent studies [59, 52].

**Sathe *et al.* [135], Subspace outlier detection in linear time with randomized hashing.** One of the most important milestones in the ensemble methods literature for anomaly detection is the development of linear time algorithms. One example is the random subspace hashing algorithm (RS-Hash) by Sathe and Aggarwal [135]. Traditional distance and density-based algorithms need to compute the full data set distance matrix [33, 132, 26]. This implies that distance-based methods have a quadratic complexity. RS-Hash is a linear-time ensemble algorithm for anomaly detection.

The algorithm relies on locality-sensitive hashing (LSH) [2, 48] to partition the feature space into randomly-sized buckets. Buckets with a low number of element

corresponds to low-density regions in the feature space, and can therefore be considered as outliers. The RS-Hash algorithm was been shown to outperform LOF, KNN and IForest [135]. These results have not been confirmed by independent studies.

## 3.3 Resources and tools

In this section, we present software resources and tools available to anomaly detection researchers and practitioners.

**ELKI data mining framework.** The ELKI data mining framework [138] is a software project providing and extensive library of data analysis techniques, focused on clustering and anomaly detection. ELKI is implemented in Java and it is available as open-source software[1].

ELKI provides an extensive library implementing popular unsupervised anomaly detection algorithms. Additionally, ELKI provides several APIs for algorithm development, such as efficient indexed data structures, standardized input format, etc. One disadvantage of ELKI is that it user interface does not enable running multiple experiment in parallel, and it cannot be easily embedded in other applications. ELKI implementations have been used extensively in the anomaly detection literature [97, 18, 140, 37].

**PyOD library.** A more recent resource is the PyOD outlier detection library [167]. PyOD is a library aimed at providing a large collection of implementations of outlier detection algorithms. The library is written in Python and it is available as open-source software[2].

Although it is less mature with respect to ELKI, PyOD already contains numerous outlier detection algorithms. Additionally, the success of scientific projects in the Python ecosystem [121, 127] makes it easy to extend the library by providing new algorithm implementations.

One disadvantage of PyOD is that it does not provide a user interface. Therefore, some scripting is required in order to use it to run experiments. On the other hand, PyOD can be easily embedded into other applications.

**DAMI benchmark repository.** The DAMI benchmark repository[3] is the most extensive repository for anomaly detection benchmark data sets.

The repository contains data sets generated for the most extensive study on anomaly detection carried out in the literature [37]. The repository contains both data sets and the results of the evaluation study. For each data set, several randomized versions are available. All data sets are represented in the Weka ARFF format[4].

**ODDS benchmark repository.** The Outlier Detection DataSets (ODDS) repository is another repository dedicated to outlier detection research[5]. It hosts a larger number of data sets with respect to the DAMI repository.

[1] https://elki-project.github.io/
[2] https://pyod.readthedocs.io/en/latest/
[3] https://www.dbs.ifi.lmu.de/research/outlier-evaluation/
[4] https://waikato.github.io/weka-wiki/formats_and_processing/arff_stable/
[5] http://odds.cs.stonybrook.edu/

Data sets are organized with respect to data properties (univariate, multivariate), as well as application domain. All data sets are represented in the MATLAB data format.

**Numenta anomaly detection benchmark.** The Numenta Anomaly detection Benchmark (NAB) is a benchmark repository containing time-series data[6]. The benchmark is described in Lavin and Ahmad [100] and Singh and Olinski [143]. NAB is an excellent resource for researchers and practitioners in the field of time-series anomaly detection.

## 3.4 Open challenges

In this section, we present what we consider are the most interesting open challenges in anomaly detection.

**Evaluation and benchmarking.** As already mentioned in Section 2.1.5, the evaluation of anomaly detection techniques is more challenging than for other data analysis tasks. This challenge is related to the unsupervised nature of anomaly detection.

Even though most anomaly detection techniques follow an unsupervised learning approach, benchmark data sets are required to assess the quality of a solution. Unfortunately, the evaluation methodologies and the use of these data sets are not consistent across the literature. This raises the issue of comparing results across different studies. Without replicable results, the value of research is greatly diminished.

A relatively small collection of benchmark data sets has been used in anomaly detection studies. However, it is common to apply different preprocessing across different studies, thus effectively obtaining different benchmarks [163, 146]. Comparing performance across different benchmarks is unfeasible, thus the only reliable way of assessing the relative performance of anomaly detection algorithms is through comparison studies. However, these studies require a lot of effort, and are therefore rare [102, 37, 69, 52].

We argue that the adoption of standard benchmarks and evaluation methodologies would be immensely beneficial to the anomaly detection field. We propose our contribution toward this challenge in Chapter 5.

**Scalability.** The generality of anomaly detection makes it interesting to apply anomaly detection algorithms to a broad spectrum of application domains. One hindrance to this is the poor scalability of anomaly detection algorithms. Most of the most reliable algorithms in the field were proposed almost two decades ago [33, 132]. These algorithms were designed in a world that never experienced the massive data volumes that we have to deal with today. For example, for multi-million data sets, the quadratic complexity of LOF is unfeasible. Therefore, the scalability of anomaly detection algorithms is a critical open challenge in the field.

One common approach to scalability is to distributed the computation on multiple machines, or parallel threads in a CPU. Some distributed approaches have been proposed for outlier detection [109, 122, 92, 13]. However, these approaches are either restricted to a particular problem settings, e.g. categorical features, or do not provide results for massive data sets.

[6] https://github.com/numenta/NAB

The ability to scale anomaly detection algorithms to huge data volumes its critical for their adoption in a wide spectrum of use cases. We investigate this challenges in Chapters 6 and 7.

**Data streams.** A particularly challenging anomaly detection setting is that of data streams [117]. Data streams represent continuous and theoretically infinite sources of information. In this context, an algorithm must provide results online, while analyzing only a portion of the data. Additionally, the dynamic nature of data streams influences what we consider as outliers. In fact, it is possible for an element to be correctly considered as an outlier at a given point in time, but to be incorrectly classified at a different time, a concept known as *concept drift* [155, 82]. As an example, consider a stream of temperature measurements. Depending on the geographical location, a temperature of 0°C might be unusual at noon, but completely normal at midnight. These features make streaming anomaly detection a particularly challenging task.

Although several works deal with streaming anomaly detection [10, 146, 18, 157, 135], several open challenges still exist with respect to accuracy, scalability and benchmarking [30].

# Chapter 4

# A Methodological Survey of Anomaly Detection

Anomaly detection has received a lot of research attention in the past two decades. Numerous papers and surveys on the topic have been published. However, the large variety of methodological approaches makes it difficult to characterize the current state of the art.

In this chapter, we present our methodological survey of the anomaly detection literature. This survey is developed around answering the research question: "What are the most widespread issues in the anomaly detection literature with respect to methodology, evaluation and reproducibility?".

In our survey, we quantitatively analyze the research methodology in the field. In particular, we analyze 760 anomaly detection papers along four methodological perspectives: application domain, definition of anomaly, data representation, and evaluation methodology. We characterize each surveyed paper along these axes, and we provide quantitative evidence of some of the open challenges in the field. Our results highlight that, contrary to their generality, anomaly detection techniques have been applied to only a handful of domains. Additionally, several different anomaly definitions make it difficult to compare algorithms' performance and hinder comparative research studies. Finally, inconsistent benchmark usage makes it difficult to clearly identify the current state of the art.

## 4.1 Introduction

Anomaly detection has received a lot of attention from the research community in recent years. The number of published papers on the topic has tripled in the last decade (see Figure 4.1). This published corpus describes a wide range of different approaches to the anomaly detection problem. Anomaly detection papers differ in both technical, e.g. how data is represented, and fundamental aspects, e.g. how anomalies are defined. These differences make it difficult to identify, and reason about, the current state of the art.

In order to shed some light on the issues in modern anomaly detection, in this chapter we present our extensive methodological survey of the recent literature.

A methodological survey is a quantitative survey focusing on analyzing methodological trends and patterns in a corpus. Contrary to other surveys [80, 39, 164, 75], a methodological survey does not focus on algorithmic aspects of a given paper. Instead, it categorizes and evaluates the research methodology presented therein.

Although uncommon in computer science, methodological surveys have been applied successfully to diverse fields such as medicine [44], psychology [61], and chemistry [17]. This type of survey is valuable in identifying issues in the research

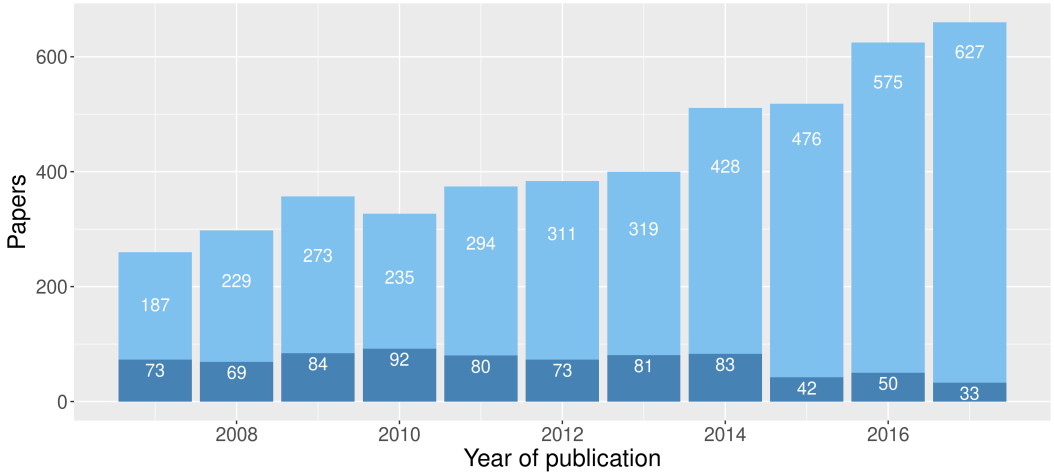


FIGURE 4.1: Published/impactful papers (upper/lower bar) on anomaly detection from 2007 to 2017 (source dblp).

methodology that might go unnoticed when analyzing only the algorithmic aspects of a proposed method. Moreover, methodological surveys also highlight possible improvement directions that can benefit the surveyed field as a whole.

In the context of our survey, we analyze 760 anomaly detection papers, published in the last decade. Papers are selected to satisfy some minimum requirements of quality and impactfulness. Data for this survey is obtained by cross-referencing the dblp[1] and Google Scholar[2] online archives. Each surveyed paper is analyzed along four methodological perspectives: application domain, anomaly definition, data representation, and evaluation methodology.

Anomaly detection can be applied to a broad range of *application domains*. Analyzing application domains makes it possible to highlight the different approaches adopted, as well as identifying gaps in the literature by considering the differences across domains.

The *anomaly definition* characterizes every anomaly detection algorithm. It is often the case that different papers propose different definitions [33, 132, 108]. One issue with this approach is that it makes it difficult to analytically compare results obtained by different algorithms. Additionally, it is not trivial to theoretically analyze algorithmic properties since different algorithms rely on different assumptions. Toward solving this issue, in this work we propose an high-level algorithm classification based on the assumptions embodied by the anomaly definition. We show that with our classification we are able to categorize the whole surveyed corpus under only six classes (see Table 4.2), thus demonstrating the usefulness of our proposal.

The *data representation* is a less fundamental aspect with respect to the anomaly definition. However, how data is represented is crucial in many application domains, since it determines which algorithms can be applied to a given use case. We provide quantitative results representing the current state of the art also with respect to data representations.

Finally, we focus on the *evaluation methodology* of anomaly detection algorithms. In particular, we stress the importance of consistent usage of benchmark data sets and performance metrics. We show how benchmarks are used rather inconsistently, thus making it difficult to identify the current state of anomaly detection research.

[1] https://dblp.uni-trier.de/
[2] https://scholar.google.com/

This chapter is structured as follows. Our survey methodology is detailed in Section 4.2. Our main discussion is presented in Section 4.3. In particular, Section 4.3.1 describes our analysis of anomaly detection applications. In Section 4.3.2, we illustrate our proposed classification of anomaly definitions. Section 4.3.3 presents the most frequent data representations used in the literature. The evaluation methodology of anomaly detection techniques is discussed in Section 4.3.4. Finally, Section 4.4 summarizes the chapter.

## 4.2 Methodology

In this section, we describe our methodology in carrying out this survey. The papers surveyed in this work were selected to be representative of prominent trends in recent anomaly detection literature. For this reason, we only considered papers published in the last decade.

We also require a minimum number of citations for each paper to filter out both niche and low-impact papers. Setting a hard threshold on the number of citations is somewhat arbitrary. However, the large number of surveyed papers makes it so that our conclusions are robust to the variation of this threshold within a reasonable range.

Papers are selected by cross-referencing the dblp and Google Scholar online archives. Data was collected in July 2018. The number of citation refers to this period. In this work, we surveyed a total of 760 papers.

Each surveyed paper must meet the following criteria, in order:

1. The paper is indexed by dblp.
2. The paper's title contains the words "anomaly detection" or "outlier detection".
3. The paper is indexed by Google Scholar.
4. The paper has at least $n$ citations, according to Google Scholar, where

$$n = \begin{cases} 20, \text{ if year} < 2016 \\ 15, \text{ if year} = 2016 \\ 10, \text{ if year} = 2017 \end{cases} \tag{4.1}$$

Each selected paper is analyzed by answering the following questions:

- What is the application domain?
- How are anomalies defined?
- How is data represented?
- How is the evaluation performed?

We select these questions since they characterize the research methodology of the paper, and they are general enough to find an answer in most surveyed papers.

Figure 4.1 depicts the total number of considered papers versus the number of surveyed papers. The full list of surveyed papers and their characterization according to the survey axes considered can be found in our online repository[3].

[3] https://github.com/passiv-me/ad-meta-survey

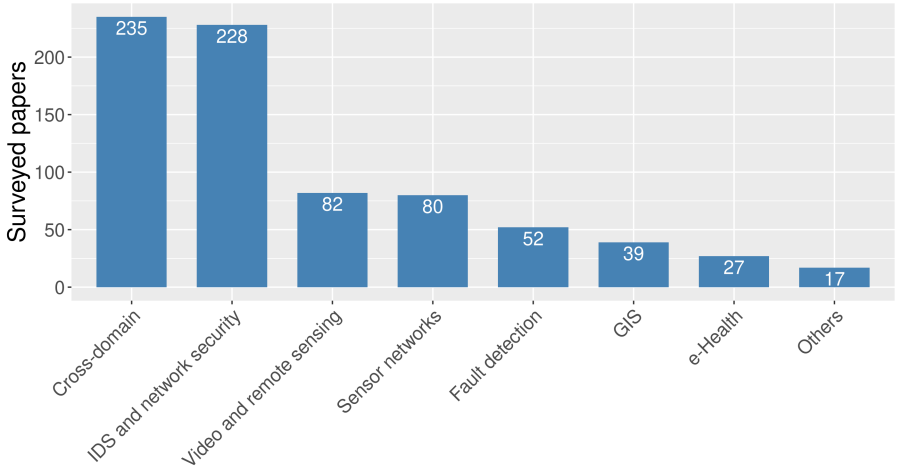


FIGURE 4.2: Surveyed papers by application domain.

## 4.3 Discussion

In this section, we present the results of our methodological survey. The discussion is organized by our four methodological perspectives.

### 4.3.1 Application domains

Anomaly detection can be applied to any data set where a semantically meaningful distinction between anomalies and normal data exists. Anomalies represent abnormal events in the phenomenon under consideration. Understanding these events might highlight actionable insights about the phenomenon. Detecting anomalies should therefore be valuable in a wide range of application domains.

In contrast to this argument, our analysis shows that anomaly detection have been investigated in a relatively small number of domains. Figure 4.2 depicts the most represented anomaly detection domains in our surveyed corpus by number of papers.

These domains are:

- **Cross-domain -** Cross-domain papers present research in anomaly detection that is not tied to a particular application domain. These papers usually propose algorithms or surveys that rely on general assumptions, e.g. relational data sets, and can be applied to several domains. Some examples in this category include the LOF [33], KNN [132] IForest [108], and Feature Bagging [101] algorithms.

- **IDSs and network security -** These papers present applications of anomaly detection to intrusion detection systems (IDSs), network monitoring, and computer security. We found this domain to be the most popular among the surveyed papers. In fact, around a third of all surveyed papers deals with intrusion detection at large.

  This can be related to several reasons. Firstly, intrusions are potentially catastrophic for an organization, thus motivating large investments and interest in their prompt detection. Secondly, intrusion detection is a very natural application of anomaly detection, and historically one of the first to be investigated [106, 63, 102]. Finally, one of the first benchmark data set developed for

anomaly detection, the KDD Cup 1999 data set [106, 148] represents an intrusion detection task. The availability of this data set certainly contributed to the development of anomaly detection techniques in the intrusion detection domain. Papers in this domain include Eskin *et al.* [63], Tylman [151] and Lin *et al.* [105].

- **Video and remote sensing -** This category includes papers dealing with video surveillance and remote sensing. Video surveillance consists in detecting abnormal events in a video stream, e.g. anomaly detection in crowded scenes [110], while remote sensing deals with detecting outliers in hyperspectral images. We consider these application domains together since they both rely on image analysis techniques. These techniques are not commonly found in other domains. Papers in this category include Mahadevan *et al.* [110], Khazai *et al.* [88], and Zhang *et al.* [166].

- **Sensor networks -** Sensor networks are distributed systems composed of several interconnected sensors. Sensor networks can be deployed in an environment in order to monitor a given phenomenon. One common assumption in the sensor network literature is that sensors are constrained in term of resources, such as battery power or network bandwidth. Sensors are also assumed to be unreliable and to have high failure rates. Moreover, sensor networks are often deployed in uncertain environments. These assumptions make it valuable to apply anomaly detection techniques to various aspects of a sensor network. These include monitoring sensors' health and resource utilization, and identifying outliers in sensor measurements. Some example papers in this domain include Rajasegarar *et al.* [131], Zhang *et al.* [165], and Egilmez and Ortega [57].

- **Fault detection -** Detecting failures or misbehaving equipment is another natural use case for anomaly detection techniques. Early detection of outliers in this domain is particularly valuable since preventing a failure might be order of magnitude less costly than repairing a broken equipment. Fault detection techniques have been applied the a broad spectrum of use cases, such as industrial equipment [16], aeronautical vehicles [142], and cloud environment monitoring [152].

- **GIS -** Geographic Information Systems (GISs) are information systems for managing spatial data. Anomalies in this domain have the advantage of being more easily explainable and comprehensible. Additionally, in this context anomalies can arise both in term of spatial and temporal data features. These types of anomalies are known as *spatial* and *temporal* outliers, respectively. Example applications in this category include maritime surveillance [93], urban traffic monitoring [103], and mobile phone data sensing [45].

- **e-Health -** Anomaly detection techniques have been investigated also in the medical domain. Example use cases include clinical patient monitoring [77] and assisted diagnosis systems [150]. It has been shown that anomaly detection algorithms can be used to discriminate between normal and abnormal medical conditions, thus assisting medical professionals in the diagnosis process. In particular, surveyed applications include automatically detecting seizures [158], eye diseases [136], and cancer [150].

TABLE 4.1: Distribution of surveyed papers by application domains.

| Application domain | Surveyed papers (%) |
|---|---|
| Cross-domain | 30.9 |
| IDS and network security | 30.0 |
| Video and remote sensing | 10.8 |
| Sensor networks | 10.5 |
| Fault detection | 6.8 |
| GIS | 5.1 |
| e-Health | 3.6 |
| Others | 2.3 |

- **Others -** Less than three percent of surveyed papers deals with application domains not included in the previous categories. Some examples include optimization [156] and business process mining [29].

For convenience, the distribution of surveyed papers by application domain is presented in Table 4.1. As already mentioned, intrusion detection systems are the most represented domain. Video and remote sensing and sensor networks respectively represent around ten percent of the surveyed corpus. Other application domains are discussed in less than seven percent of papers.

### 4.3.2 Anomaly definitions

As introduced in Section 2.1.4, each anomaly detection algorithm is characterized by a particular anomaly definition. An anomaly definition identifies which data elements are to be considered anomalous by the algorithm. Since the property of being anomalous cannot be defined in absolute terms, an anomaly definition usually depends on user-defined hyperparameters [33, 108, 129].

A large number of anomaly definitions have been proposed [39]. This can be related to the large number of cluster definition proposed in the clustering literature [65]. Namely, the definitions of both anomalies and clusters rely on human intuition and there is no one-size-fit-all definition that encompasses all aspects of the problem.

The relation between different anomaly definitions has been partially investigated in Knorr and Ng [89], Yamanishi and Takeuchi [160], and Schubert *et al.* [141]. However, which definition is better suited for a given anomaly detection task still remains an open question.

TABLE 4.2: Distribution of surveyed papers by anomaly definition class.

| Anomaly definition | Surveyed papers (%) |
|---|---|
| Likelihood anomaly | 33.3 |
| Model-based anomaly | 28.6 |
| Similarity-based anomaly | 17.5 |
| Cluster-based anomaly | 11.2 |
| Frequency-based anomaly | 7.0 |
| Rule-based anomaly | 2.4 |

In this work, we propose an high-level classification of anomaly definitions, and consequently algorithms, into six classes. These classes are based on the theoretical assumptions that an algorithm relies on. Noticeably, most anomaly detection algorithms rely an a limited set of assumptions. In fact, all 760 surveyed papers could be classified using only six classes.

Our classification finds its value, for example, in comparison studies, where it might make little sense to compare algorithms based on profoundly different assumptions. Our proposed classification is summarized in Table 4.2.

In particular, we propose the following classes:

- **Likelihood-based anomalies -** Algorithms in this class consider a data element as anomalous if its probability, or likelihood, according to a given generative stochastic process, is below a given threshold. This class relies on the assumption that the data distribution is either known (parametric approach) or can be estimated (non-parametric approach). Some examples of likelihood-based algorithms include Bayesian Networks [151], GMMs [161], and Information Entropy models [46].

- **Model-based anomalies -** Algorithms in this class consider as outliers those data elements whose reconstruction error, according to a machine learning model, is above a given threshold. The assumption in this class is that a training data set is available. Some machine learning models also require labeled data to be available. Some examples of these models include SVMs [60, 130], and wavelet decomposition [71]. Neural networks have also been investigated for outlier detection [79].

- **Similarity-based anomalies -** These algorithms consider a data element anomalous if its distance, or a function of its distance, to other data elements exceeds a given threshold. The underlying assumption is that a similarity metric can be defined between data elements, and that such a metric makes sense in the considered feature space [28].

  This class of algorithms is one of the most popular for unsupervised outlier detection due to its explainability and good baseline performance [37]. However, algorithms based on distance computations are known to suffer from the curse of dimensionality in high-dimensional data sets [28]. This class encompasses both distance-based approaches [90, 132] and density-based approaches [33].

- **Cluster-based anomalies -** This class of algorithms considers as anomalies those data elements that do not belong to any of the prominent clusters in the data. Algorithms in this class assume that a cluster definition is available. These mainly differ with respect to the particular clustering approach adopted. The most popular approaches are density-based clustering [134], $k$-means clustering [58], and fuzzy clustering [107].

- **Frequency-based anomalies -** For algorithms in this class a data element is anomalous if the values of its attributes have low frequency with respect to the majority of the data. This class of algorithms assumes that data contains discrete attributes which can only take on a limited number of values. Some examples include Bezerra *et al.* [29] for business process logs, and Koufakou *et al.* [92] for categorical data sets.

- **Rule-based anomalies -** Algorithms in this class consider a data element anomalous if it does not match a set of rules set by a domain expert. These algorithms

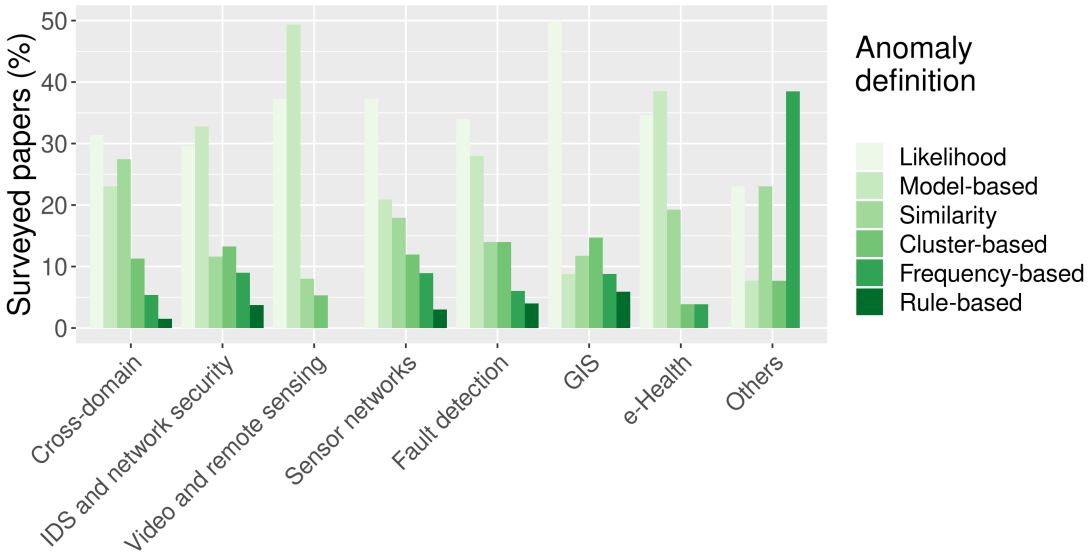


FIGURE 4.3: Distribution of anomaly definitions by application domain

assume that domain knowledge is available in order to define the rules. Examples in this class include Narita and Kitagawa [118] and Duffield *et al.* [55].

The distribution of surveyed papers with respect to anomaly definition classes is presented in Table 4.2. The most represented classes are the likelihood and model-based anomalies, amounting to a total of 61.9% of the surveyed papers. Similarity-based approaches, although very popular in unsupervised outlier detection [37], amount only for 17.5% of the surveyed papers. Frequency and rule-based approaches have been analyzed in less than 10% of the surveyed papers. Notice that these approaches are generally less articulated and more easy to implement with respect to other approaches.

To complement our analysis, we also present the anomaly definition distribution with respect to the application domains introduced in Section 4.3.1. Figure 4.3 illustrates our results. Most noticeably, likelihood-based approaches have been investigated consistently across all application domains considered, with frequencies varying between 20% and 50%. Model-based approaches are also fairly popular, in particular in the remote sensing domain (almost 50% of surveyed papers). However, they are particularly underrepresented in the GIS domain. Interestingly, similarity-based algorithms have been investigated successfully in cross-domains papers (around 27% of surveyed papers), but not as much in domain-specific papers.

Finally, notice that in the less investigated application domains, i.e. "Others", the prevalent approach is frequency-based. This might be due to the ease of implementation and low resource requirements of frequency-based algorithms, which makes them appropriate to a large variety of use cases.

### 4.3.3 Data representations

Data can be represented in a variety of formats. Popular data representations include tabular data, time series [83] and graph data [56]. Most anomaly detection

TABLE 4.3: Distribution of data representations in surveyed papers.

| Data representation | Surveyed papers (%) |
|---|---|
| Real-valued vectors | 63.5 |
| Discrete vectors | 18.9 |
| Videos | 5.9 |
| Hyperspectral images | 5 |
| Graphs | 3.9 |
| Mixed-attribute vectors | 2.8 |

algorithms assume data is represented in a particular format. Thus, data representations affect which techniques can be applied to a particular use case. In this section, we analyze our surveyed corpus with respect to data representations.

Table 4.3 presents the distribution of surveyed papers by data representation. The most common data representation in anomaly detection is real-valued feature vectors, i.e. real-valued data matrices. This is to be expected, since real-valued data sets are one of the most common representations in data analysis and machine learning applications [115].

The second most common data representation is discrete vectors. This representation is used for data attributes that can only take on a limited set of values. Some examples include application logs or network packet traces. Other data representations in our surveyed corpus include multimedia formats, such as video or images, and graph data. The least popular data representation consists of mixed-attribute data sets, i.e. data sets containing a mixture of continuous and discrete attributes. Noticeably, this representation, which is the most general, is also the least investigated in our surveyed corpus.

### 4.3.4 Evaluation methodology

As mentioned in Section 2.1.5, the evaluation of anomaly detection techniques presents several challenges.

The most common approach for evaluating an anomaly detection algorithm consists in executing the algorithm on a benchmark data set [37]. A benchmark data set, or *benchmark*, contains for each data element a label stating whether the element is anomalous or normal. This information is known as the *ground truth* of the data set. By comparing the results of the algorithm with the ground truth we can assess its performance. This comparison is carried out by using binary classification metrics such as *precision*, *recall* and the area under the ROC curve (ROC AUC) [49, 66].

One issue with this approach is that it relies on benchmark data sets. Realistic benchmarks are seldom available in anomaly detection. Anomalies are rare by definition, and labeling large data sets requires a lot of time and effort. Therefore, anomaly detection research and practitioners must settle for surrogates of realistic benchmarks.

One of the first benchmark developed for anomaly detection is the KDD Cup 1999 intrusion detection data set [106]. This benchmark contains data on cybersecurity attacks in a computer network, as well as "clean" network data. This benchmark is a surrogate data set, since it was collected in a simulated environment. This diminishes the reliability of the benchmark, as well as introducing other issues highlighted in different studies [148, 147].

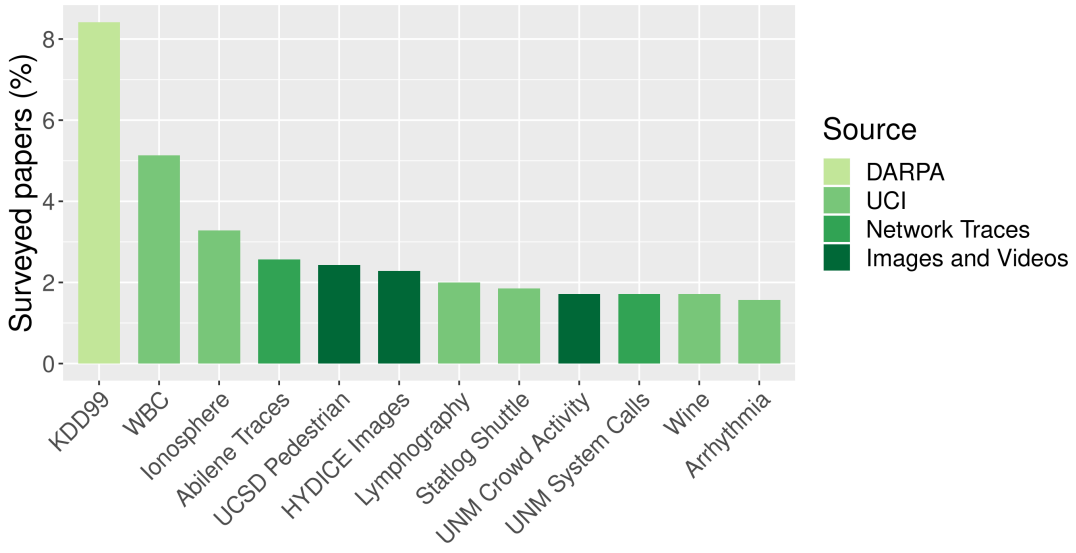


FIGURE 4.4: Distribution of benchmark data sets.

A popular approach for producing benchmark data sets for anomaly detection is to manipulate binary classification benchmarks to introduce characteristics of anomaly detection problem instances, e.g. class imbalance [59, 37]. This approach produces better quality benchmarks with respect to adopting binary classification benchmarks without modifications [37].

In the context of our analysis, Figure 4.4 depicts the most commonly used benchmarks in our surveyed corpus. The benchmark source is also presented.

Our results show that the KDD Cup 1999 data set is clearly the most frequently used benchmark. However, it is important to notice that this benchmark was used in less than ten percent of surveyed papers. Considering that, as discussed in Section 4.3.1, intrusion detection papers comprise around thirty percent of the surveyed corpus (see Table 4.1), this implies that the most common benchmark is not widely used even within the community for which it was designed. Less popular benchmarks are used even less consistently.

This highlights an important issue with data set usage in anomaly detection. Namely, data sets for comparative research are either not publicly available [153, 81, 52] or not used consistently, e.g. using the same data set but considering different anomalous classes. Modifying a benchmark, effectively obtaining a different data set, generates unnecessary confusion and hinders the replicability of results.

Notice that the majority of surveyed benchmarks comes from the UCI machine learning repository [104]. This repository contains binary classification data sets which are not fit for testing anomaly detection algorithms. Using binary classification data sets to assess the performance of anomaly detection algorithms might lead to inconsistent results [37]. Luckily, several anomaly detection benchmarks and repository have been recently proposed [59, 37, 133]. These should be preferred over more generic benchmarks.

Finally, we note that most surveyed anomaly detection papers focus on the ROC AUC metric as the only performance metric. Although this approach improves comparability of results across papers, it also has some drawbacks. For instance, it is known that the ROC AUC metric tend to overestimate the quality of an algorithm with respect to more conservative metrics, e.g. the average precision score [49]. We

advocate for a multi-metric approach, since considering different metrics has the power to highlight different aspects of a given solution.

## 4.4 Summary

In this chapter, we presented our quantitative methodological survey of anomaly detection. Methodological surveys have been proven valuable in a broad spectrum of fields including medicine and chemistry. In particular, methodological surveys highlight open challenges in the surveyed field.

In our survey, we analyzed anomaly detection papers along four methodological aspects, namely: application domain, definition of anomaly, data representation, and evaluation methodology. We considered for review a large number of anomaly detection papers published in the last decade and surveyed a total of 760 papers.

With respect to application domains, we showed that anomaly detection has been investigated in a limited number of domain. This is in contrast with the generality of anomaly detection techniques. The most represented anomaly detection domain is intrusion detection. Other popular domains include sensor networks and remote sensing. We showed how the popularity of a given domain is related to the availability of benchmark data set for that domain.

Then, we analyzed surveyed papers by anomaly definitions. We demonstrated that a large number of anomaly definitions have been proposed. This variety makes it difficult to compare different techniques and clearly establish the state of the art. To tackle this problem, we proposed an high-level classification of surveyed papers based on six anomaly definition classes. We showed how our classification can be used to organize the literature into easily comparable subfields.

We then analyzed the surveyed corpus with respect to both anomaly definitions and application domains, and we presented our discussion on how different classes are more popular in different domains. Similarly, we analyzed the surveyed corpus with respect to data representations.

Finally, we discussed the most popular evaluation methodologies in the field and highlighted several issues. In particular, we demonstrated how benchmark data sets are not used consistently, thus making it difficult both to replicate published results and to conduct comparative research studies.

# Part II

# Benchmarking

# Chapter 5

# BAD: Benchmarking for Anomaly Detection

Most anomaly detection algorithms depend on one to several hyperparameters to be set by the user. Finding the optimal settings for these hyperparameters requires testing several configurations for each data set under analysis. This approach is both time and effort-consuming, and it is often unfeasible. For this reason, users usually choose hyperparameter settings by following rules of thumb, which leads to suboptimal performance.

In this chapter, we present our framework for Benchmarking Anomaly Detection (BAD). BAD aims at answering the following research question: "How can we improve the reproducibility of anomaly detection evaluation experiments across different studies?". BAD reduces the burden of hyperparameter tuning by testing several hyperparameter configurations in parallel on a cluster of machines. We demonstrate BAD's usefulness and effectiveness by analyzing four state-of-the-art anomaly detection techniques on eight benchmark data sets. Our results show that rule-of-thumb settings are unreliable, and that hyperparameter tuning leads to significant performance gains on the majority of the problem instances considered. Additionally, we demonstrate how different hyperparameter tuning strategies lead to different outcomes when trying to reproduce comparative research studies. Finally, we highlight the effectiveness and scalability of BAD with respect to other anomaly detection frameworks.

An extended version of this chapter has been accepted for publication in the Big Data Research journal.

## 5.1 Introduction

A large collection of anomaly detection techniques has been developed over the years [39, 3]. Most of these techniques require the user to set one to several hyperparameters. An *hyperparameter* is a setting that characterizes the execution of an anomaly detection algorithm. Each hyperparameter is defined on a given domain, e.g. the set of positive integers [108], or a probability between zero and one [137].

The notion of an hyperparameter is not unique to anomaly detection. For example, the number of clusters in the famous $k$-means clustering algorithm [84] is an example of an hyperparameter. In the context of anomaly detection, an example of hyperparameter is the number of neighbors in the Local Outlier Factor (LOF) algorithm [33], defined on the set of positive integers smaller then the data set size.

Hyperparameter settings determine the performance of an anomaly detection algorithm on a particular data set. The performance of an anomaly detection algorithm can be measured via classification metrics, such as the average precision score

(AP) or the area under the ROC curve (ROC AUC) [49]. These metrics provide a quantifiable measure of the quality of a solution provided by the algorithm to the anomaly detection task.

Finding the *optimal* hyperparameter settings with respect to a performance metric is not trivial. The number of hyperparameter configurations to test grows combinatorially with the number of hyperparameters. Additionally, even a single hyperparameter might lead to a large number of possible configurations, e.g. in $k$-means the $k$ hyperparameter is defined over the set of all positive integers. In these cases, i.e when the hyperparameter domain is unbounded, proving the optimality of a given configuration might be impossible.

The need to test all possible hyperparameter configurations stems from the fact that the dependency between the hyperparameter settings and the performance metric is not known. In general, this relation is arbitrary and it depends on the data set under consideration. Thus, when searching for the optimal hyperparameter settings, we cannot assume the monotonicity, or continuity, of the metric with respect to the hyperparameters. This make it impossible to prune the search space [15]. For the same reasons, if the hyperparameter domain is unbounded, we cannot prove that a given configuration is optimal, since we cannot test all possible configurations.

These issues make hyperparameter tuning a difficult and time-consuming task. In the case where finding the optimal hyperparameter configuration is unfeasible, hyperparameter tuning deals with finding an "appropriate" configuration following heuristic procedures. The most common heuristic procedures are rules of thumb and grid searches in the hyperparameter space.

Finding an acceptable hyperparameter configuration for a given anomaly detection task is critical to both practitioners and researchers. Practitioners are interested in detecting anomalies as accurately as possible, since detection errors might lead to economical losses. On the other hand, researchers are interested in pushing the state-of-the-art by developing competitive and novel techniques. The competitiveness of a technique is established through experimental comparison with the state-of-the-art. However, without proper hyperparameter tuning, the comparison might result biased toward a particular algorithm [95].

As our contribution toward overcoming these issues, in this chapter we present our framework for Benchmarking Anomaly Detection (BAD). BAD is a distributed framework for benchmarking and hyperparameter tuning of unsupervised anomaly detection techniques. BAD leverages distributed computation on commodity hardware to enable parallel execution of a large number of experiments. This let users perform massive hyperparameter grid searches in a timely manner. BAD includes a library of anomaly detection algorithms to foster comparison studies. At the time of this writing, BAD library features seven state-of-the-art anomaly detection techniques. Additionally, this library can be easily extended with novel techniques. BAD also includes a collection of benchmark data sets from the literature that can be used to compare algorithms in a reproducible and shareable manner. Finally, BAD features a simple web-based graphical user interface to monitor experiments execution and to visually analyze the results.

In this chapter, we introduce the hyperparameter tuning problem in Section 5.1.1. We describe the BAD framework in details in Section 5.2. In Section 5.3.2, we make use of BAD to analyze several issues with rules of thumb hyperparameter settings. We compare rules of thumb with grid-search hyperparameter tuning in Section 5.3.3. Then, in Section 5.3.4, we demonstrate how hyperparameter tuning affects comparison studies. Finally, in Section 5.3.5, we compare the scalability of BAD with respect

to the ELKI framework [138], a popular choice among researchers to implement anomaly detection algorithms, on an hyperparameter tuning task. Section 5.4 concludes the chapter.

### 5.1.1 Hyperparameter tuning

In the remainder of this chapter, we focus on the hyperparameter tuning problem. This is related to Problem 1 defined in Chapter 1. We extend our notation to account for the fact that an anomaly detection algorithm might depend on user-defined hyperparameters. We represent a scoring function as $s : \mathbb{R}^{n\times d} \times \Theta \to \mathbb{R}^{n\times(d+1)}$, where $\Theta$ is the hyperparameter domain. For example, if $s$ depends on a positive integer parameter, then $\Theta \equiv \mathbb{N}^+$. If $s$ depends on a positive integer and a real-valued hyperparameter then $\Theta \equiv \mathbb{N}^+ \times \mathbb{R}$. We can now define the hyperparameter tuning problem as follows:

**Problem 2 (Hyperparameter Tuning)** *Given a benchmark data set $D \in \mathbb{R}^{n\times d}$ with ground truth $g : \mathbb{R}^{n\times d} \to \{0.0, 1.0\}^n$, a metric $m : \{0.0, 1.0\}^n \times \mathbb{R}^{n\times(d+1)} \to \mathbb{R}$, and a scoring function $s$ from the family $\mathcal{F} = \{f \mid f : \mathbb{R}^{n\times d} \times \Theta \to \mathbb{R}^{n\times(d+1)}\}$, find an hyperparameter configuration $\theta^* \in \Theta$ such that*

$$\theta^* = \arg\max_{\theta\in\Theta} \{m(g(D), s(D, \theta))\} \tag{5.1}$$

Without any assumption on the shape of $m(g(D), s(D, \theta))$ with respect to $\theta$, finding $\theta^*$ requires considering all hyperparameter configurations $\theta \in \Theta$. When $\Theta$ is unbounded this is unfeasible. A common approach in hyperparameter tuning is to find an appropriate hyperparameter configuration for scoring function $s$ on dataset $D$. An appropriate configuration can be defined as the optimal configuration in a heuristically defined subset of the hyperparameter domain $\Theta$.

The two most common approaches to hyperparameter tuning in anomaly detection are to apply rules of thumb, or to perform a grid search. Applying a rule of thumb (RoT) for an algorithm consists in reusing hyperparameter settings proposed in the original paper describing the algorithm.

For example, the original paper for LOF suggests to set the number of neighbors in the range $[10, 50]$ independently from the data set [33]. These settings have been used in several studies throughout the literature [124, 108, 96]. This approach requires the least amount of effort, since it does not rely on testing a large number of hyperparameter configurations. On the other hand, this approach does not adapt itself to the data set under consideration, which might lead to unreliable results when considering a large variety of data sets.

A grid search (GS) consists in testing hyperparameter settings at regular intervals in the hyperparameter space $\Theta$. A grid search requires testing all configurations in the grid. With this approach, it is possible to trade-off the time spent in hyperparameter tuning with the likelihood of finding an appropriate hyperparameter configuration. This feature is desirable in resource constrained environments, e.g. in industrial applications, where the cost of hyperparameter tuning might be an issue [22]. The more configurations are tested during the grid search, i.e. the larger the grid, the more likely we are to find a good configuration. Although a grid search requires more effort, this approach is more robust than the rule-of-thumb approach, as we will demonstrate in Section 5.3.2. This is because a grid search exploits the information in the data set under analysis. In fact, with a grid search we can analyze the most promising hyperparameter configurations, and we can iteratively refine the search to move toward promising directions.

## 5.2 The BAD framework

In this section, we present the Benchmarking Anomaly Detection (BAD) framework. The BAD framework enables running extensive hyperparameter grid searches for anomaly detection techniques. The BAD framework is written in Python. All source code and documentation of BAD can be found in the BAD repository[1].

In the following sections, we describe the implementation details of BAD with respect to design goals (Section 5.2.1), architecture (Section 5.2.2), algorithms (Section 5.2.3), data sets (Section 5.2.4), hyperparameters (Section 5.2.5), and performance metrics (Section 5.2.6).

### 5.2.1 Requirements and design goals

Effective hyperparameter tuning requires a large number experiments to be executed. This determines the number of hyperparameter configuration in $\Theta$ that can be tested during a grid search. The more configurations we test the more likely we are to find a "good-enough" hyperparameter configuration. Following these considerations, a good metric for a benchmarking framework is the *experiment throughput*, i.e. the number of experiments executed per unit of time. Naturally, the experiment throughput depends on the type of experiments executed (e.g. cardinality of the data set, anomaly detection algorithm, etc.) since different experiments amount to different execution times.

If we want to maximize the experiment throughput and without assuming any prior information on the type of experiments, the only viable approach is to implement parallel experiment execution. BAD implements this paradigm by distributing the experiments on a cluster of machines. This provides high experiment throughput, and it also has the desirable property of introducing a trade-off between cost, i.e. amount of computational resources used or execution time, and quality of the hyperparameter tuning results.

BAD was also designed to produce easily replicable and shareable results. This is important in the research community to produce high-quality research. In order for results to be replicable, each experiment must be completely specified, and specifications must be easy to understand, edit and share. An anomaly detection experiment is completely specified by selecting an algorithm, a data set and an hyperparameter configuration. BAD uses plain-text files to specify each of these aspects, thus making specifications easy to share (e.g. see Section 5.2.5).

### 5.2.2 Architecture

The BAD framework consists of several processes that can be either instantiated on a single host or distributed on multiple hosts. BAD uses a client-server architecture. The main components of BAD are:

- **BAD client -** the BAD client provides the main user interface for the BAD framework. It is responsible for parsing user input (configuration files and options) and sending the required information to the server-side processes, as well as downloading the results once execution has completed.
- **BAD master -** the BAD master is the main server-side process. The master is responsible for scheduling experiments and collecting results. The BAD master is also the only communication point between the server-side processes and

[1] https://github.com/passiv-me/bad-framework

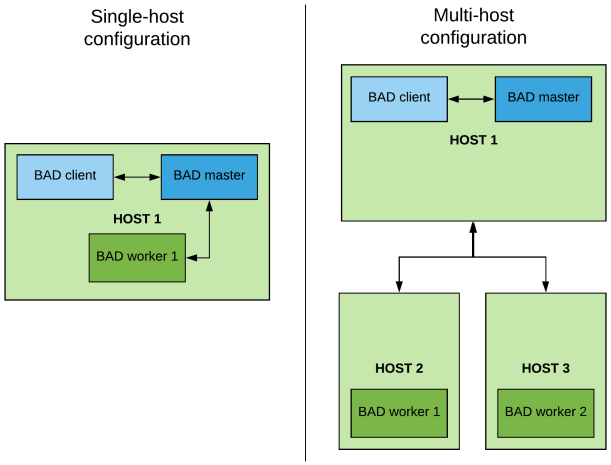


FIGURE 5.1: Example deployments of the BAD framework. BAD can be deployed either on a single machine (left) or on a cluster (right).

the BAD client. Finally, the BAD master exposes the Web GUI that the user can use to monitor the execution. Only one master can be active at any given time.

- **BAD worker -** the BAD worker executes experiments within the BAD framework. Several workers can be instantiated within the framework. Each worker is responsible for running the experiments that the master has assigned to it. Workers are also responsible for sending the results of each completed experiment to the master.

Both master and workers are implemented as Web applications. There are no constraints on the BAD client as long as it implements the required communication protocol with the BAD master. All inter-process communication within BAD relies on the HTTP protocol.

Figure 5.1 depicts two possible deployments of the BAD framework. On the left, we have a single-host deployments. In this case, all processes are instantiated on the same machine. On the right, an example of multi-host configuration is shown. In this case, client and master reside on the same host while workers reside on different hosts.

### 5.2.3 Candidates

A *Candidate* is the main BAD abstraction for an anomaly detection algorithm. The name "candidate" refers to the benchmarking purpose of BAD, i.e. to find the best candidate algorithm for a given problem.

A Candidate must implement a scoring function, as defined in Problem 1. This restricts a Candidate to implement scoring functions for numerical data sets only. We argue that this assumption is not too restrictive since most unsupervised anomaly detection research deals with numerical data sets, and there are ways to convert categorical features into numerical ones [37, 68].

In practice, a Candidate is a Python module implementing an anomaly detection scoring function according to a specific interface. The scoring function takes as input a data element and produces and outlier score as output. Each candidate can also be fitted on a particular data set using the fit function. This interface is general enough to be easily implemented for most unsupervised anomaly detection algorithms.

TABLE 5.1: BAD Candidate library.

| Candidate | Source | Reference |
|---|---|---|
| LOF | scikit-learn | [33] |
| KNN | scikit-learn | [132] |
| Weighted KNN | scikit-learn | [11] |
| OCSVM | PyOD | [137] |
| LOCI | PyOD | [124] |
| FeatureBagging | PyOD | [101] |
| IForest | scikit-learn | [108] |

BAD includes a library of Candidate implementations. Each Candidate in this library implements one state-of-the-art anomaly detection technique. At the time of this writing the Candidate library contains the algorithms represented in Table 5.1. For a detailed description of the Candidates used in this chapter, see Section 5.3.1.

The BAD Candidate library makes it easy to compare different anomaly detection techniques. Most implementations comes directly from popular data analysis frameworks, such as scikit-learn [127] and PyOD [167]. Additionally, the BAD Candidate library can be easily extended with novel algorithms. The choice of Python as programming language makes it easy to implement new anomaly detection algorithms by leveraging the large collection of APIs found in the scientific Python ecosystem [121, 127].

### 5.2.4 Data sets

As already described in Section 5.1.1, in order to measure an anomaly detection performance metric a benchmark data set is required. According to the principle of domain-specific benchmarks [72], a benchmark data set should represent a real-world problem instance. This is important since the purpose for anomaly detection algorithms is to be applied to real-world anomaly detection tasks. When using a benchmark data set we assume that the performance on the algorithm on the benchmark is an approximation of the performance of the algorithm on any anomaly detection task. If the benchmark does not represent a realistic anomaly detection task this assumption does not hold.

Realistic benchmark data sets for anomaly detection are rare. One way to overcome this issue is to adapt binary classification data sets to anomaly detection [59, 37]. The adaptation process is required in order to produce realistic data sets. For example, one common assumption in anomaly detection is that the anomaly class is much less frequent than the normal class, i.e. classes are unbalanced. This assumption does not hold for many classification data sets. In these cases, the anomaly class should be subsampled to enforce class imbalance. In some cases, other forms of preprocessing are also necessary [37]. Several repositories with adapted anomaly detection data sets exist [37, 133].

Another issue with respect to anomaly detection benchmarking is that data sets are not used consistently across different studies. One example of this is when the same data set is used in different studies, but it is preprocessed differently, effectively generating different problem instances [42, 163]. This hinders the replicability of results, and it creates confusion.

The BAD framework solves these problems by decoupling algorithm design and evaluation. BAD includes a collection of benchmark data sets from the literature.

TABLE 5.2: BAD collection of benchmark data sets.

| Data set | Data elements | Features | Outliers (%) | Source |
|---|---|---|---|---|
| annthyroid | 7129 | 21 | 7.49 | DAMI [37] |
| covertype | 286048 | 10 | 0.9 | ODDS [133] |
| glass | 213 | 9 | 4.22 | ODDS [133] |
| kdd99 | 48133 | 40 | 0.42 | DAMI [37] |
| pendigits | 6870 | 16 | 2.27 | DAMI [37] |
| shuttle | 1013 | 9 | 1.28 | DAMI [37] |
| wbc | 377 | 30 | 5.3 | ODDS [133] |
| wine | 129 | 13 | 7.7 | ODDS [133] |

These data sets have been used extensively for evaluating anomaly detection algorithms and they have been shown to represent realistic problem instances [37, 133].

The adoption of a static collection of data sets has several advantages: i) performance metrics for different algorithms can be easily compared and shared, ii) algorithm comparison cannot be biased by hand-picking particular data sets, iii) all data sets are ready to use, without the need for preprocessing or format conversion.

All BAD data sets are represented using the ARFF format[2]. The ARFF format has the advantage to be both humanly-readable and easily parseable. Additionally, ARFF data can be easily shared since each data set is represented by a single text file.

The BAD data set collection is listed in Table 5.2. To improve diversity, the data sets are selected from a wide range of applications domains and have varying characteristics. The number of data elements ranges from a few hundreds to several thousands. The outlier frequency ranges between approximately 1% to 8%.

Although a static collection of benchmark data sets has several advantages for benchmarking, we designed the BAD data set collection to be extensible so that other use cases might also be included.

### 5.2.5 Hyperparameter specification

The BAD framework is designed to run anomaly detection experiments. Each experiments is defined by a data set, a Candidate, and the Candidate's hyperparameter configuration.

As with data sets and Candidates, hyperparameter configurations are specified using plain-text files. This ensures simple usage and easy sharing of configurations. BAD uses the same interface for hyperparameter settings and grid searches. For each hyperparameter, either a value or a range of values can be specified in the hyperparameter file. An example hyperparameter specification is:

```
# Hyperparameter setting example: <name>  <value>
x  10

# Grid search example: <name>  <start>  <end>  <step>
y  1  10  1
```

If a single value is specified, a single experiment is run. If, on the other hand, a range of values is specified, an experiment for each value in the range is run, thus performing a grid search. Through this interface, the user can specify both single experiment runs, as well as multiple experiment runs, i.e. *experiment suites*. BAD

[2]https://waikato.github.io/weka-wiki/formats_and_processing/arff_stable/

supports integer, real-valued and string hyperparameters. Grid searches are not defined for string hyperparameters.

### 5.2.6 Performance metrics

Performance metrics provide a quantifiable measure of quality for a solution. The most commonly used metrics for the effectiveness of anomaly detection techniques are the area under the ROC curve (ROC AUC), the precision/recall, and other derived metrics [76, 49]. For each completed experiment, BAD reports the ROC AUC and the average precision score (AP).

The ROC AUC is defined as the area under the ROC curve [76]. The ROC curve is the curve in the true positive rate/false positive rate plane composed by varying the decision threshold of a classifier. ROC AUC can range in $[0.0, 1.0]$, where 1.0 represents a perfect classifier. The theoretical ROC AUC of a random classifier is 0.5. ROC AUC has the advantage of being independent on the magnitude of the outlier scores, since it depends only on their relative ordering. A perfect ROC AUC implies that all outliers have higher scores with respect to any normal data elements. One disadvantage of the ROC AUC is that the interpolation used to compute the area can overestimate the performance of the algorithm.

AP is defined as:

$$AP = \sum_n (R_n - R_{n-1}) P_n$$

where $n$ varies within the classifier decision thresholds, and $P_n$ and $R_n$ are the precision and recall at threshold $n$, respectively. This formulation does not interpolate the area under a curve. Thus, it is more robust to overestimation.

Most anomaly detection studies focus on ROC AUC as the only effectiveness metric. In the design of BAD, we followed a multi-metric approach. Using multiple metrics is advisable since even though performance metrics might be correlated, considering multiple metrics usually exposes useful information [49]. We provide evidence of this in the following section.

## 5.3 Experimental evaluation

In this section, we leverage the power of the BAD framework to answer the following questions:

- Is the rule-of-thumb approach reliable for hyperparameter tuning of anomaly detection techniques?
- How does the rule-of-thumb approach compares with grid searches?
- Is the performance gain obtained via grid searches significant with respect to rule-of-thumb settings?
- Does using different performance metrics and hyperparameter tuning approaches affect the relative rankings of anomaly detection algorithms?
- How does BAD compare with other anomaly detection frameworks in term of experiment throughput?

To answer these questions we present a comparative evaluation of some of the algorithms in the BAD Candidate library on the BAD data set collection.

All reported experiments are executed on Ubuntu Linux 18.04 LTS machines powered by an Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz and 8GB of RAM.

### 5.3.1 Candidates

In this section, we introduce the Candidates used in our experiments. All Candidate implementations can be found in the BAD Candidate library.

**LOF** The Local Outlier Factor algorithm (LOF) [33] is one of the most popular algorithms for unsupervised anomaly detection. LOF is classified as a local density-based technique. LOF's scoring function assigns a score to each data element depending on the relative $k$-nearest neighborhood density between the data element and its neighbors. This approach is known to work well in data sets with varying degrees of density, where a global approach cannot discriminate between low-density and high-density regions [33]. The only hyperparameter of LOF is the number of neighbors $k$ to consider in the computation.

**KNN** The KNN algorithm (KNN) [132] is another popular algorithm in anomaly detection. Similarly to LOF, KNN also uses distances between data elements to compute the scoring function. However, contrary to LOF, the score for each element is the Euclidean distance to its $k$-th nearest neighbor. KNN can be classified as a global distance-based approach [141]. Similarly to LOF, the only hyperparameter of KNN is the neighborhood size $k$.

**IForest** The Isolation Forest algorithm (IForest) [108] is an ensemble method for unsupervised outlier detection. Instead of relying on distance computations it relies on the concept of isolation. Intuitively, anomalous elements are the ones that can be more easily isolated from the rest of the data set. For each data element, IForest computes the number of random cuts (i.e. hyperplanes in the feature space) required to isolate it from other data elements. This procedure is repeated multiple times by subsampling the data set and generating a forest of isolation trees. The average number of cuts averaged within the forest is used to compute the outlier score for the element.

The hyperparameters for IForest are the number of trees, the size of the sample used to build each tree, and the random number generation seed which determines the sampling process.

**OCSVM** The one-class Support Vector Machine (OCSVM) [137] is an optimization technique for binary classification that can be adapted to anomaly detection. This technique relies on solving the problem of estimating a region in space that contains the majority of the data set, i.e. where the data distribution is positive. This region is defined by its boundary data elements, known as support vectors (SV). A classification is obtained by evaluating the position of each data element with respect to the computed region. Non-linearly separable classes can be mapped onto a linearly separable space using a kernel function.

The hyperparameter of OCSVM are the "softness" of the boundary, the $\epsilon$-precision of the solver, and the parameter specifying the kernel function.

### 5.3.2 Rule-of-thumb settings vs. grid searches

In this section, we investigate the performance of rule-of-thumb settings with respect to grid searches for hyperparameter tuning. Applying a rule of thumb corresponds to selecting an hyperparameter value based on previous studies, i.e. without experimental validation. For example, a common choice for the LOF's hyperparameter $k$

| Algorithm | POS | NEG | NUL |
|---|---|---|---|
| LOF | 25% | 75% | 0% |
| KNN | 25% | 50% | 25% |
| OCSVM | 0% | 62.5% | 37.5% |
| Total | 16.66% | 62.5% | 20.83 % |

TABLE 5.3: Aggregated results for rule-of-thumb experiments. POS refers to the case in which rule-of-thumb settings are optimal. NEG refers to the case in which they are suboptimal. NUL refers to the case in which performance does not depend on hyperparameter settings.

is $k = 10$, as suggested in the original paper [33] and applied in several other studies [124, 108, 96]. On the other hand, a grid search consists in testing hyperparameter configurations at regular intervals in the hyperparameter domain.

In order to test the robustness of rule-of-thumb settings with respect to grid searches, we analyze the performance of LOF, KNN, and OCSVM on all data sets in the BAD data set collection (see Table 5.2). We use the rule-of-thumb value $k = 10$ for both LOF and KNN. For OCSVM, we use the value $\nu = 0.05$, as in the original paper [137]. With respect to grid searches, we consider a grid search in the range $k \in [1, max(n, 500)]$, where $n$ is the cardinality of the data set, for both LOF and KNN. For OCSVM we consider $\nu$ values in the range $\nu \in [1, 100]$.

As our performance metrics we consider both ROC AUC and AP. Results are presented in Figures 5.2 and 5.3. To improve readability we only illustrate one-dimensional grid searches. We also only report the most representative trends. The complete set of results can be found in our online repository.

**Discussion** Figure 5.2 depicts examples in which rule-of-thumb settings are optimal with respect to the grid search considered. In the following, we refer to these cases as *POS*. Figure 5.3 shows cases where rule-of-thumb setting are suboptimal with respect to the grid search. We will refer to these cases as *NEG*. Finally, we will refer to cases in which the performance does not seem to depend on the analyzed hyperparameters as NUL.

Table 5.3 summarizes the experiment results. Among the 24 configurations analyzed (three algorithms on eight data sets), the results show that in 62.5% of the cases hyperparameter tuning via grid search provides a better configuration with respect to rule-of-thumb settings. In the considered examples, the additional effort required to perform a grid search is compensated by the significant performance gain with respect to rule-of-thumb settings. We analyze this performance gain in more details in Section 5.3.3.

The results also show that the relationship between performance metrics and hyperparameter configurations depends on the data set considered. For example, the rule-of-thumb value of $k = 10$ for the LOF method leads to suboptimal performance on the larger convertype and kdd99 data sets (see Figures 5.3.a and 5.3.b). In these instances, the optimal value for $k$ seems to be much higher than the rule-of-thumb setting. This is in line with the fact that the LOF algorithm depends on the analysis of $k$-nearest neighborhoods. In a larger data set neighborhoods are likely to be more dense. Thus, it is more likely that a larger number of data elements needs to be analyzed in order to correctly discriminate anomalous elements.

Figure 5.2.b shows a good example of a metric (ROC AUC) being non-monotonic with respect to the hyperparameter ($k$). In this case, a grid search considering $k \in$

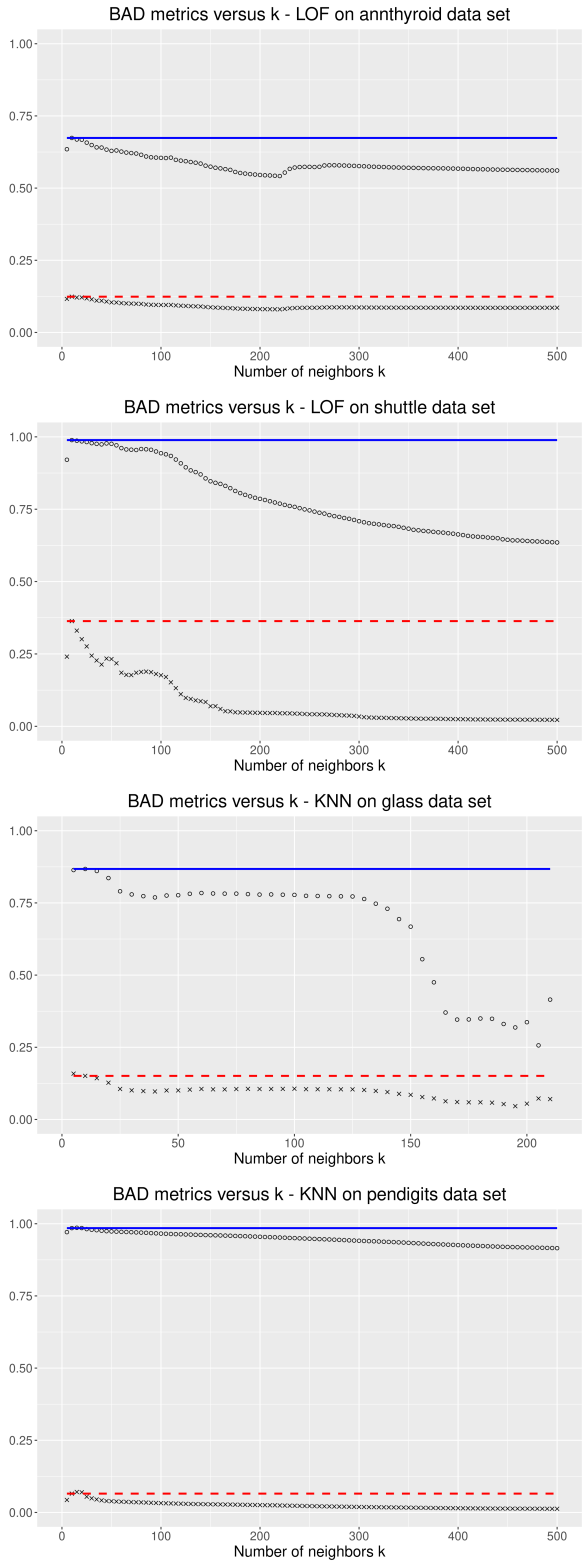


FIGURE 5.2: Rules of thumb analysis. Cases in which rule-of-thumb settings are optimal with respect to the grid search considered (POS). The solid lines and circles represent the ROC AUC metric. Dashed lines and crosses represent the AP metric. Lines correspond to the rule-of-thumb setting. Points correspond to grid search results. For both metrics higher values are better (best printed in colors).

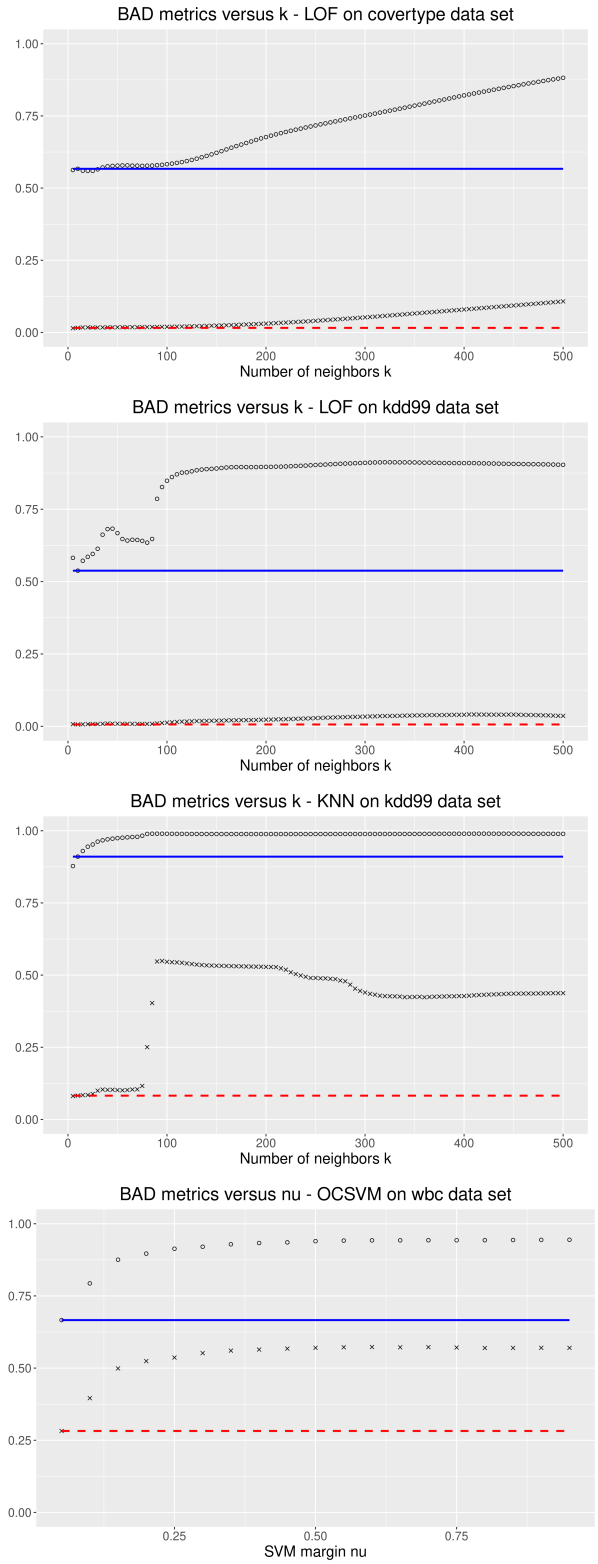


FIGURE 5.3: Rules of thumb analysis. Cases in which rule-of-thumb settings are suboptimal with respect to the grid search considered (NEG). The solid lines and circles represent the ROC AUC metric. Dashed lines and crosses represent the AP metric. Lines correspond to the rule-of-thumb setting. Points correspond to grid search results. For both metrics higher values are better (best printed in colors).

$[0, 100]$ (e.g. [37]) would probably set for the local optimum around $k = 50$. However, a better hyperparameter configuration can be found when considering values larger than one hundred.

Finally, our results also show that considering different metrics provides different insights. For example, even though ROC AUC and AP are correlated, AP is systematically lower than ROC AUC. An extreme example is the KNN algorithm on pendigits (Figure 5.2.d), where ROC AUC is close to one, whereas AP is close to zero. This might be due to the large number of false positives in the results. This analysis demonstrates one drawback of using a single-metric evaluation approach, since a single metric cannot capture all characteristics of a solution.

### 5.3.3 Performance gain

The effort required to perform a grid search must be justified by the performance gain obtained with respect to rule-of-thumb settings. In this section, we analyze this performance gain in more detail. To this end, we present an experimental comparison of the considered algorithms. For each algorithm and data set we report the performance obtained with rule-of-thumb settings, the performance obtained with the optimal hyperparameter configuration found in the grid search, as well as the relative performance gain.

Each grid search is carried on by adopting the following procedure. If an increasing performance trend is found (e.g. Figure 5.3), the hyperparameters are increased until a plateau is found. If a descending trend is found (e.g. Figure 5.2), the current best hyperparameter settings are used. If several hyperparameter settings correspond to the same performance, the smallest hyperparameter value is used.

Consistently with the rest of the chapter, the comparison is repeated for both ROC AUC and AP metrics. In order to produce these results, approximately 50K experiments where executed using BAD. More details on the final hyperparameter settings used in the experiments can be found in **??**. Tables 5.4 and 5.5 present the results for the ROC AUC and AP metrics, respectively.

**Discussion** Considering the ROC AUC (Table 5.4), we notice that, when using grid searches for hyperparameter tuning, either LOF or KNN achieve the best performance on all analyzed data sets. In most data sets considered, LOF and KNN are

| Data set | LOF | | KNN | | IForest | | OCSVM | |
|---|---|---|---|---|---|---|---|---|
| | RoT | GS | RoT | GS | RoT | GS | RoT | GS |
| **annthyroid** | **0.674** | **0.674 (+0%)** | 0.613 | 0.659 (+8%) | 0.617 | 0.661 (+7%) | 0.473 | 0.483 (+2%) |
| **covertype** | 0.567 | **0.973 (+72%)** | 0.852 | 0.905 (+6%) | **0.889** | 0.908 (+2%) | 0.501 | 0.501 (+0%) |
| **glass** | 0.783 | 0.824 (+5%) | **0.869** | **0.869 (+0%)** | 0.7 | 0.711 (+2%) | 0.456 | 0.462 (+1%) |
| **kdd99** | 0.538 | 0.912 (+69%) | 0.91 | **0.99 (+9%)** | **0.989** | 0.989 (+0%) | **0.989** | 0.989 (+0%) |
| **pendigits** | 0.624 | 0.917 (+47%) | **0.987** | **0.987 (+0%)** | 0.738 | 0.881 (+19%) | 0.21 | 0.318 (+51%) |
| **shuttle** | **0.989** | **0.989** (+0%) | 0.974 | **0.989 (+2%)** | 0.858 | 0.88 (+3%) | 0.695 | 0.731 (+5%) |
| **wbc** | 0.895 | **0.954 (+7%)** | **0.954** | **0.954 (+0%)** | 0.952 | 0.952 (+0%) | 0.666 | 0.92 (+38%) |
| **wine** | 0.936 | **1.0 (+7%)** | **0.999** | 0.999 (+0%) | 0.775 | 0.816 (+5%) | 0.519 | 0.681 (+31%) |

TABLE 5.4: Experimental comparison of unsupervised anomaly detection algorithms with respect to ROC AUC. Results in parenthesis show the performance gain with respect to the rule-of-thumb settings. The best performance for each data set and hyperparameter tuning approach (RoT vs. GS) is represented in bold.

| Data set | LOF | | KNN | | IForest | | OCSVM | |
|---|---|---|---|---|---|---|---|---|
| | RoT | GS | RoT | GS | RoT | GS | RoT | GS |
| **annthyroid** | **0.124** | 0.124 (+0%) | 0.102 | 0.121 (+19%) | 0.112 | **0.143 (+28%)** | 0.069 | 0.071 (+3%) |
| **covertype** | 0.016 | **0.278 (+1637%)** | **0.070** | 0.070 (0%) | 0.065 | 0.075 (+15%) | 0.009 | 0.009 (+0%) |
| **glass** | **0.178** | **0.207 (+16%)** | 0.151 | 0.206 (+36%) | 0.097 | 0.108 (+11%) | 0.088 | 0.201 (+128%) |
| **kdd99** | 0.006 | 0.037 (+516%) | 0.082 | **0.549 (+569%)** | 0.361 | 0.446 (+24%) | **0.432** | 0.438 (+1%) |
| **pendigits** | 0.009 | 0.015 (+87%) | **0.065** | **0.072 (+10%)** | 0.004 | 0.009 (+125%) | 0.001 | 0.002 (+100%) |
| **shuttle** | **0.363** | 0.378 (+4%) | 0.205 | **0.386 (+88%)** | 0.062 | 0.099 (+60%) | 0.033 | 0.048 (+45%) |
| **wbc** | 0.267 | 0.583 (+118%) | 0.516 | 0.601 (+16%) | **0.624** | **0.652 (+4%)** | 0.282 | 0.552 (+95%) |
| **wine** | 0.616 | **1.0 (+62%)** | **0.991** | 0.991 (+0%) | 0.178 | 0.235 (+32%) | 0.112 | 0.22 (+96%) |

TABLE 5.5: Experimental comparison of unsupervised anomaly detection algorithms with respect to average precision score. Results in parenthesis show the performance gain with respect to the rule-of-thumb hyperparameter settings. The best performance for each data set and hyperparameter tuning approach (RoT vs. GS) is represented in bold.

the best performing algorithms independently on the hyperparameter tuning approach (best value for both RoT and GS in Table 5.4). This is not the case for IForest which achieves the best performance on covertype and kdd99 when considering rule-of-thumb settings, but is outperformed by LOF and KNN when considering a grid search.

This might explain the results in Domingues *et al.* [52], where the authors follow a rule-of-thumb hyperparameter tuning approach, and where it is reported that IForest outperforms LOF on the data sets considered. Our results after extensive grid search hyperparameter tuning disagree with theirs.

With respect to relative performance gains, we can see that the gain ranges from 0% (where rule-of-thumb settings are optimal, i.e. POS cases) to a maximum of 72% (LOF on covertype). The average performance gain with respect to ROC AUC is 12.4%, possibly justifying the investment in grid searches.

Considering the AP metric (Table 5.5), results notably show that the relative ranking of algorithms has changed. This is analyzed in more detail in Section 5.3.4. The average performance gain with respect to average precision is 119%. Thus, in this case, tuning the hyperparameters via grid searches doubles the algorithm performance on average.

By comparing Table 5.4 and Table 5.5, we can confirm our finding that AP values are much lower than ROC AUC values. This is additional evidence for the fact that ROC AUC tends to overestimate algorithm performance and it should not be considered as the only performance metric.

We were also able to reproduce the results of Campos *et al.* [37], when the optimal hyperparameter settings were found in the grid that the authors considered in their study. In other cases, we found better hyperparameter settings by analyzing a larger grid using the BAD framework.

### 5.3.4 Relative rankings

In this section, we analyze how different hyperparameter settings and performance metrics affect the relative ranking of algorithms in a comparison study. To this end, we use the Kendall ranking correlation coefficient [87] as our main metric. The Kendall's coefficient can be used to establish the correlation between different rankings. A coefficient value of one corresponds to perfect correlation (i.e., the rankings are the same), a value of minus one corresponds to inverse correlation (i.e., the

| Data set | RoT vs GS (ROC) | RoT vs GS (AP) | ROC vs AP (RoT) | ROC vs AP (GS) |
|---|---|---|---|---|
| annthyroid | 1.0 | 0.66 | 1.0 | 0.66 |
| covertype | 0.33 | 0.0 | 0.66 | 1.0 |
| glass | 1.0 | 0.66 | 0.66 | 0.33 |
| kdd99 | 0.0 | 0.66 | 1.0 | 1.0 |
| pendigits | 0.66 | 1.0 | 0.66 | 1.0 |
| shuttle | 1.0 | 0.66 | 1.0 | 0.66 |
| wbc | 0.33 | 0.66 | 0.33 | 0.0 |
| wine | 0.66 | 0.66 | 1.0 | 1.0 |

TABLE 5.6: Kendall rank correlation coefficient for algorithm rankings. A value of one means perfect correlation, a value of zero represents uncorrelated rankings. Only 37.5% of rankings are perfectly correlated, suggesting that different metrics and hyperparameter tuning approaches affect algorithm relative performance.

rankings are one the inverse of the other). Finally, a value of zero corresponds to uncorrelated ranking.

We analyze rankings obtained in four different scenarios:

- **RoT vs GS (ROC AUC) -** The compared rankings are obtained considering rule-of-thumb settings versus grid-search settings. ROC AUC is selected as performance metric.
- **RoT vs GS (AP) -** The compared rankings are obtained considering rule-of-thumb settings versus grid-search settings. Average precision score is selected as metric.
- **ROC AUC vs AP (RoT) -** The compared rankings are obtained considering the ROC AUC metric versus the AP metric. Hyperparameters are set to rules-of-thumb settings.
- **ROC AUC vs AP (GS)** The compared rankings are obtained considering the ROC AUC metric versus the AP metric. Hyperparameters are set to grid-search settings.

Results of this analysis are presented in Table 5.6. As expected, these results clearly show that different metrics and hyperparameter tuning approaches do affect algorithm relative rankings. We obtain perfectly correlated rankings in only 37.5% of considered cases. Thus, in the majority of these there is at least one discordance between the compared rankings.

This analysis validates both the need for reliable hyperparameter tuning in comparative research studies, as well as the advantages of using a multi-metric approach when analyzing anomaly detection algorithms. Notably, both grid searches and the AP metric considerably change the relative ranking of methods up to the point of questioning results published in other comparative studies [52].

### 5.3.5 Scalability

Having demonstrated the advantages of robust hyperparameter tuning practices, in this section we analyze whether BAD is indeed a good framework for performing grid searches.

In the last set of experiments, we compare BAD with the ELKI framework [138]. ELKI is a popular choice among researchers to implement and compare anomaly

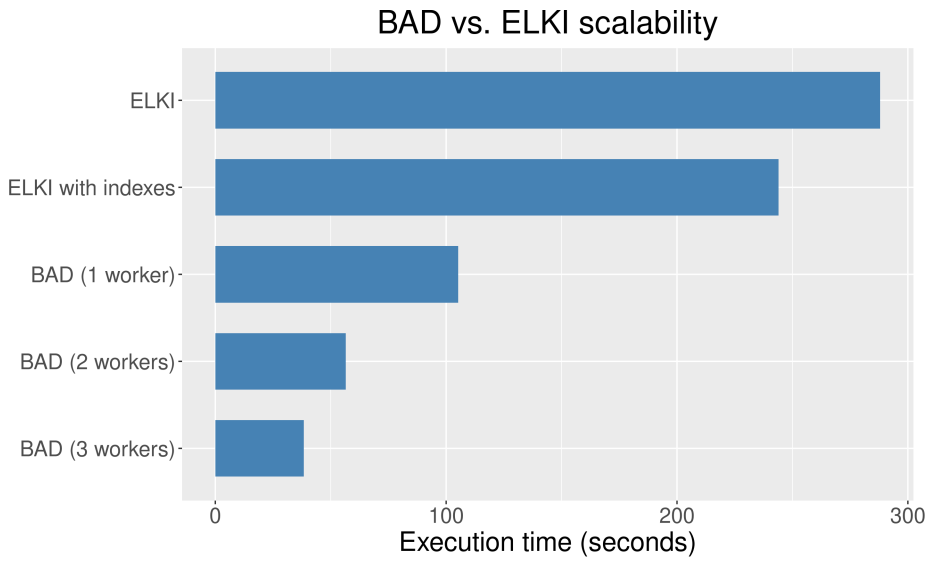


FIGURE 5.4: Scalability of the BAD framework versus ELKI. Execution time refers to running a suite of 100 experiments (LOF on annthyroid data set) with different hyperparameter configurations ($k \in [1, 100]$). Reported times are averaged over ten runs. Even when using a single worker the BAD framework exhibits good scalability with respect to ELKI. Running on more workers improves scalability, as expected.

detection algorithms. ELKI also provides several efficient data structures that can be used to speed up the analysis. We consider these data structures in our evaluation. In our comparison, we use as our metric the experiment throughput, defined in Section 5.2.

For consistency, we run the same experiment suite on both frameworks. The suite represents a grid search for the $k$ hyperparameter of the LOF algorithm on the annthyroid data set. The hyperparameter range corresponds to $k \in [1, 100]$. Reported execution times are averaged over ten executions.

In our experiments, we compare the following configurations:

- ELKI - ELKI implementation of LOF (without indexes).
- ELKI with indexes - ELKI implementation of LOF using a KD-tree in-memory index for faster neighbor searches.
- BAD (1 worker) - BAD implementation of LOF. BAD framework running on a single worker (distributed).
- BAD (2 workers) - BAD implementation of LOF. BAD framework running on two workers (distributed).
- BAD (3 workers) - BAD implementation of LOF. BAD framework running on three workers (distributed).

Results are depicted in the bar chart in Figure 5.4. The plot shows the competitiveness of BAD with respect to ELKI. Even when running on a single worker, BAD is twice as fast as ELKI.

When considering experiment throughput, BAD with one worker achieves 0.95 experiment per second, versus 0.35 for ELKI (0.41 with indexes). When increasing the number of workers, the experiment throughput improves, as expected. Notice

| Data set | LOF ($k$) | KNN ($k$) | OCSVM ($\nu$) | IForest (num. trees) | IForest (sample size) |
|---|---|---|---|---|---|
| **annthyroid** | 11 | 1 | 0.01 | 450 | 100 |
| **covertype** | 1300 | 182 | 0.3 | 50 | 900 |
| **glass** | 17 | 6 | 0.3 | 50 | 300 |
| **kdd99** | 322 | 90 | 0.06 | 450 | 900 |
| **pendigits** | 69 | 13 | 0.01 | 300 | 1000 |
| **shuttle** | 9 | 4 | 0.01 | 100 | 600 |
| **wbc** | 66 | 68 | 0.3 | 450 | 100 |
| **wine** | 28 | 10 | 0.04 | 250 | 100 |

TABLE 5.7: Hyperparameter settings used in this study when optimizing ROC AUC.

| Data set | LOF ($k$) | KNN ($k$) | OCSVM ($\nu$) | IForest (num. trees) | IForest (sample size) |
|---|---|---|---|---|---|
| **annthyroid** | 11 | 1 | 0.01 | 450 | 100 |
| **covertype** | 1500 | 10 | 0.3 | 50 | 900 |
| **glass** | 5 | 1 | 0.06 | 50 | 300 |
| **kdd99** | 350 | 93 | 0.03 | 50 | 500 |
| **pendigits** | 60 | 13 | 0.02 | 300 | 1000 |
| **shuttle** | 9 | 4 | 0.01 | 450 | 700 |
| **wbc** | 97 | 309 | 0.3 | 450 | 100 |
| **wine** | 28 | 10 | 0.16 | 500 | 100 |

TABLE 5.8: Hyperparameter settings used in this study when optimizing AP.

that the performance gain is not linear due to the cluster management overhead and network communication delay. This is expected, as these are common bottlenecks in distributed systems.

These results show that BAD is able to efficiently run extensive hyperparameter searches. This makes it a good fit for benchmarking and hyperparameter tuning of anomaly detection algorithms.

### 5.3.6 Replicability

In this section, we report the hyperparameter settings used to run the experiments described in this chapter. For stochastic algorithms, i.e. IForest, the random number generator seed was varied between one and ten and the reported results are averaged over those ten runs.

These hyperparameter settings where found following the heuristic grid search procedure described in Section 5.3.3. It is possible that by considering a larger grid better hyperparameter settings could be found.

Table 5.7 reports, for each algorithm and data set, the optimal hyperparameter settings with respect to the ROC AUC metric. Table 5.8 illustrates the optimal hyperparameter settings for the AP metric.

## 5.4 Summary

In this chapter, we presented the BAD framework for benchmarking anomaly detection. BAD enables the execution of a large number of anomaly detection experiments in parallel on commodity hardware. This is achieved by distributing the computation on a cluster of machines. Running a large number of experiments in parallel is

important both for hyperparameter tuning of anomaly detection algorithms, as well as for comparative research studies.

The traditional approach for hyperparameter tuning in anomaly detection is to set hyperparameters via rules of thumb. We showed that this approach is unreliable and leads to suboptimal performance with respect to grid searches in the majority of cases considered.

We showed that hyperparameter tuning via grid searches leads to large performance gains with respect to rule-of-thumb settings. This is particularly significant when considering the AP metric over the ROC AUC. We also demonstrated that different metrics and hyperparameter tuning approaches influence the relative ranking of anomaly detection algorithms in comparative evaluation studies.

Having proven the advantage of grid search hyperparameter tuning, we analyzed the efficiency of BAD in running grid searches with respect to the popular ELKI data mining framework. Our results showed that, in this task, BAD outperforms ELKI, even when running on a single worker.

We release the BAD framework as open-source software for the community to provide feedback and contribute to it. We hope that our work will contribute to the development of robust and reproducible research in the anomaly detection field.

# Part III

# Scalable Anomaly Detection

# Chapter 6

# Cost-aware Data Analysis

In this chapter, we begin to investigate the scalability of anomaly detection techniques. A common approach to scaling data analysis tasks is to distribute the computation on a cluster of machines. A distributed system is able to achieve superior performance by managing a large pool of resources as a single entity. However, in the industrial context, performance is not the only important metric. For example, when comparing equivalent solutions to a given task, the cost of the solution becomes an important factor.

This chapter presents our analysis of the price-performance trade-offs in the context of a streaming anomaly detection task. This contribution deals with the following research question: "At which data scale is a distributed approach more cost-effective than a single-threaded application with respect to the anomaly detection task?".To this end, we describe our empirical evaluation of two equivalent anomaly detection solutions by considering price-performance metrics. Our use case is taken from the Telecommunication industry. Our results show that, in case of both online and periodic analysis, the benefits of distributed processing are outvalued by the higher cost of distributed data ingestion. However, if we fix the data ingestion costs, our results show that the most cost-effective solution depends on the data set size.

The work described in this chapter was developed at the Semantics and Knowledge Innovation Lab (SKIL) at TIM[1].

Some of the contents of this chapter have been previously published in the proceedings of the 12th ACM International Conference on Distributed and Event-based Systems [22].

## 6.1 Introduction

The massive amounts of data we deal with daily is pushing the development of dedicated tools for their collection, storage, and analysis. The most well-established tools to deal with such *Big Data* are distributed computing systems. A distributed system is made up of several independent components, or nodes, that communicate with each other to solve a given problem. For example, a node could be a PC or a wireless sensor. The great advantage of distributed systems is that they can seamlessly manage a large pool of resources as a single entity.

Modern computing infrastructures increasingly rely on distributed systems to manage huge volumes of data, and to increase performance and reliability. Cloud service providers, such as AWS[2] and Microsoft Azure[3], are key enablers for this

[1] https://www.tim.it/
[2] https://aws.amazon.com/
[3] https://azure.microsoft.com/

trend. These providers make computing resources available to their customers under a pay-per-use billing policy. This model greatly reduces operational costs with respect to the total cost-of-ownership. Also, this eases the development of distributed applications running on clusters of computers.

Nonetheless, when dealing with data analysis tasks, distributed systems are just one of the possible solutions. A drawback of distributed systems is that they add a significant overhead to many use cases. Additionally, using several components usually implies multiplying the total cost of the solution by the number of components. Thus, going for a distributed solution whenever a data analysis need arises, might not be the best choice, especially in cost-aware environments.

In recent years, the benchmarking of distributed systems against single-threaded implementations has drawn some attention [114, 31]. One of the main results [114] reports that expertly implemented single-threaded solutions can outperform state-of-the-art distributed systems, even at scale. The analysis is carried out on various graph processing tasks. The reported results show that single-threaded libraries outperform distributed solutions by an order of magnitude, at a fraction of the cost. This approach highlights some flaws in the current evaluation methodology of distributed systems and algorithms, where single-threaded implementations are usually not considered as baselines.

In this work, we present an empirical comparison between a distributed solution and an equivalent single-threaded implementation for a streaming anomaly detection task. The focus of our analysis is less on performance, and more on the total cost of solving the task. This shift is motivated by the industrial setting in which this work was conceived. In industry, solutions must be evaluated both in terms of cost-effectiveness and efficacy.

It is well established [72] that performance metrics are frail when they ignore cost-related indexes. For this reason, differently from previous work [14, 43] that focuses on latency and throughput, we base our analysis on the total solution cost. This cost is obtained by multiplying the price-per-second of the machines storing the data and running the solution by the execution time needed for the analysis task. With this choice, we want to highlight the cost-effectiveness of a solution.

Our use case is an on-line anomaly detection task. Our goal is to detect unusually crowded areas in a city. Our dataset consists of the mobile phone connection data collected in the city. The possibility to perform this task is well documented in [98, 35, 34]. Both of our solutions use the same anomaly detection strategy. This consists in a statistical model-based anomaly detector trained on historical data [23].

We compare the performance of two equivalent solutions for this task. The first solution is based on Natron, the most recent version of [24, 21], a single-threaded framework for stream processing. The second solution is based on Apache Spark Structured Streaming API [162]. Both solutions use Apache Kafka [94] as their storage layer. We describe the tuning of both solutions to our particular use case. Then, we compare them on the total cost required to solve the anomaly detection task. In order to assess the solutions' scalability, the analysis is replicated multiple times and for different data volumes.

The design of an industrial solution also requires operational considerations. With the term operational, we refer to the choices regarding when and how data is ingested, stored, and processed. Depending on the use case, there might be different operational requirements. In our use case, data is generated continuously from the mobile phone network. To avoid data losses, our only operational requirement is that data must be ingested continuously. For our analysis, we consider the following two consumption policies: (i) *continuous*; data is consumed in real-time as soon as it

is ingested, and (ii) *periodic*; data is consumed at regular time intervals (e.g., once a day, or once a week). These choices influence the total solution cost. For example, if we want to analyze data continuously, we need dedicated hardware running 24/7.

The chapter is organized as follows. We begin the chapter by describing our data and our problem setting (Section 6.2). We introduce the software systems used (Section 6.3), and present our considerations regarding the design of the solutions (Section 6.4). Then, we describe our methodology and experimental settings (Section 6.5). Finally, we present our results in Section 6.6. In Section 6.7, we conclude the chapter and present possible extensions.

## 6.2 Problem setting

In this section, we present our data set and our problem setting. The data set comes from a real-world industrial use case in the mobile telecommunication (a.k.a., telco) sector.

### 6.2.1 Data description

Mobile phone data can offer relevant and real-time hints about the presence of people in a geographical area [36, 98, 35, 34, 23, 51]. Such analyses can be used to describe a territory's macro-dynamics. In particular, we refer to mobile phone connection data, commonly known as *Call Detail Records* (or CDRs). Every mobile phone generates a CDR every time a call, SMS or Internet connection is made. The CDR contains information about the customer, the type of connection, and the cellular tower instantiating the connection. This data is used by telco companies for billing purposes.

Each CDR can be associated to a base tower, and each tower can be associated to a geographical location. Then, it is possible to map mobile phone activity to geographical locations. The number of active mobile phones at a location, computed following a privacy-preserving methodology, can approximate the number of people in the area. To make analysis more understandable, the mobile network is often approximated with a regular grid[4]. In our case, each grid cell represents a 250x250 meters square. We term each square a *pixel* and we represent the city as a series of frames composed by pixels [20].

In this work, we use CDRs collected in the city of Milan, Italy, during the months of February, March, April and June 2016. Data was made available thanks to the collaboration with TIM – Telecom Italia.

In order to preserve user privacy, these data are aggregated at pixel level using 15-minutes-long windows; that is we count the number of distinct mobile phone users in each pixel each 15 minutes to generate a time series of integers per pixel. If the counting goes below a given threshold, it is set to zero.

The data collected in the month of April is the most significant; in this period the city of Milan hosts a design festival[5] that attracts half a million of visitors, and an anomalous density of people can be detected in the 11 districts of Milan that host the 1.151 events [23] of the festival. This dataset comprises CDRs of calls and SMSs collected between April 13th and April 17th 2016. CDRs of Internet connections are filtered out since this data is missing in the majority of the months considered.

[4]For an industrial implementation of this solution see TIM Big Data https://www.olivetti.com/en/retail/data-driven-solutions/tim-big-data

[5]http://archivio.fuorisalone.it/2016/en

This one-week dataset occupies 1.7GB, and contains around 24 millions calls and 17 millions SMS records. We name this dataset Mobile 1, and we shorten it as MOB1.

We use the rest of the data (March, February, June) for training the models described in Section 6.2.2. The cost of this activity is not considered in this work.

In order to include the scalability dimension in our analysis, we generated several datasets by scaling our original MOB1 dataset. The scaling procedure takes as input an integer scaling factor $k$, and it replicates each CDR in the dataset $k$ times.

Through scaling, we generated several additional datasets for our experiments. The most representative ones are:

- MOB1 (1.7GB), original dataset, representative of weekly mobile traffic (excluding Internet connections) in a large metropolitan area (Milan).
- MOB10 (17GB), $k = 10$, representative of weekly mobile traffic (including Internet connections) in a large metropolitan area (Milan).
- MOB30 (50GB), $k = 30$, representative of weekly mobile traffic in a country (Italy).
- MOB50 (83GB), MOB100 (170GB), extreme situations.

Representative sizes are based on internal TIM metrics. Unfortunately, all datasets used in this study are not available for public disclosure under TIM policies. Aggregated data similar to the one we produced internally when processing the raw CDRs is available as part of the TIM Big Data challenge 2015 dataset[6].

### 6.2.2 Problem

In our use case, we are interested in finding out which areas of a metropolitan city are unusually crowded. The people present in a certain area can be approximated by the number of active mobile phones in the area.

We can cast this use case into an on-line time series anomaly detection problem. Anomaly, or outlier, detection is a fundamental data analysis task [3]. An anomalous pattern could be composed of a single or several data elements. Anomaly detection relies on the ability of building a model of normality for a system or phenomenon. The model is then used to detect anomalies by computing the "distance" between the model and the anomalous element.

In our case, an anomaly represents an infrequent event in the city, which attracts a large number of people. A model of normality can be built by analyzing mobile phone data in periods where no event occurs. This is usually known as *training* in the machine learning community. Then, the trained model is compared with the collected data to detect anomalies.

We follow the anomaly detection approach described in [23]. For training, we consider a time series for each geographical pixel. The series contains the number of mobile phone connections inside the pixel aggregated every fifteen minutes. Following [23], we assume each pixel follows a Gaussian distribution, and we approximate its parameters by computing the sample mean and variance in periods where no sizable event happens (i.e., in February, March, and June). We repeat this process for weekdays and weekends, since they present different mobile activity patterns. This accounts for $2 \times 24 \times 4 = 192$ models for each pixel, i.e., 1.92 million models

[6] http://www.telecomitalia.com/tit/en/bigdatachallenge.html

considering the 10.000 pixels the city is divided in. Anomalies are detected at runtime by joining each pixel measurement with the corresponding model distribution. Measurement $x$ is reported as an anomaly if its z-score is larger than 3, that is

$$\frac{|\bar{\mu} - x|}{\bar{\sigma}} > 3 \tag{6.1}$$

where $\bar{\mu}$ and $\bar{\sigma}$ are the estimated mean and standard deviation for $x$'s pixel in the corresponding fifteen minutes slot.

Note that there exists a plethora of more advanced anomaly detection techniques (for an extensive reference see [3]). Finding the most accurate detector is outside the scope of this work. We use the Gaussian model since it has been show [23] to be well-fit for the problem at hand. In particular, our method can be executed in parallel on a cluster of computers, since every pixel can be analyzed independently from the others. As already mentioned, the only operational requirement for our use case is that data is collected in real-time to avoid data losses.

## 6.3 Background

In this section, we briefly present the software systems used in our solutions. Interested readers can learn more by checking out the references.

### 6.3.1 Apache Kafka

Apache Kafka [94] is a distributed message broker with stream processing capabilities. Kafka organizes data into *topics*. Each topic is made up of one or several *partitions*. Each partition is assigned to a node in the Kafka cluster.

The Kafka APIs are based on the *producer* and *consumer* components. The producer is responsible for transferring data from an external source to a Kafka cluster. Conversely, the consumer is responsible for reading data from a Kafka cluster and sending it to an external sink. By instantiating and using these components, an application can integrate Kafka as its storage solution.

Kafka is designed to enable high-throughput applications, and it supports at-least-, at-most-, and exactly-once message delivery.

### 6.3.2 Natron

Natron is a general-purpose, pluggable system for stream processing implemented in Java. Natron exploits the *generic functions* abstraction, to deal with data *variety*, and it generalizes the Streaming Linked Data (SLD) framework [24, 21]. Natron's users can define pipelines that continuously receive external data streams from several sources, publish them on an internal type-agnostic shared bus, process them and emit results to multiple sinks.

Figure 6.1 illustrates the architecture of the Natron framework. Natron includes three main components: Receivers, Processors and Translators. These components are interconnected through a Generic Data Stream Bus.

A typical Natron application receives a set of streams as input through a Receiver component. Receivers are responsible for accessing external data streams and push their contents to the internal Generic Data Stream Bus. Data is analyzed by Processors. A Processor manages and manipulates internal data streams, i.e. it

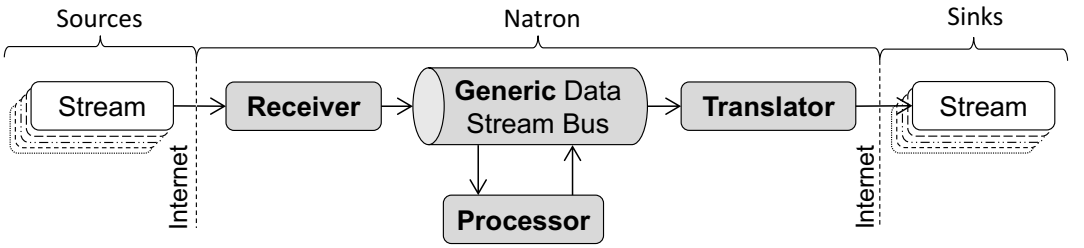


FIGURE 6.1: The architecture of Natron

creates and processes portions of internal streams by applying time-based or size-based windows. Processors' intermediate results are then published to the Generic Data Stream Bus. Finally, Translator components publish the final results to external sinks. Natron relies on a monolithic single-threaded architecture.

### 6.3.3 Apache Spark

Apache Spark [162] is a distributed processing engine which improves upon the MapReduce [50] programming model for processing massive amounts of data in parallel. The main advantage over MapReduce is that intermediate results can be stored into main memory, thus reducing disk I/O operations.

Spark environment consist of several components, which communicate with each other via the network. The highest level components are the *master* and *worker*. The master is responsible for coordinating the execution of a Spark application and presenting its results. Workers are responsible for managing the execution of distributed application code. There can be more than one worker, and each physical machine can host several workers. Master and workers are implemented as separate processes running in the JVM.

Apache Spark is based on the Resilient Distributed Dataset (RDD) abstraction. An RDD represents an immutable dataset distributed over a cluster of machines. Each fragment of the dataset is termed a *partition*. A Spark application consists of a sequence of transformations on a collection of RDDs. During execution, these transformations run in parallel on each partition. When an aggregated result is needed, e.g. COUNT after GROUP BY, Spark performs a shuffle operation by transferring partitions over the network between master and workers. Each worker spawns several subprocesses known as *executors*. Executors run the distributed application code. The atomic unit of parallel execution is called a *task*. At runtime each task is assigned to an executor.

## 6.4 Solution design

The cost of an analytics solution depends on infrastructural, architectural, and operational choices. We here describe our considerations in designing the solutions.

Our data analysis task can be decomposed into three main phases (see Figure 6.2):

- data ingestion – data is collected from the mobile network and transferred to a storage layer.
- data consumption – data is transferred from the storage layer to the analysis layer.

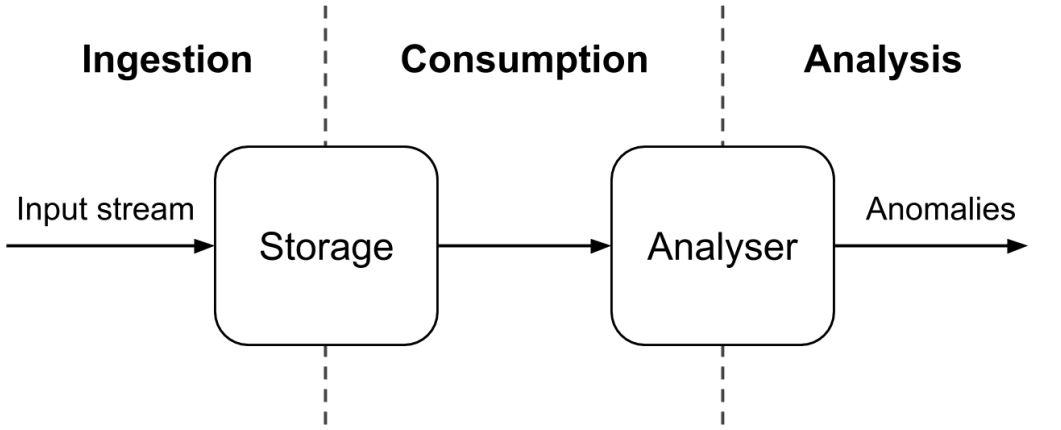


FIGURE 6.2: General architecture of our solution

- data analysis – data is processed and results are generated by joining streaming data with the static models.

Note that we add a storage layer between ingestion and consumption to decouple the two phases. This means that we can ingest data in real-time, and analyze it at a later stage. This also enables various operational scenarios.

### 6.4.1 Infrastructure

An infrastructural choice specifies where a solution is deployed. The hardware used to run an application can be bought, or rented from a cloud service provider. We restrict our analysis to cloud services, since they usually reduce the operational cost of the solution.

When instantiating virtual machines (VMs), cloud service providers usually offer two types of billing policies: pay-per-use instances and reserved instances. Reserved instances (RIs) can be held for a fixed amount of time at a reduced price with respect to pay-per-use instances. RIs are well-fit to reduce the cost of continuous data analysis solutions, while pay-per-user instances are better fit for bursty workloads, such as periodic analysis tasks. In the following, we refer to pay-per-use instances as shared.

Table 6.1 presents the characteristics of the virtual machines used in this study. The last column contains the approximated cost of running a shared instance versus a reserved instance. The reported costs and characteristics refer to Fsv2-series VMs

TABLE 6.1: Azure VM sizes (January 2018)

| VM Type | Cores | RAM (GB) | S/R (€/month) |
|---|---|---|---|
| VM1 | 2 | 4 | 64.62/60.33 |
| VM2 | 4 | 8 | 127.99/121.58 |
| VM3 | 8 | 16 | 256.61/242.42 |
| VM4 | 16 | 32 | 513.23/484.92 |

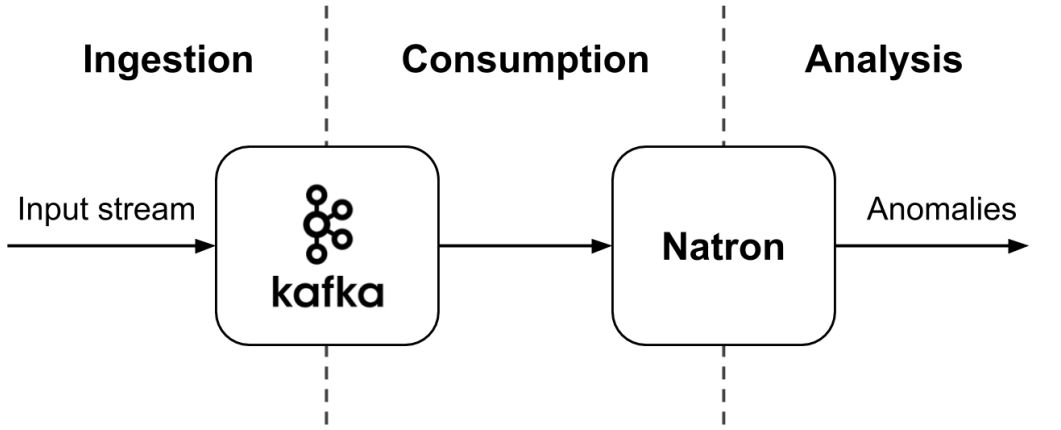


FIGURE 6.3: Architecture for the single-threaded solution

of Microsoft Azure[7]. We chose the Fsv2-series because it is equipped with computation optimized hardware that fitted our needs at affordable cost. Nevertheless, reported costs do not differ significantly from those of other cloud service providers.

### 6.4.2 Architecture

We designed our solutions according to the general architecture depicted in Figure 6.2.

The storage layer is responsible for ingesting data in real-time from the mobile network. Due to the arbitrary velocity of the mobile data stream, the streaming storage must be able to scale seamlessly to huge data volumes. Moreover, the streaming storage must be able to record data continuously, since this is one of our operational requirements. The space required to store the generated models is constant, and it can fit comfortably into memory. The storage cost for the raw CDR data is not considered in our analysis, since in the real use case we can aggregate data on-line using windowing operators.

The analysis layer is responsible for processing data and producing results. The analysis layer communicates with the storage layer to retrieve the data, and it produces the results by performing the necessary aggregation queries. Data processing can happen continuously or periodically. We consider both settings in our analysis.

Note that our architecture is related to the lambda architecture [112], since we produce results by combining data from both batch and speed layers.

### 6.4.3 Implementation details

In this section, we present some details about the implementation of our single-threaded and distributed solutions. Both solutions use Kafka as storage.

#### Apache Kafka

In this work, we use Kafka as our streaming data storage. We use two different configurations, one for each solution. The single-threaded solution, based on Natron, reads data from a single VM1 machine. The distributed solution, based on Spark, reads data from a Kafka cluster composed of four VM2. In the distributed setting,

[7] https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-compute

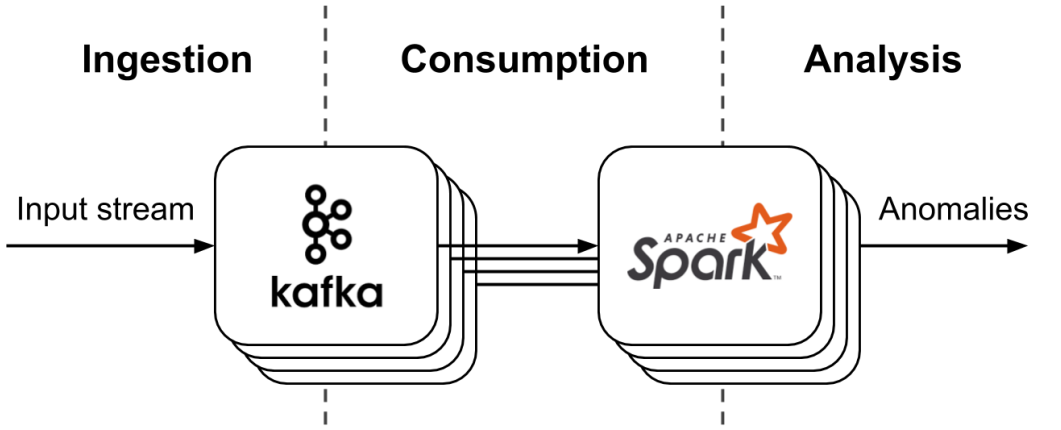


FIGURE 6.4: Architecture for the distributed solution

we set the number of partitions for each topic to eight. We choose this value by considering the number of executors used in the experiments, since executors can read in parallel from different partitions.

### Natron

In the single-threaded solution, the consumption phase is implemented using a Kafka Receiver that polls the data from the server in comma separated value format. The architecture for this solution is depicted in Figure 6.3. The Receiver connects to an Apache Kafka server that provides the data. The data enters the system as a stream of generic objects. Each object contains its event timestamp. Downstream to the Receiver, a Processor takes the data from the Bus and transforms each element into a domain-specific Java object (i.e. a Java representation of a CDR, named PixelCDR).

Then, an Esper[8] Processor performs the analysis. The internal stream of PixelCDRs flows into Esper, which performs the query presented in Listing 6.1. The query counts, every 15 minutes, the number of calls/SMSs grouped by pixelId, i.e. the pixel identifier. The window operation is performed on the event timestamp. We use Kafka exactly-once message delivery to analyze the whole data stream. The query produces the list of anomalous pixels. The anomalies are identified using the *isAnomalous* user defined function, that access the models file, stored in memory, and implements Equation (6.1). The query results are then saved to the file system by a Translator.

```
SELECT pixelId, MAX(timestamp)
FROM PixelCDR.WIN:EXT_TIMED_BATCH(timestamp, 15 min)
GROUP BY pixelId
HAVING isAnomalous(pixelId, COUNT(*), MAX(timestamp))
```

LISTING 6.1: EPL query performed by Esper Processor

### Apache Spark

We implemented our distributed streaming pipeline using Spark Structured Streaming. Spark Structured Streaming is the relational streaming API for Apache Spark. It enables the evaluation of continuous query over both static and streaming data.

[8] http://www.espertech.com/esper/

Structured Streaming works on dynamic relational tables that get updated as the stream data arrives. Through the Structured Streaming API, users can express relational (SQL) queries over streaming data and generate result tables.

In our implementation, we register both the static models table, and the CDR data stream as temporary views that can be queried through the Structured Streaming API. The CDR view is actually a dynamic table that gets updated as data is ingested. The anomaly detection method is implemented as a SQL query that performs a join on the aforementioned tables, and filters the results based on the anomaly condition defined in Equation (6.1). Listing 6.2 contains the pseudocode for the query.

The distributed application is deployed on a multi-node Spark cluster, while data is ingested from a multi-node Kafka cluster. This deployment is represented in Figure 6.4. Spark is integrated with Kafka to provide parallel reads from multiple Kafka partitions. Section 6.5.2 presents more details about the Spark cluster configuration.

We choose Apache Spark for our distributed solution due to its wide spread use in industry, and the availability of previously developed source code and expertise. In Section 6.7 we propose future works considering other distributed processing engines.

```
SELECT pixelId, timestamp
FROM (
SELECT cdrs.pixelId, cdrs.timestamp, COUNT(1)
FROM cdr_stream AS cdrs
WINDOW ON cdrs.timestamp EVERY 15 minutes
GROUP BY cdrs.pixelId
) AS windowed_cdrs LEFT JOIN models ON models.timestamp = windowed_cdrs.window.start
WHERE isAnomalous(windowed_cdrs.value, model.mean, model.sd)
```

LISTING 6.2: Spark SQL anomaly detection query

### 6.4.4 Operational considerations

Operational requirements are related to business choices. They deal, for example, with how often a result report should be produced. We consider the following two operational scenarios:

- *Continuous ingestion – continuous consumption and analysis* This scenario includes real-time use cases, such as crowd monitoring for security purposes. Data is consumed as soon as it is produced, and the delay with which results are produced corresponds to the latency of the system. In this regime, results are produced continuously with whatever latency the system might have. This scenario requires the continuous utilization of reserved resources, since the solution must run without interruptions.

- *Continuous ingestion – periodic consumption and analysis* Periodic analysis represents a common scenario. In many use cases, the results of the analysis can be summarized in a periodical report, and the real-time analysis is not necessary. The ingestion layer must still run continuously to avoid data losses. On the other hand, the analysis layer can be allocated only for the amount of time needed to perform the analysis and generate the results.

Another important considerations when designing an industrial analytics system are fault-tolerance and redundancy. Apache Kafka and Apache Spark respectively provide out-of-the-box redundancy and fault-tolerance. Nonetheless, we do

not include these aspects in our analysis, since the total solution cost of a fault-tolerant system can be approximated as the total solution cost multiplied by the redundancy factor. If we apply this consideration to both solutions, it does not affect our final results.

## 6.5 Experimental settings

In this section, we describe our experimental methodology and the tuning of our solutions that led to the final settings used in the experiments.

### 6.5.1 Methodology

The goal of our experimental methodology is to find the most cost-effective solution for the given problem. To assess this, we run our solutions on both real and simulated problem instances. The real data MOB1 is collected from the mobile phone network of TIM. Starting from this real data we generated several other datasets (MOB10, MOB30, etc.). These datasets were generated to analyze the scalability of our solutions. The dataset generation procedure is described in Section 6.2.

We compare our solutions based on their total cost when they both provide correct results. This is not always the case since the most economic single-threaded configurations struggle to deal with the most demanding problem instances. The solution cost is computed by multiplying the cost of the solution (i.e., price-per-second of the used VMs) with the execution time of the experiment (if completed correctly). The cost of the solution also depends on the operational requirements, e.g., a continuous solution can run on reserved instances, thus reducing the price-per-second.

We executed all experiments on Microsoft Azure Linux VMs. For each experiment, we performed five experimental runs. All reported results are average over four runs by discarding the worst outcome. We do not include error bars in the plots since their bounds are so tight that they simply overlap with the point shapes and clutter the images.

We do not consider latency in our analysis due to the following reasons:

1. In the continuous analysis scenario, at regime the latency of the system does not influence the stream of results. Moreover, the latency to analyze one minute of data is below 1.5 seconds for both solutions, which is appropriate for our use case.

2. In the periodic analysis scenario, the latency of both systems is negligible with respect to the periods considered (i.e. every day or every week).

Thus, in the following we omit latency from our discussion.

### 6.5.2 Configurations

In this section, we present our analysis on the configuration for our solutions. Our goal is to find the most cost-effective configuration which solves the problem. We restrict our analysis to Fsv2-series VMs under the assumption that in cost-aware scenarios more general-purpose VMs are preferrable to workload-optimized VMs, since they can be shared and used by different workloads. Figures 6.5 and 6.6 show the solution cost as a function of the scale factor for different configurations of Natron and Spark.

**Natron**

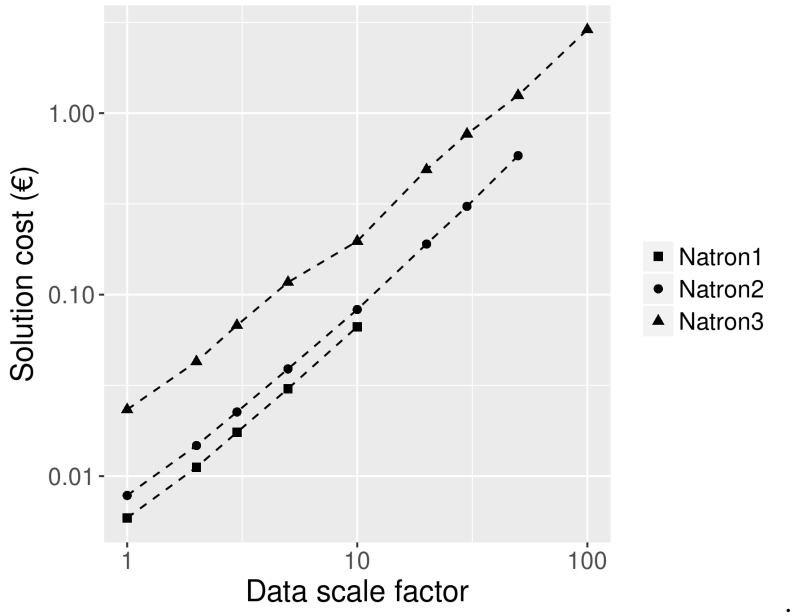


FIGURE 6.5: Solution cost over data scale for different Natron configurations.

Natron was deployed using a docker container to create a sandbox environment and to ease the monitoring operations for CPU and memory consumption. The whole infrastructure needed a single VM for each experiment in addition to the VM needed for the data provider, i.e. a single partition Kafka server on a VM1. We run multiple experiments for each dataset and remove the outliers, e.g. the first run of each experiments was considered as a system setup, collect data result for correctness check, i.e. anomalies, and CPU/memory consumption log to monitor the health of the infrastructure. During each run the container exploits all available virtual machine resources for the computation. We vary the dimension of the VMs in azure to stress the environment and get the upper limit of the resource needed to handle a given amount of data.

We experimented with three configurations, having different number of cores, RAM, disk I/O, and network I/O available:

- Natron1: one VM1
- Natron2: one VM2
- Natron3: one VM4

The single-threaded implementation suffers from the volume of the data, a single VM cannot scale horizontally to deal with a continuously increasing amount of data. Figure 6.5 clearly shows that the different configurations can bear different loads of data. Natron1 can handle dataset MOB1, which represents the original data size, and can perform the anomaly detection in about 120 seconds. This configuration can handle up to dataset MOB10, but bigger dataset results are problematic. Configurations Natron2 and Natron3 can bear at most dataset MOB50 and MOB100 (respectively), but are more expensive than configuration Natron1 . The three chosen configurations widely explore the hardware offerings in order to find the best

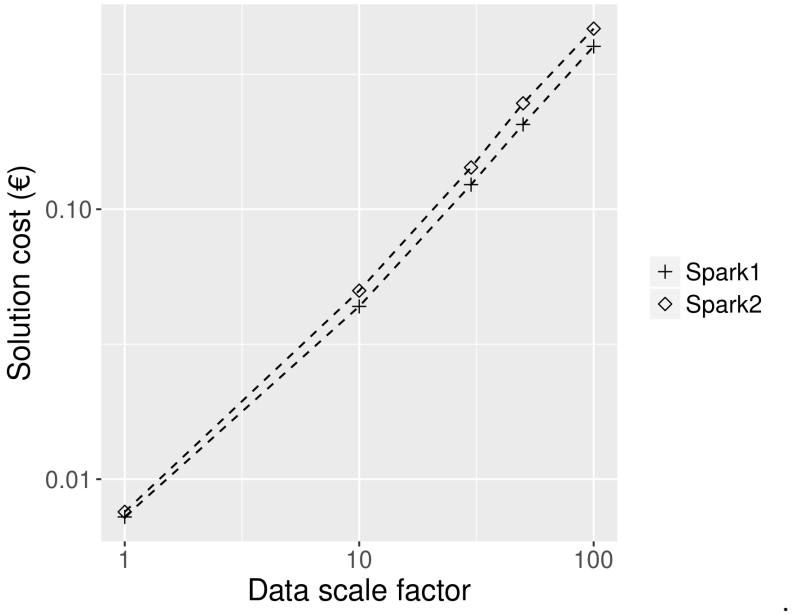


FIGURE 6.6: Solution cost over data scale for different Spark configurations.

solution related to the data loads. Due to the variability of configurations' behaviors, we tested the system against more dataset than the ones listed in Section 6.2.1, i.e. we tested dataset with scale factor k=2, k=3, k=5, and k=20.

We compare all the three Natron configurations with the best configuration chosen for the distributed system in order to have a complete overview for the different input volumes. During the experiments, regardless of the Natron configuration, the normality models are loaded in memory, while streaming data is read from the Kafka cluster described in Section 6.4.3.

### Apache Spark

We tuned our Spark cluster using the total solution cost as a metric, and experimenting with three parameters which commonly affect Spark's performance. Our intention here is to present our findings on the best Spark configuration for our specific use case, datasets, and problem setting. We implemented our Apache Spark cluster using Azure Linux VMs (see Table 6.1).

We experimented with the following cluster parameters:

- Virtual machine size,
- Number of executors per worker (or number of cores per executor), and
- Memory allocated per executor.

All other parameters were set to their default values. Note that, since in Microsoft Azure each virtual core (vCPU) corresponds to a single thread, in the following we use the terms core and thread interchangeably.

**Virtual machine size -** Cloud service providers offer several VM types. These types vary depending on the number of cores, RAM, disk I/O, and network I/O available to user applications running on the VM. Thus, an important consideration when deploying a cloud solution is the choice of VM type.

We evaluated two different cost-equivalent configurations for our Spark cluster (refer to Table 6.1 for VM characteristics):

- Spark1: one VM2 master and four VM2 workers
- Spark2: one VM2 master and two VM3 workers

Note that we also experimented with smaller cluster configurations (e.g., a single VM2 worker). However, we found that these were not as cost-effective as the configurations described above. This might seem counterintuitive. However, consider that a smaller configuration usually takes more time to perform the analysis. Since our metric is the total solution cost, to be cost-effective a solution's cost reduction should compensate for its performance penalty.

As an example, we found out that a cluster with a single VM2 worker takes from 2.75 to 3 more time (depending on the dataset) to perform the task with respect to our Spark1 configuration, while only costing 2.5 less.

Figure 6.6 shows the cost of the solution for both configurations. All Spark settings were set to their default values (all available cores, 1GB of RAM per executor). The figure highlights that the Spark1 configuration tend to be more cost-efficient, even though the total number of used cores is the same in both configurations.

Even after tuning both clusters, i.e. by changing the default parameters, we could not find a configuration for Spark2 outperforming Spark1. We used Spark1 for all other experiments. The two following sections provide more details on the experiments we performed measuring the sensitivity of the selected configuration to changes in the number of cores per executor and in the amount of RAM per executor.

**Cores per executor -** An important parameter in Spark configuration is the number of cores allocated to each executor. The default configuration allocates all available cores. Incidentally, the number of cores per executor also determines the number of executor processes that a worker can spawn. Thus, we perform our sensitivity analysis in term of executors per worker. We fixed the total RAM to 24GB and varied the number of executors per worker machine. Figure 6.7 shows our results. We can see that having a single executor on each worker outperforms other configurations. This is supposedly due to the fact that when multiple executors reside on the same machine, the JVM must handle a large volume of I/O network traffic in order for them to communicate. This could possibly influence application performance.

**RAM per executor -** Another important parameter is the amount of RAM designated to each executor. In this case, we picked the best configuration from the previous analysis, i.e. one executor per worker, and varied the RAM allocated to each executor. Figure 6.8 shows our results to this sensitivity analysis. We can notice that the amount of memory allocated to each executor does not seem to affect execution time. This is surprising, considering the common knowledge that Spark performance is proportional to the amount of main memory available. However, our particular use case, i.e. windowed and watermarked relational query, is executed considering one window of data at a time. Even at maximum scale (x100), our windows do not exceed 1GB of RAM, and therefore in this particular scenario the system is not memory-bounded.

All the following experiments were executed using configuration Spark1 with 4 cores and 3GB of RAM per executor. The normality models are stored in a static file over the Spark cluster, while streaming data is read from the Kafka cluster described in Section 6.4.3.

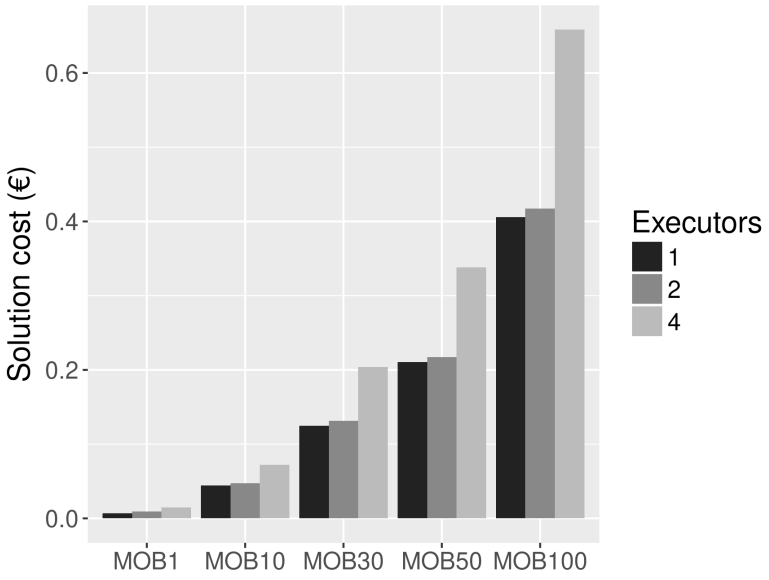


FIGURE 6.7: Solution cost over number of executors per worker on different datasets (RAM at 24GB).

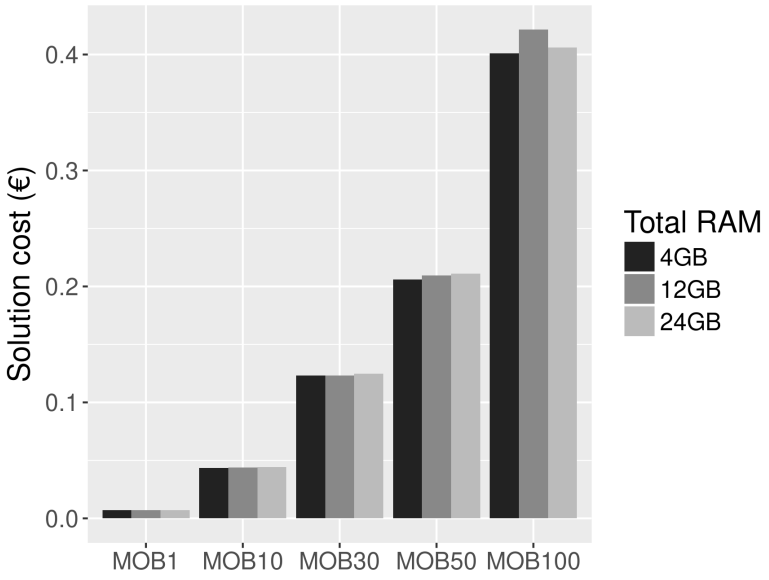


FIGURE 6.8: Solution cost over total number of RAM in GB for different datasets (1 executor per worker).

## 6.6 Results and discussion

In this Section, we present our experimental results. We organize our discussion based on the operational requirements considered in Section 6.4. The analyzed scenarios are summarized in Table 6.2. The resulting monthly solution costs per scenario are represented in Table 6.3. All costs refer to the MOB100 dataset. Periodic scenarios (S2 and S3) refer to analysis carried out daily, i.e., 30 times per month.

#### S1 – Continuous ingestion – continuous consumption and analysis

In this scenario, we consider the case in which we require a continuous analytics solution. The whole infrastructure must be continuously up and running to support the ingestion, consumption and analysis phases. We can compute a monthly solution cost by considering the reservation price of all VMs used in the solution.

From Table 6.3, we can see the estimated monthly solution cost for scenario S1. Ingestion cost is calculated using reserved instance price, since these machines must run continuously. This is the same for analysis cost. Consumption cost is included in the ingestion, since the Kafka VMs perform both phases continuously. The single-thread cost is calculated considering configuration Natron3.

In this case, we can clearly see that the single-threaded implementation is the most cost-effective solution for the problem.

#### S2 – Continuous ingestion – periodic consumption and analysis

This scenario represents a use case where the continuous analysis is not necessary, but periodic reports are needed. Table 6.3 contains the cost analysis for this scenario. The costs of ingestion and consumption are equivalent to S1. The analysis cost is computed on the more demanding dataset MOB100, using Spark1 and Natron3 configurations. We report the monthly cost for an analysis performed daily. The ingestion phase must be continuous and, consequently, the infrastructure that support the ingestion and consumption phases can be deployed on reserved hardware. The analysis is periodic (once a day), and can be executed on pay-per-use VMs which can be turned on only for the duration of the analysis.

In this scenario, we can see that the Spark system is more cost-effective with respect to the analysis phase, but not to the ingestion phase. The cost of continuously ingesting data using a distributed cluster outvalues the benefits of processing such data in parallel. This is still true at lower data scales, where the convenience of the single-threaded solution is even more evident.

After realizing this fact we included a final scenario (S3) in our analysis. This scenario is a situation where data ingestion is provided at a fixed and small price, i.e. it does not depend on VMs cost but only on data throughput and retention. This

TABLE 6.2: Operational scenarios. Each layer of the system can run continuously (C) or periodically (P), and on shared (S) or reserved (R) hardware. Data ingestion and consumption are both handled by Apache Kafka, therefore they are always executed on the same hardware.

| Scenario | Ingestion | Consumption | Analysis |
|---|---|---|---|
| S1 | C/R | C/R | C/R |
| S2 | C/R | P/R | P/S |

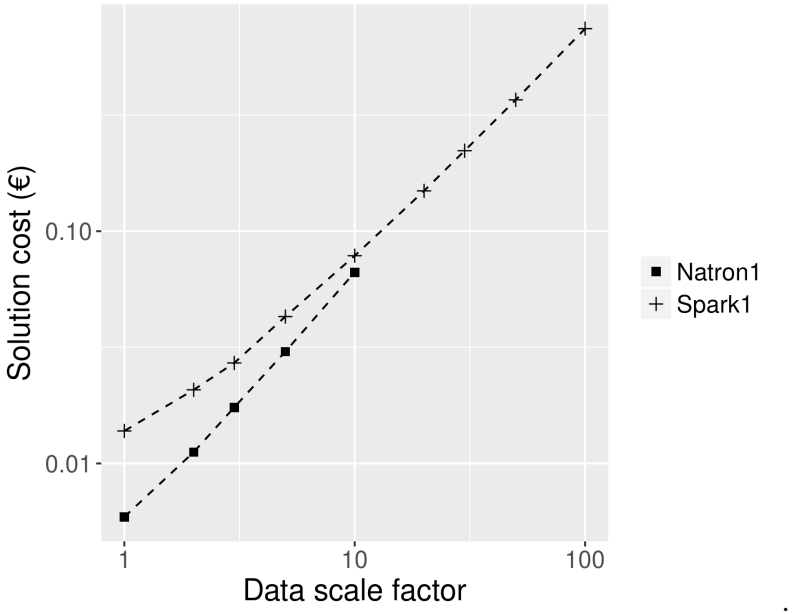


FIGURE 6.9: Total solution cost for S3: Natron1 vs Spark1. Natron1 is the lowest cost solution, but it can handle only datasets of modest size.

is the case with some particular offers from cloud providers such as Confluent[9]. Since the throughput and the retention are fixed, in this scenario the ingestion cost is the same for both solutions.

**S3 – Continuous ingestion at fixed/small price – periodic consumption and analysis**

In this scenario the total cost of the solution depends on the number of machines active during the analysis phase, and on the duration of this phase. Thus, if the additional costs of using more machines in the distributed setting implies reducing

[9] https://www.confluent.io

TABLE 6.3: Monthly solution costs. The monthly cost of our solution depending on the operational scenario. Notice that if we perform continuous ingestion, the consumption costs are included (Incl.). The third scenario represents the case in which ingestion costs are fixed, i.e. they do not depend on the number of machines, but only on data throughput. The most cost-effective solution is highlighted.

| Scenario | | Ingestion | Consum. | Analysis | Total |
|---|---|---|---|---|---|
| S1 | Spark1 | €486.32 | Incl. | €607.9 | €1094.22 |
| | Natron3 | €60.33 | Incl. | €484.92 | **€545.25** |
| S2 | Spark1 | €486.32 | Incl. | €12,41 | €498,73 |
| | Natron3 | €60.33 | Incl. | €76.85 | **€137.18** |
| S3 | Spark1 | Fixed | €9.93 | €12.41 | **€22.34** |
| | Natron3 | Fixed | €9.68 | €76.85 | €86.53 |

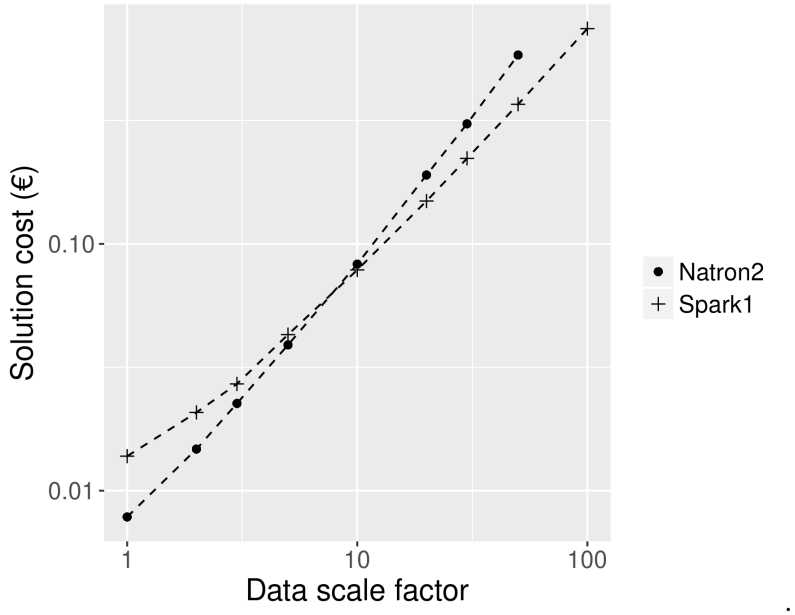


FIGURE 6.10: Total solution cost for S3: Natron2 vs Spark1. The two solutions are cost-equivalent at a scale factor around 10. After that, Spark1 becomes the most cost-effective solution.

the execution time by the same factor, then the distributed solution is the most cost-effective.

Table 6.3 presents the results for this scenario. The results compare configuration Spark1 versus configuration Natron3 when processing the dataset MOB100. We assume the analysis is carried out periodically each day. We can see that the reduced execution time for the analysis makes up for the increased number of VMs. This makes the distributed solution around 3.8 times more cost-efficient than the single-threaded system.

We provide more insight on this scenario by considering different data scales. We compare configuration Spark1 with the less expensive Natron configuration that can handle a given data scale: configuration Natron1 for a scale factor up to 10, configuration Natron2 for a scale factor up to 50, and configuration Natron3 for the dataset MOB100.

We can see that, in this setting, the most cost-effective solution depends on the data size. At small data scales, configuration Natron1 is the most cost-effective solution. The configuration Natron1 can only deal with data volumes up to scale factor 10 (city scale), but, until this point, it is more cost-effective than configuration Spark1 (Figure 6.9). When the data size increases, the solutions first become equivalent in term of cost around city scale (Figure 6.10), and, then, configuration Spark1 becomes the most cost-effective solution (Figure 6.11) when dealing with national and extreme scales.

## 6.7 Summary

Distributed systems have become widely used as data analysis tools. Those systems are designed to ease the management of a pool of resources as a single entity. This makes them scalable to massive volumes of data.

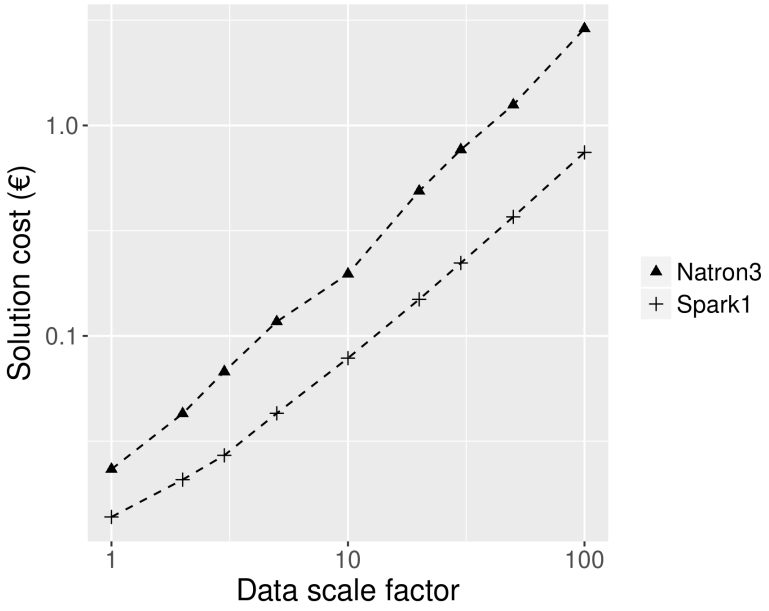


FIGURE 6.11: Total solution cost for S3: Natron3 vs Spark1. Natron3 can handle all datasets considered, however it is less cost-effective than the distributed system at all scales.

Recently,the research community showed some interest in benchmarking distributed systems against single-threaded libraries [114, 31]. The results of those works show that, in particular problem settings, distributed systems are inferior to expertly implemented single-threaded solutions.

In this chapter, we presented an experimental cost-aware comparison between two performance-equivalent solutions for a streaming anomaly detection task. The first solution is a single-threaded application based on the Natron stream processing engine, whilst the second solution is a distributed application based on Apache Spark. We based our analysis on a real industrial data analysis task, and we use the total solution cost as our metric.

Our results show that in case of continuous analysis, the single-threaded solution is the most cost-effective option.

When periodic analysis is considered, the distributed solution is the most cost-effective in analyzing the data. However, this benefit is outvalued by the costs of distributed data ingestion. Thus, the single-threaded application remains the best choice also in this case.

Finally, if we assume that data ingestion costs only depends on data throughput and retention, i.e. they are fixed and small, we show that the most cost-effective choice depends on the data size. The single-threaded application is cost-effective when managing small datasets, which is our setting are the CDRs generated by Milan when including Internet or those of the entire Italy if limiting the analysis to calls and SMSs. However, as the data size grows to the size of Italy including Internet, the distributed solution becomes the most cost-effective option.

# Chapter 7

# Scalable Unsupervised Anomaly Detection

In this chapter, we continue our analysis on the scalability of anomaly detection techniques. Distance-based algorithms are one of the most well-known techniques for anomaly detection. These algorithms have been shown to produce good accuracy in a wide range of applications. However, distance-based algorithms are limited in terms of scalability by their quadratic complexity.

To overcome this limitation, in this chapter we propose a parallel formulation of the popular KNN distance-based algorithm for unsupervised anomaly detection. More specifically we analyze the following research question: "Is it possible to scale the $k$-nearest neighbors anomaly detection algorithm to arbitrarily large data sets without significant losses in detection accuracy?".

Our formulation focused on improving scalability by analyzing in parallel different parts of the data. This finds use whenever the analyzed data does not fit comfortably into memory, or when it is distributed across several locations. To reduce synchronization latency, the proposed approach computes approximate nearest neighborhoods. We show, both theoretically and empirically, how this approach greatly improves scalability without significantly penalizing detection accuracy. To validate our approach, we perform extensive experiments on both synthetic and real-world data sets.

This work was developed at the Artificial Intelligence Center of Excellence (AICE) at F-Secure[1]. Some of the techniques presented herein have been deployed as part of an intrusion detection system at F-Secure. An extended version of this chapter has been accepted for publication in the proceedings of the 9th Workshop on Scalable Cloud Data Management.

## 7.1 Introduction

Given the large number of applications, anomaly detection algorithms can be used in environments where data velocity and volume vary widely. For high-velocity environments, several streaming anomaly detection techniques have been proposed [128, 18, 74]. However, a streaming analysis requires to continuously ingest, process and analyze data. This poses several non-trivial challenges, and it limits the adoption of the streaming approach.

A more common analysis scenario is when data is collected to be analyzed at a later time, e.g. daily. This scenario is known as batch analysis. The volume of data handled by batch processing systems can grow very rapidly. It is not uncommon for production systems to generate hundreds of gigabytes of information every day,

[1] https://www.f-secure.com/

even for modest applications. For this reason, efficient and scalable batch anomaly detection algorithms are required. Unfortunately, the scalability of anomaly detection techniques to massive data volumes still represents an open challenge.

The scarce availability of large realistic data sets, makes it so that most studies focus on modestly sized data sets [59, 37, 52]. Even in studies that focus on scalability, the data sets used in the experiments are seldom larger than one million data elements [132, 91, 139].

Among the numerous anomaly detection techniques proposed over the years [39, 3], distance-based [132] and density-based [33] techniques have stood the test of time by proving to be the most reliable in recent comparison studies [37, 69]. One limitation of distance-based algorithms is that they rely on the computation of all pair-wise distances between elements in a data set. The computational complexity of this procedure is known to be quadratic in the number of data elements. This limits the scalability of distance-based methods to huge data volumes.

In this chapter, we challenge this limitation by proposing a parallel formulation of the famous $k$-Nearest Neighbor (KNN) algorithm for distance-based anomaly detection. Our algorithm is termed partition-wise KNN (PartKNN). The $k$-Nearest Neighbor algorithm, originally proposed by Ramaswamy *et .al* [132], is one of the most commonly used distance-based algorithm in the literature. The KNN algorithm identifies outliers as isolated elements in the data set. This is done by computing the $k$-nearest neighbor distance, i.e. the Euclidean distance to the $k$-th nearest neighbor, for each element in the data set. The $k$-NN distance is then used as a measure of outlierness, e.g. the top-$n$ elements with highest $k$-NN distance are flagged as outliers. This approach rely on the fact that outliers are generally farther from their $k$-nearest neighbor than inliers, i.e. inliers belongs to clusters of size greater than or equal to $k$.

A common approach to improve the scalability of an algorithm is to divide its input into chunks, or *partitions*, and parallelize the processing of each partition. This can be by either leveraging multi-core architectures or distributing the computation on a cluster of machines. If partitions are not independent, this approach requires a synchronization step to make sure that analyzing a partition in isolation does not lead to the wrong results.

With respect to the KNN algorithm, the most expensive procedure is the $k$-nearest neighborhood search for each data element. If our goal is to compute exact nearest neighborhoods, a synchronization step is required to check whether potential neighbors are found outside of the local partition. However, the ultimate goal of the KNN algorithm is not to compute exact neighborhoods, it is instead to reliably detect outliers. For this reason, our parallel formulation drops the constraint to compute exact nearest neighborhoods. This enables us to analyze each partition independently without any synchronization overhead. This leads to massive speedups. Additionally, although our algorithm uses approximate neighborhood, we demonstrate both theoretically and empirically how this does not affect the detection accuracy of isolated data elements as outliers. This bound can be related to the data set size, and it becomes tighter as this size increases, thus making our algorithm particularly well-fit for analyzing massive data sets.

Our approach was motivated by the need to deploy a batch anomaly detection system in production at F-Secure[2]. F-Secure systems generate hundreds of millions of daily events which need to be analyzed in a timely fashion. To the best of our

[2]https://www.f-secure.com/

knowledge, these demanding requirements were not satisfied by any of the approaches proposed in the literature. Some of the results of this study have been deployed as a data analysis pipeline in production at F-Secure, handling millions of data elements per week.

This chapter is structured as follows. Section 7.2 describes distance-based anomaly detection algorithms and the KNN algorithm in particular. Section 7.3 present our partition-wise formulation of the KNN algorithm, as well as its theoretical properties, described in Section 7.3.1. In Section 7.4, we empirically evaluate the proposed algorithm in terms of both detection accuracy and scalability. Finally, Section 7.5 summarizes the chapter.

## 7.2 Distance-based anomaly detection

Distance- and density-based anomaly detection algorithms have been shown to consistently outperform other techniques in recent comparison studies [37, 69]. The key idea of these algorithms is to detect outliers by analyzing the distribution of distances between elements in a data set. Intuitively, data elements *isolated* from the majority of the data are considered as outliers. Several ways have been proposed to formally define the concept of isolation.

Distance-based algorithms were initially proposed by Knorr *et al.* [90] at the beginning of the 21st century. However, their outlier definition relies on the knowledge of structural properties of the data set, which are seldom known in practice, making their approach difficult to apply. This lead to the development of more practical outlier definitions. One of the most famous anomaly detection algorithm, the Local Outlier Factor (LOF) [33], build on the concept of distances between data elements by definition the first density-based anomaly detection algorithm. Related to distance, density refers to the number of elements belonging to the local neighborhood of an elements. Defining the neighborhood relies on a parameter $k$, representing the number of neighborhood element to consider.

A similar approach was proposed with the $k$-Nearest Neighbors algorithm [132]. Contrary to LOF, which defines the measure of outlierness as a function of the distances within a data element's neighborhood, the KNN algorithm simplifies the definition by simply assigning as outlier score the $k$-NN distance. This simple approach turned out to be extremely successful in practice, and, two decades later, it is still one of the most consistently performing algorithms across comparison studies. For a more in-depth discussion on distance- and density-based algorithms see Schubert *et al.* [141].

### 7.2.1 The KNN algorithm

The KNN algorithm [132] is the most popular distance-based algorithm for unsupervised anomaly detection. The algorithms takes as input a numerical data set $D \subset \mathbb{R}^d$ and an integer parameter $k \in [1, n]$, where $n$ is the size of the data set. Then, it computes the $k$-nearest neighborhood for each element $x \in D$. Finally, the each element is assigned an outlier score corresponding to the Euclidean distance to its $k$-th nearest neighbor.

Some variations of this base algorithm have been proposed. For example, one variation considers as outlier score the mean distance, or the sum of all distances [12], instead of the largest distance.

The KNN algorithm identifies outliers as isolated elements in a data set. More formally, outliers are those elements whose $k$-nearest neighbor distance is largest, with respect to other data elements. This formal outlier definition is not unique in the literature. In fact, it is common practice for each anomaly detection technique to propose a different definition of outlier [33, 132, 124, 108]. This is due to the fact that there does not exist an universal way to define outliers, as there does not exist an universal way to define an unusual event [65]. Moreover, there is no experimental evidence that a particular outlier definition is better than another in a given context.

For the KNN algorithm, the fact that the distance to the $k$-th nearest neighbor is taken as outlier score is quite arbitrary. Therefore, the success of the KNN algorithm might not depend on computing $k$-th nearest neighborhoods exactly. In fact, as will be demonstrated in the following, computing exact nearest neighborhoods is not necessary to obtain good detection accuracy. To explain this fact we must consider how the KNN algorithm works. By using the distance to the $k$-nearest neighbor as outlier score, we are assuming that for the majority of inliers, there exist at least $k$ other data elements belonging to a dense neighborhood around the inlier. We are also assuming that the opposite is true for the majority of outliers. If these assumptions do not hold, the KNN algorithm fails at correctly discriminating outliers from inliers.

In the following section, we will discuss how to make use of this insight to improve the scalability of the KNN algorithm while maintaining good detection accuracy.

## 7.3 Partition-wise KNN

The complexity of the KNN algorithm is lower-bounded by the complexity of the nearest neighborhood search. Therefore, the KNN algorithm has a quadratic complexity.

However, one important thing to notice is that exactly computing nearest neighborhoods is not necessary for the accuracy of the KNN algorithm. In fact, as long as for the most inliers, their $k$-th nearest neighbor is approximately closer than that of most outliers, the algorithm will maintain a good detection accuracy. This consideration was the main motivation for the main contribution of this work, the partition-wise KNN algorithm (PartKNN).

PartKNN is a parallel formulation of the KNN algorithm. PartKNN aims at maximizing the scalability of KNN while maintaining high detection accuracy. In order to make KNN scalable, our formulation divides the data set into a given number of partitions. Then, it analyzes each partition in parallel.

The original KNN algorithm assumes that exact $k$-nearest neighborhood are computed for each data element. In order to parallelize this procedure, an expensive synchronization step is required to make sure that candidate nearest neighbors do not exist outside of the local partition. However, the primary goal of KNN is not to compute neighborhoods, but the reliably detect outliers. This goal can be achieved even without considering exact nearest neighborhoods, as long as enough neighbors for each inlier are included in each partition. The PartKNN algorithm avoids the overhead of synchronization by only considering nearest neighborhoods only within a given partition.

The PartKNN algorithm is depicted in Algorithm 7.1. The algorithm works as follows: given a numerical data set $D \subset \mathbb{R}^d$, a positive integer $k \in [1, n]$, where $n = |D|$, and a positive integer $p \geq 0$ representing the total number of partitions, do:

1. For each data element $x \in D$, assign a partition number $p_x$ to $x$ uniformly at random, where $p_x \in [0, p)$.
2. Then, for each $p_i \in [0, p)$, consider the partition $P_i = \{x \in D | p_x = p_i\}$.
3. For each data element in $x \in P_i$ compute its outlier score $s_x$ using the KNN algorithm on $P_i$.

This simple formulation has several advantages. Sampling data uniformly at random guarantees that local differences in data distribution in the original dataset are preserved in each partition, assuming that each partition contains enough data elements. Notice that this is expected since our formulation is designed to handle massive data sets, thus we can assume that $p \ll n$.

This formulation allows us to also provide some insight on the expected results of PartKNN with respect to the base KNN algorithm. In particular we can compute the probability of KNN and PartKNN to produce similar results, as demonstrated in the following section.

**Algorithm 7.1** PartKNN algorithm.

**Require:** data set $D$, number of partitions $p$, neighborhood size $k$.
  **for** $x \in D$ **do**
    $p_x \leftarrow uniform(0, p)$
  **end for**
  **for** $0 \leq p_i < p$ **do**
    $P_i \leftarrow \{x \in D | p_x = p_i\}$
    **for** $x \in P_i$ **do**
      $s_x \leftarrow KNN(x, P_i, k)$
    **end for**
  **end for**

### 7.3.1 On the quality of the approximation

The PartKNN algorithm partitions the original data set into $p$ partitions. This might cause issues since an inlier might be end up isolated in a given partition. To analyze the expected occurrence of this situation let us assume, as it is often the case, that outliers are rare and isolated occurrences, while inliers have common properties, and are therefore clustered and close by to other inliers. Notice that if this assumption does not hold, distance-based algorithms, by their nature, cannot discriminate properly between outliers and inliers

As already mentioned, the PartKNN algorithm partition the data set uniformly at random. With this procedure, we are guaranteed that outliers, i.e. isolated element in the original data space, are still isolated in their respective partitions. On the other hand, inliers might loose relevant neighbors in their partition, and therefore they might be erroneously considered as outliers.

Let us assume that we subsample the original data set into a set of $p$ partitions. We denote with $P$ the set of all partitions. Each partition contains approximately $n/p$ elements, selected uniformly at random. The probability of an element $x \in D$ to end up in a given partition $\hat{P} \in P$ is:

$$\mathbb{P}(x \in \hat{P}) = \frac{1}{p} \tag{7.1}$$

Now, let $kNN_x \subset D$ denote the set of the $k$ nearest neighbors of $x$ in the original data set. The probability of all elements $y \in kNN_x$ of ending up in a given partition $\hat{P}$ is:

$$\mathbb{P}(\forall y \in kNN_x, y \in \hat{P}) = \frac{1}{p^k} \tag{7.2}$$

Thus, the chance of our partitioning scheme to preserve exact neighborhood is quite small.

However, considering the fact that our goal is not to compute neighborhoods, but to identify outliers, we might be content as long as the subsampling process does not "dilutes" too much the nature of data elements; outliers should remain isolated while inliers should remain clustered.

It is trivial to notice that isolated elements in the original data set can only become "more isolated" by removing elements with subsampling. With respect to inliers, we can demonstrate that the subsampling procedure should not cause excessive disruption in the results of the original KNN algorithm.

To demonstrate this, consider an inlier $x_i \in D$. We assume that $x_i$ is similar to other inliers with respect to the features we are considering, thus we can assume that $x_i$ belongs to some form of data cluster $C \subset D$. Let us denote with $c$ the size of this cluster, i.e. $c = |C|$.

We have already seen that the probability that all exact $k$ nearest neighbors of $x_i$ are included in the same partition as $x_i$ is small. However, let us compute the probability that at least $k$ elements from $C$ are included in a given partition $\hat{P}$. Since the KNN algorithm considers only the $k$ closest elements, as long as at least $k$ elements of the cluster are preserved inside a given partition, the algorithm should behave consistently with respect to considering the whole data set.

The probability that at least $k$ elements from $C$ are included in a given partition $\hat{P}$ is:

$$\mathbb{P}(|\{y \in \hat{P} | y \in C\}| > k) \geq 1 - \left(\frac{p-1}{p}\right)^{c-k} \tag{7.3}$$

To provide some context to this finding, let us consider a large data set. It is common to assume that outliers represent less than 10% of the data set, and we can reasonably imagine that the smallest inlier cluster must have at least more elements than there are outliers in the data. Considering also that it is common to consider values of $k \leq 100$ for the KNN algorithm. Thus, we can assume that $c \gg k$.

Plugging in some numbers in the above equation, e.g. a data set of $150K$ elements, where the smallest cluster has size $10K$, 1000 partitions, and considering $k$ as high as $k = 1000$, the above probability becomes:

$$\begin{aligned}\mathbb{P}(|\{y \in \hat{P} | y \in C\}| > k) &\geq 1 - \left(\frac{1000-1}{1000}\right)^{10K-1K} \\ &= 0.9998\end{aligned}$$

Thus, we have very high confidence that at least $k$ points in the cluster will be preserved by partitioning, this implies that inliers will approximately maintain their clustered property. Most importantly, notice that this confidence increases with the data set size, thus we expect PartKNN to provide detection accuracy similar to KNN as the size of the data set grows. This theoretical insight is validated in the following section.

TABLE 7.1: Real-world data sets characteristics

| Name | Size | Features | Outliers (%) |
|---|---|---|---|
| shuttle | 1'013 | 9 | 1.30 |
| annthyroid | 7'129 | 21 | 7.49 |
| pendigits | 9'868 | 16 | 0.20 |
| kdd99 | 48'133 | 40 | 0.41 |

## 7.4 Experimental evaluation

In this section, we evaluate the PartKNN algorithm with respect to both detection accuracy and scalability.

Detection accuracy is validated by comparing the area under the ROC curve (ROC AUC) [76] of PartKNN with the one of the original KNN algorithm. Scalability is validated by comparing execution times on large synthetic data sets.

To evaluate the detection accuracy of our proposed approach we selected four real-world benchmark data sets commonly used in the literature.

- **annthyroid -** this data set contains medical cases of normal and abnormal thyroid conditions. Abnormal conditions are labeled as outliers.
- **kdd99 -** this data set contains normal traffic and intrusions in a computer network. Intrusions are labeled as outliers. This data set was derived from the KDD 1999 Big Data Challenge data[3], which in turn was derived from the DARPA 1998 intrusion detection data set [106].
- **pendigits -** this data set contains handwritten digits. Badly written digits are labeled as outliers.
- **shuttle -** this data set contains measurements from the space shuttle sensors. Abnormal readings are labeled as outliers.

The characteristics of each benchmark are summarized in Table 7.1. The exact benchmark data sets used in the experiments can be also found in the our online repository[4].

For the scalability experiments, a collection of synthetic data sets was generated. Each data set consists of a random number of clusters sampled from a multivariate Gaussian distribution with uniformly sampled random mean and covariance matrix. These clusters corresponds to inliers. All data points are sampled within an hypercube centered in the origin with a side of 50. Outliers are generated by sampling uniformly at random within this hypercube. The percentage of outliers is selected uniformly at random between 2% and 7%.

All experiments were executed on a workstation featuring an Intel(R) Core(TM) i7-4810MQ, 2.80GHz processor and 16GB of RAM running Manjaro Linux. All presented algorithm were implemented in Python. Unless otherwise noted, the number of partitions was set to the number of available CPU cores, i.e. 8.

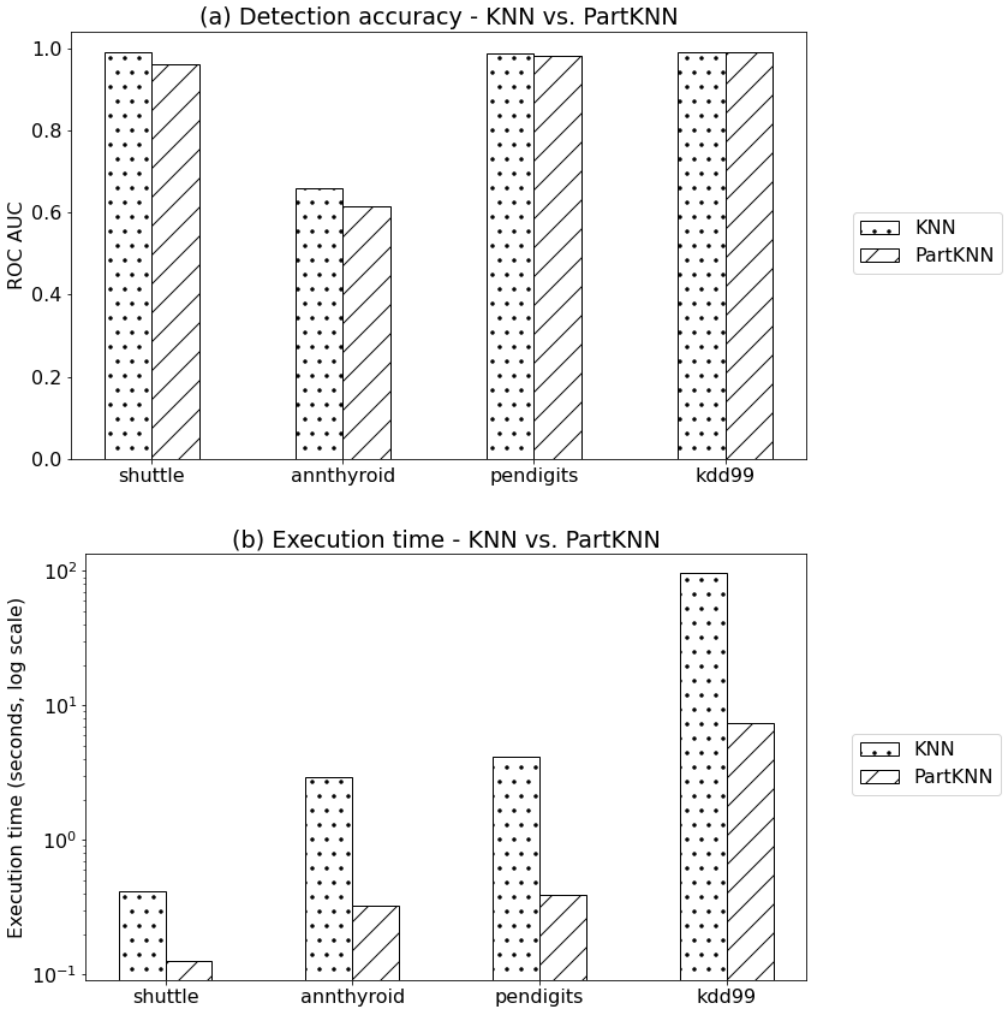


FIGURE 7.1: ROC AUC (a) and execution time (b) for KNN and PartKNN. Data sets are ordered by cardinality (increasing from left to right). PartKNN present similar accuracy to KNN but much lower execution times (around one order of magnitude).

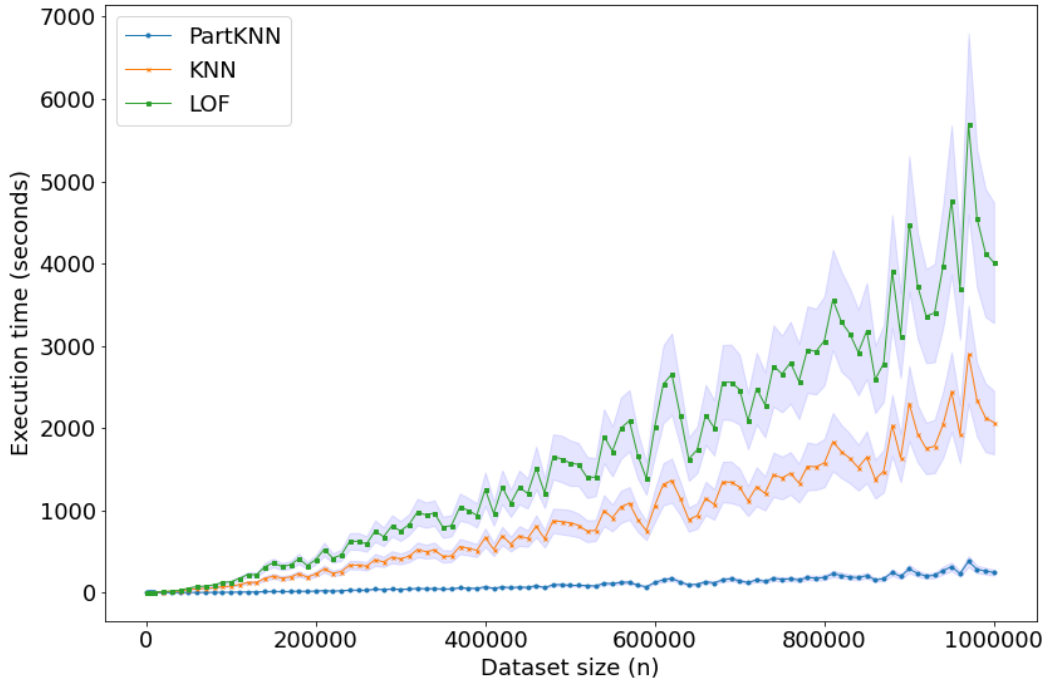


FIGURE 7.2: Scalability of PartKNN with respect to KNN and LOF. The execution time is averaged over five runs. The grey area represents the standard deviation across different runs. PartKNN clearly outperforms both KNN and LOF with respect to scalability.

### 7.4.1 Detection accuracy

The results of detection accuracy experiments are depicted in Figure 7.1. We report both detection accuracy (Figure 7.1.a) and execution time (Figure 7.1.b) for each considered benchmark.

Detection accuracy corresponds to the maximum ROC AUC with $k \in [1, 100]$. Execution time is averaged over all $k \in [1, 100]$ Benchmarks are ordered by increasing data set size from left to right.

The figure clearly depicts the similarity in accuracy between the PartKNN and KNN algorithms. A small difference can be noticed for the smaller *shuttle* data set. This is due to the small size of the data, only 1013 data elements, which imply that partitions contain too few elements to correctly characterize outliers. As expected, the results improve as the data set size grows, so much so that there is practically no difference between the two algorithms on the larger *kdd99* dataset, with around 48K elements.

With respect to execution times, the PartKNN outperforms the original KNN algorithm by one order of magnitude for all considered data sets (notice the log scale). This is expected, since the partitioning approach introduces an expected speedup of $p$, where $p$ is the number of partitions.

### 7.4.2 Scalability

To evaluate the scalability of the PartKNN algorithm we compare it to both KNN and LOF. We use data sets ranging from 10K to 1M elements. Scalability results are depicted in Figure 7.2.

For each data set, we report the average execution time over fine runs with $k$ ranging from 20 to 100. We also report the standard deviation, depicted as a confidence interval. As expected, the figure clearly depicts the better scalability of the PartKNN algorithm with respect to both KNN and LOF. This demonstrates the ability of PartKNN to scale to multi-million data sets. This ability was also demonstrated by analyzing massive production data sets at F-Secure.

## 7.5 Summary

In this chapter, we presented a novel approach for scaling the popular KNN algorithm to massive data sets. The scalability of KNN is limited by the quadratic complexity of the exact nearest neighborhood search. Our approach leverages parallel processing to greatly speedup the computation. The expensive synchronization step required to compute exact nearest neighborhood can be avoided by considering the clustered property of inliers in large data sets. This consideration also provides a theoretical bound on the quality of the algorithm with respect to the original KNN formulation.

We also presented an extensive experimental evaluation of the proposed approach. Our results demonstrate that the PartKNN algorithm clearly outperforms the considered baseline both with respect to scalability and detection accuracy. In particular, when compared with the original KNN formulation, PartKNN achieves similar accuracy with an execution time an order of magnitude smaller.

Some of the techniques presented in this study have been deployed and evaluated as part of an intrusion detection system at F-Secure. The evaluation was carried out by security analysts. The approach showed good performance with respect to both accuracy and scalability.

[3]https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[4]https://github.com/passiv-me/bad-framework

# Part IV

# Gravity-based Anomaly Detection

# Chapter 8

# Gravity-based Anomaly Detection

In this chapter, we present the final original contribution described in this thesis, the Gravity anomaly detection algorithm. Gravity is an unsupervised anomaly detection algorithm taking inspiration from the law of universal gravitation. More specifically, in this chapter we consider the following research question: "Is it possible to detect outliers in a data set by considering massive data elements and using attractive and repulsive forces as an outlying criterion?". Gravity identifies outliers by assigning a mass to each data element and defining attraction forces between them. Elements that are close enough collapse onto each other to form gravitational clusters, or *centroids*. An anomaly score is then computed for each element depending on it either belonging to a cluster or being isolated.

Our results show that Gravity is competitive with respect to other state-of-the-art anomaly detection algorithms. Morever, the properties of its hyperparameter $g$, i.e. the gravitational constant, make it easier to find an optimal hyperparameter configuration. This makes Gravity a good candidate for situations where other algorithms require large hyperparameter searches.

## 8.1 Introduction

As discussed throughout this thesis, outliers can be defined in many ways. In Section 2.1.4, we introduced some of the most popular anomaly definitions found in the literature. These include defining anomalies with respect to their distance to other data elements [33, 132], as elements belonging to low density regions of the data distribution [161, 137, 135], or as elements having high residuals with respect to a predictive model [79, 71].

Distance-based definitions have been shown to outperform other algorithms in recent comparison studies [37, 69]. However, there is no theoretical evidence that a given definition is superior to another. Different definitions rely on different assumptions on the nature of outliers, e.g. outliers correspond to isolated elements. These assumptions depend on the nature of the data under consideration. It is thus difficult to define apriori which definition is better suited for a particular data set.

It is well-known that two massive bodies, $x$ and $y$, exert an attraction force with respect to each other corresponding to the expression

$$f_a(x,y) = g\frac{m_x m_y}{d(x,y)^2} \tag{8.1}$$

where $m_x$ and $m_y$ are the masses of the two bodies, $d(x,y)$ is the distance between them and $g$ is the gravitational constant.

Gravity causes massive elements to interact. If these elements are in movement, gravity is responsible for their orbital interaction. On the other hand, if elements are too close or too slow, gravity causes them to collapse onto each other.

If is not difficult to imagine elements in an $n$-dimensional data set as celestial bodies. If $n = 3$, this similarity is even more obvious. Data elements are defined by their position in the feature space. However, contrary to celestial bodies, there is no concept of mass in a data set.

In this work, we investigate how we can use ideas from gravitation to detect outliers. Outliers usually correspond to isolated elements in a data set. Assuming it is possible to define the concept of gravity in a data set, it might be possible to detect outliers by analyzing elements with low "gravitational" interactions with respect to the rest of the data. In this chapter, we develop these ideas into a novel algorithm for unsupervised anomaly detection. We call this algorithm Gravity.

Gravity can be classified as a distance-based algorithm. However, contrary to other popular approaches [33, 132], it does not explicitly rely on computing $k$-nearest neighborhoods for each element in the data set. Instead, Gravity uses the concepts of distance and mass to define an attractive force between data elements. This force is defined using an expression similar to Equation 8.1. If the attractive force is above a given threshold, Gravity collapses data elements and substitutes them with a new element, referred to as a *centroid*, corresponding to their center of mass. Once the centroids have been computed, data elements are assigned an anomaly score according to their closest centroid. Gravity relies on the definition of the gravitational constant $g$ as its only hyperparameter.

Our experimental evaluation shows that Gravity is competitive with respect to popular state-of-the-art anomaly detection algorithms. Moreover, the properties of Gravity makes it so that the optimal value for $g$ is necessarily bounded for each data set. This eases the effort required for hyperparameter tuning with respect to other distance-based algorithms, e.g. finding the optimal value of $k$ for LOF [33]. This property makes Gravity a good fit for cases where extensive hyperparameter tuning is unfeasible.

This chapter is structured as follows. In Section 8.2, we present our proposed Gravity algorithm in details. In Section 8.2.1, we describe the properties of the hyperparameter $g$ and provide some empirical evidence of these properties. In Section 8.3, we compare Gravity to other state-of-the-art techniques with respect to detection accuracy using a multi-metric approach. Finally, Section 8.4 summarizes the chapter.

## 8.2 The Gravity algorithm

This section presents the Gravity algorithm for unsupervised anomaly detection. Gravity define outliers as data elements with weak interactions with other elements. Element interactions are defined following an expression similar to the law of universal gravitation between massive bodies.

We define the attraction force between two elements $x$ and $y \in D$, as

$$f_a(x, y) = \frac{g}{dist(x, y)} \tag{8.2}$$

where $g$ is a user-defined hyperparameter, referred to as the *gravitational constant*. Notice that Equation 8.2 resembles the law of universal gravitation, if we assume that each data element in $D$ has unit mass, and we avoid the quadratic relationship

**Algorithm 8.1** Gravity - Centroid search

**Require:** data set $D$, gravitational constant $g$
  $D \leftarrow$ standard_scaler(D)
  $C \leftarrow \{\}$
  **for** $x, y \in D, x \neq y$ **do**
    $d_{xy} \leftarrow \text{dist}(x, y)$
  **end for**
  **for** $x \in D$ **do**
    $c_x \leftarrow \text{find\_centroid}(x)$
    $C \leftarrow C \cup c_x$
  **end for**

with respect to distance. These assumptions simplify the computation without affecting the main ideas behind the algorithm.

One important thing to notice is that elements in a data set are static, i.e their position does not change in time. Therefore, if we consider the attraction force as the only interaction, all elements would naturally collapse into a single centroid. This centroid would end up in the center of mass of the whole data set. This does not happen in the physical world, since celestial entities are not static and move with respect to each other. In order to avoid all elements collapsing, we introduce the concept of constant repulsion force between elements, $r$. Since for each value of $\hat{r}$ of $r$ there exists a value of $g$ which produces the same algorithm results, $r$ is considered as an internal parameter of Gravity.

Depending on the magnitude of the attractive forces between them, elements can either be unaffected or they can collapse onto each other. The algorithm models this idea by introducing the concept of a *centroid*. A centroid represents either an isolated data element or a group of collapsed data elements. Each centroid is identified by a position and an integer size. Isolated centroids have position equal to that of the data element they represent and size equal to one. Collapsed centroids have position equal to the mean of the positions of the collapsed elements and size equal to the number of collapsed elements.

Gravity consists of two main procedures: a centroid search and an element scoring procedure. The goal of the centroid search is to find a suitable set of centroids representing the structure of the data set. The centroid search takes as input the data set $D$ and a positive real-valued hyperparameter $g$, and produces as output a set of centroids. This procedure is described in Algorithm 8.1. Initially, the data set is scaled to have zero mean and unit variance. This limits the hyperparameter search space for $g$, as described in Section 8.2.1. Then, we compute all pairwise distances in the data sets. The distance matrix is used to find all centroids in the data set. Each data element $x$ is considered to be part of a centroid. If $x$ is distant from other elements, then for each $y \in D, y \neq x$ we have $f_a(x, y) \leq r$. This causes $x$ to generate a isolated centroid. Otherwise, i.e. if there exists $y \in D, y \neq x$ such that $f_a(x, y) > r$, we create a centroid by merging $x$ with all such elements. This procedure is repeated for each element of the data set until all elements belong to centroids, either isolated or collapsed. Depending on the value of $g$, several scenarios are possible. If $g$ is too small, no data elements collapse onto each other, therefore the procedure generate one centroid for each data element. Naturally, this scenario does not provide any value with respect to the anomaly detection task. On the opposite spectrum, i.e. when $g$ is too large, all elements collapse into a single centroid corresponding to the center of mass of the data set. We discuss these scenarios in more details in

Section 8.2.1.

The second procedure in Gravity is the scoring procedure, depicted in Algorithm 8.2. The scoring procedure takes as input the precomputed set of centroids $C$, the data set $D$, and outputs an anomaly score for each $x \in D$. The scoring procedure depends on which kind of centroid element $x$ belongs to. If $x$ belongs to an isolated centroid, $x$ is assigned a larger score, corresponding to the number of centroids in the data set. If $x$ belongs to a collapsed centroid, $x$ is assigned a score proportional to its distance to the centroid center of mass and inversely proportional to the number of elements in the centroid. This represents the fact that collapsed centroids represents dense region in the feature space. Intuitively, the denser the region the less likely it is for an element belonging to it to be an outlier. Following the same reasoning, elements closest to the center of a dense region are less likely to be outliers. Notice that the results of an anomaly detection algorithm are invariant with respect to scaling or translation of the scores. In other words, what matters for the success of the algorithm is that outliers are scored higher than inliers, independently on the actual score values. Thus, selecting the number of centroids as the score value for isolated centroids does not affect the results of the algorithm In Section 8.3, we empirically evaluate our approach.

**Algorithm 8.2** Gravity - Scoring procedure

**Require:** data set D, centroid list $C$
  **for** $x \in D$ **do**
    $c_x \leftarrow$ find_closest_centroid(x, C)
    **if** $|c_x| = 1$ **then**
      $s_x \leftarrow |C|$
    **else**
      $s_x \leftarrow \text{dist}(x, c_x)/|c_x|$
    **end if**
  **end for**

### 8.2.1 Finding the gravitational constant

The gravitational constant is a physical constant which can be defined empirically. In our algorithm, we also use a gravitational constant, modeled by the hyperparameter $g$. Contrary the the physical constant, the hyperparameter $g$ needs to be fitted on the structure of the data set. This is required in order for Gravity to detect outlier correctly.

Notice that the dependency on an hyperparameter is very common among anomaly detection algorithms [33, 132, 108]. For example, most distance-based algorithms require the hyperparameter $k$, corresponding to the nearest neighborhood size, to be set by the user. This hyperparameter is defined between one and the size of the data set $n = |D|$. If the data set is large, finding the optimal value for $k$ using a grid search might be unfeasible. Contrary to $k$, the hyperparameter $g$ has an important property which avoids this limitation.

As already mentioned, depending on the value of $g$ two extreme cases are possible:

1. all centroids are isolated centroids.

2. all elements collapse into one centroid.

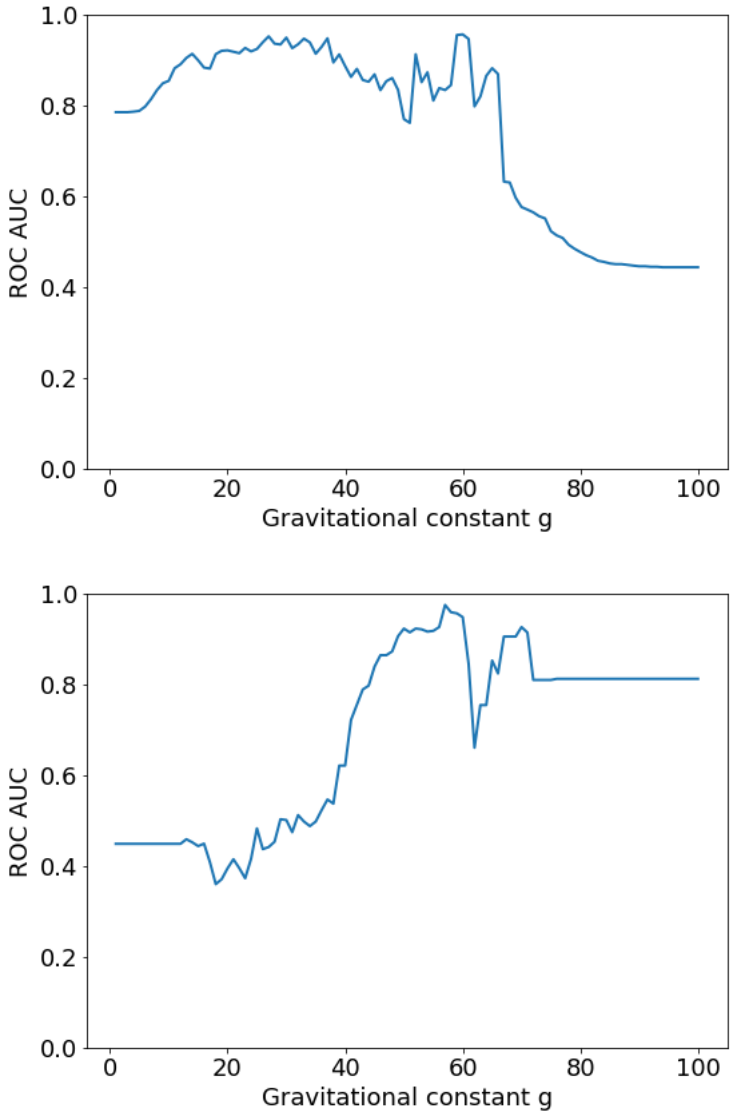


FIGURE 8.1: Hyperparameter search for $g$. Data set pendigits (top) and wine (bottom). The optimal value for $g$ is found using the same search space for both data sets, even though these have very different characteristics.

TABLE 8.1: Data sets used in the experiments.

| Data set | Data elements | Features | Outliers (%) |
|---|---|---|---|
| annthyroid | 7129 | 21 | 7.49 |
| glass | 213 | 9 | 4.22 |
| kdd99 | 48133 | 40 | 0.42 |
| pendigits | 6870 | 16 | 2.27 |
| shuttle | 1013 | 9 | 1.28 |
| wbc | 377 | 30 | 5.3 |
| wine | 129 | 13 | 7.7 |

In fact, it is easy to see that for each data set $D$, there exist two values $g_m$ and $g_M$, with $0 < g_m \leq g_M$, such that if $g \leq g_m$ scenario (1) happens, and if $g \geq g_M$ scenario two happens. Naturally, these two scenarios are not interesting for the anomaly detection task, since in scenario (1) all elements are assigned the same score, while in scenario (2) the algorithm identifies as outliers the elements at the limits of the feature space.

The existence of $g_m$ and $g_M$ is important since it restricts the search space for tuning the $g$ hyperparameter. More specifically, once we identify $g_m$ and $g_M$, the optimal value for $g$ must necessarily lie between $g_m$ and $g_M$. Examples of this behavior are depicted in Figure 8.1. It is easy to notice that for the pendigits data set (Figure 8.1 top) $g_m$ is approximately equal to 8 and $g_M$ is around 82. For the wbc data set (Figure 8.1 bottom) $g_m$ is around 15 and $g_M$ is around 70. Notice that pendigits and wbc have very different characteristics (see Table 8.1). Figure 8.1 demonstrates how the optimal value for $g$ does not seem to depend on the data sets characteristics. This is not the case other algorithms' hyperparameters, such as $k$, for which there are no theoretical bounds on the optimal value. Scaling the data set, as described in Algorithm 8.1 makes it even easier to find appropriate values for $g_m$ and $g_M$.

## 8.3 Experimental evaluation

In this section, we present an experimental evaluation for the proposed Gravity algorithm. The evaluation is conducted using the BAD framework, described in Chapter 5.

TABLE 8.2: Detection accuracy of Gravity with respect to the ROC AUC metric (ROC AUC). The best results for each data set are highlighted in bold. Gravity shows the best performance on shuttle and is within 2% of the best performance for the majority of the considered benchmarks.

| **Data set** | **Gravity** | **LOF** | **KNN** | **IForest** | **OCSVM** |
|---|---|---|---|---|---|
| annthyroid | 0.646 | **0.674** | 0.659 | 0.661 | 0.483 |
| glass | 0.726 | 0.824 | 0.869 | 0.711 | 0.462 |
| kdd99 | 0.968 | 0.917 | **0.987** | 0.881 | 0.318 |
| pendigits | 0.958 | 0.917 | **0.987** | 0.881 | 0.318 |
| shuttle | **0.996** | 0.989 | 0.989 | 0.88 | 0.731 |
| wbc | 0.932 | **0.954** | **0.954** | 0.952 | 0.92 |
| wine | 0.976 | **1.0** | 0.999 | 0.816 | 0.681 |

TABLE 8.3: Detection accuracy of Gravity with respect to the average precision score metrics (AP). The best results for each data set are highlighted in bold. Gravity shows the best performance on annthyroid, pendigits and shuttle. Higher values for AP indicate an highest resiliency to false positives.

| Data set | Gravity | LOF | KNN | IForest | OCSVM |
|---|---|---|---|---|---|
| annthyroid | **0.144** | 0.124 | 0.121 | 0.143 | 0.071 |
| glass | 0.103 | **0.207** | 0.206 | 0.108 | 0.201 |
| pendigits | **0.163** | 0.015 | 0.072 | 0.009 | 0.002 |
| shuttle | **0.586** | 0.378 | 0.386 | 0.099 | 0.048 |
| wbc | 0.435 | 0.583 | 0.601 | **0.652** | 0.552 |
| wine | 0.642 | **1.0** | 0.991 | 0.235 | 0.22 |

We evaluate gravity in terms of detection accuracy. We consider eight benchmarks from the literature. The benchmarks are described in Table 8.1. We compare Gravity with respect to four baseline algorithms from the literature. As our baselines, we consider the LOF [33], KNN [132], IForest [108] and OCSVM [137] algorithms. In the comparison, we follow a multi-metric approach. In particular, we focus on both the area under the ROC curve (ROC AUC) and the average precision score (AP) [49].

For each algorithm and data set, we perform an extensive grid search to find an appropriate hyperparameter configuration. The reported metrics correspond to the optimal hyperparameter value found during the search.

Results for the ROC AUC metric are depicted in Table 8.2. The highest metric value for each data set is depicted in bold. When considering ROC AUC, Gravity shows performance comparable with the current state of the art. Gravity obtains the best performance on the shuttle data set. Moreover, Gravity is within a couple points with respect to the best algorithm with respect to all other considered benchmarks, except for glass. Considering the hyperparameter properties described in Section 8.2.1, this performance is not to be disregarded as disappointing. In fact, we argue that this fact makes Gravity a good candidate for applications where hyperparameter tuning is particularly expensive.

This is also demonstrated by the fact that we used the same grid search strategy, i.e. $g \in [1.0, 100.0]$, for Gravity on all data sets. This approach is not feasible for other algorithms since the optimal hyperparameter setting directly depends on the cardinality of the data set.

Table 8.3 presents our results with respect to the average precision score metric. Considering this metric, Gravity shows the best performance on the annthyroid, pendigits, and shuttle benchmarks. These results suggests that Gravity might be particularly well-fit for applications where false positive errors are particularly costly (see Section 2.1.5).

## 8.4 Summary

In this chapter, we presented our novel algorithm for anomaly detection, named Gravity. Gravity is inspired by ideas for the law of universal gravitation. In particular, Gravity defines outliers by assigning a mass to each data element, and computing attraction forces between elements. Elements with strong enough interactions are less likely to be outliers.

We defined the concept of centroid, as an abstraction used to model dense regions in the feature space. Gravity learns centroids from the data set, and later uses them to assign an anomaly score to each data element. This procedure relies on setting the $g$ hyperparameter, referred to as the gravitational constant.

Gravity can be classified as a distance-based algorithm. However, contrary to other distance-based algorithms, the optimal hyperparameter settings for $g$ doed not depend on the characteristics of the data set. This makes it possible to find the optimal hyperparameter settings for Gravity more easily than with respect to other algorithms.

To evaluate Gravity, we compared it with four state-of-the-art anomaly detection algorithms on eight benchmark data sets. Our results show that Gravity's performance is competitive with respect to the considered state-of-the-art algorithms. In particular, Gravity shows superior performance when considering the average precision score metric. This work shows the advantages of Gravity in cases where hyperparameter tuning is particularly expensive, or where a low false positive error rate is critical.

# Part V

# Conclusions

# Chapter 9

# Conclusions

In this thesis, we presented our contributions to the anomaly detection field. Anomaly detection is a challenging data analysis task with several critical applications. The development of reliable and accurate anomaly detection techniques has the potential of revolutionizing applications ranging from computer security to remote sensing, system monitoring and e-health.

We began our analysis by presenting the anomaly detection field and the current state of the art. In Chapter 2, we presented the main characteristics of the anomaly detection problem, and its differences with respect to other data analysis tasks. We also presented an in-depth review of the anomaly detection approaches proposed in the literature. In Chapter 3, we presented some of the most seminal works in the field. In particular, we illustrated the historical development of the field from the outlier identification problem in Statistics. We also described what we consider to be the most important open challenges in the field.

In Chapter 4, we presented our first original contribution to the field. Our goal with this contribution was to find the most widespread issues in the anomaly detection literature with respect to methodology, evaluation and reproducibility. With this goal in mind, we conducted a methodological survey of the most recent anomaly detection literature. We selected a collection of 760 papers on anomaly detection. These papers were selected to be representative of impactful contributions to the field. We analyzed each paper along four methodological perspectives, namely application domain, anomaly definition, data representation and evaluation methodology.

The results of our survey provide several interesting insights. Most notably, the application domains to which anomaly detection has been applied are still limited. This is in contrast with the generality of the anomaly detection task. Secondly, even though a large number of anomaly definitions exist in the literature, these can be classified using only six classes. The proposed classification is based the core assumptions underlying a given definition, and is beneficial when comparing different anomaly detection techniques. Additionally, most of the published literature deal with numerical data sets, while the more general setting of mixed-attribute data set has been only partially investigated. The evaluation of anomaly detection presents several challenges. Benchmark data sets are not used consistently across different studies. This hinders the replicability of published results. Finally, when different approaches are compared, it is common to heuristically set hyperparameters values using rules of thumb. This approach is less robust with respect to grid search hyperparameter tuning, especially in comparison studies.

We further analyzed the evaluation methodology for anomaly detection in Chapter 5. In this chapter, we proposed a novel framework, named BAD, for benchmarking anomaly detection algorithms. BAD enables the execution of massive hyperparameter searches by distributing the computation on a cluster of machines. Using BAD, we empirically demonstrated some of the evaluation issues highlighted in our

survey. We also showed how BAD is designed to solve most of these issues. In particular, we illustrated how rule-of-thumb settings are unreliable for the majority of considered benchmarks, and how a grid-search approach to hyperparameter tuning provides significant performance gains. Additionally, we demonstrated how different performance metrics and hyperparameter settings affect the relative rankings of anomaly detection algorithms in comparison studies. This result is particularly concerning, since most published research rely on rule-of-thumb settings and use the area under the ROC curve as the only efficacy metric. In fact, we demonstrated how our results diverge from other comparison studies which do not follow our best practices, while we were partially able to replicate the results of studies considering the grid-search hyperparameter tuning approach. Finally, we demonstrated the efficiency of BAD with respect to the popular ELKI framework, when considering the hyperparameter tuning task.

In the second part of the thesis, we presented our works toward solving the challenge of scaling anomaly detection techniques to massive data sets. This challenge arose in several industrial use cases we dealt with in the context of this doctoral thesis. As our first contribution in this direction, in Chapter 6 we analyzed the trade-offs between a single-threaded and a distributed solution for an anomaly detection task in term of price-performance. This type of analysis was motivated by the industrial setting, where cost is an important factor when assessing the feasibility of a project. In this work, we compared two equivalent solutions for the anomaly detection task. The first solution is a single-threaded application, whereas the second solution is distributed. Our results demonstrate that the most cost-effective solution depends on the data size, evidencing that at smaller scales, the single-threaded application is more cost-effective than the distributed solution.

In Chapter 7, we presented our approach to scaling the popular KNN anomaly detection algorithm to massive data set. The KNN algorithm's scalability is bounded by its quadratic complexity. To overcome this limitation, we proposed a parallel formulation of KNN which greatly reduces execution times on massive data sets. Contrary to other approaches, our formulation does not require a synchronization step, and it detect outliers based on approximated nearest neighborhoods. We showed both theoretically and empirically how this approach does not hinder the quality of the solution. We demonstrated both the scalability and efficacy of our proposed approach through extensive experimental evaluation on both real-world and synthetic data sets.

Finally, in Chapter 8 we presented Gravity, our novel algorithm for anomaly detection. Gravity takes inspiration from the law of universal gravitation to detect outliers based on attractive and repulsive forces between massive elements in a data set. Data elements can collapse into each other depending on the balance between attractive and repulsive forces. Elements with low attractive forces are classified as outliers. This novel outlier definition requires finding the hyperparameter $g$, corresponding to the gravitational constant. We demonstrated how this can be done efficiently for a large variety of data sets. Finally, we showed how Gravity is competitive with other state-of-the-art anomaly detection algorithms with respect to detection accuracy on a collection of benchmark data sets.

## 9.1 Limitations and future works

In this section, we describe the limiting assumptions of our contributions. This analysis highlights the strength and weaknesses of our approach, and it helps us to define interesting future works.

**A Methodological Survey of Anomaly Detection.** Our methodological anomaly detection survey, presented in Chapter 4, provides a broad view of the anomaly detection literature from a methodological perspective. One of our key assumption in this work is that the surveyed papers are representative of the anomaly detection literature as a whole. If this assumption hold, we can presume that the issues highlighted in our survey are indeed issues of anomaly detection research. The arbitrary condition on the number of citations described in Section 4.2 might invalidate our assumption. However, we experimented with different conditions and considering the large number of surveyed papers, the inclusion or exclusion of a small portion of papers from the surveyed corpus does not influence the macro analysis carried out in the survey.

One possible extension of this work could be to consider more methodological perspectives, e.g performance metrics, streaming versus batch, etc. Another interesting extension could be to survey each application domain, or anomaly definition, or data representation independently. This approach could certainly provide more detailed insights on the research methodology in one particular subfield.

**BAD: Benchmarking for Anomaly Detection.** The design and implementation of BAD required several design choices. The current version of BAD is limited to numerical data sets. Although this is a common assumption in the literature [68], it is nonetheless a limitation. Apart from the obvious improvements in software quality, BAD is also limited to static data sets, i.e. it does not provide APIs for the evaluation of streaming anomaly detection techniques. This is an interesting challenge, and it has been investigated in several studies [30, 99, 100]. One obvious extension for BAD would be to define and implement streaming evaluation APIs.

Another interesting future use of BAD could be the production of a large corpus of comparison studies. This could lead to a better understanding of the anomaly detection state of the art. Extensions to the BAD collection of benchmark data sets and algorithms are also possible. Finally, optimization-based parameter tuning approaches could also be implemented within BAD [27].

**Cost-aware Data Analysis.** In our study on cost-aware data analysis (Chapter 6), we analyzed a common data analysis task under a novel perspective. This type of analysis is interesting and much work still need to be done. Our study could be easily extended by considering different data analysis tasks and settings, e.g. streaming classification, streaming clustering, etc. This direction has already been partially investigated [114, 31]. This could further validate our results, since the analysis task influences the query's complexity, execution time and cost.

Also, the data sets considered could be varied along different dimensions. In our work, we only considered the data set size dimension. Other interesting dimensions to consider are the stream variability, the number of features, and the presence of categorical attributes. We implemented our solutions to the best of our skill. However, our system implementations might not be optimal. Another direction of

investigation to extend our work is how the system implementation affects the total solution cost. This is particularly interesting for the distributed solution, where several processing engines are available. Another extension could be to consider different hardware configurations. For example Azure offer VM series optimized for various workloads, e.g. memory-optimized, HPC-optimized. An analysis in term of cost-effectiveness of such machines could be interesting. Another interesting line of work parallel to our approach is to investigate how to choose the optimal cluster configuration. In our work, we analyzed several configurations, however the results might be different depending on the particular data analysis task.

**Scalable Unsupervised Anomaly Detection.** The parallel formulation of KNN, presented in Chapter 7, represents a step towards solving the scalability issue of anomaly detection algorithms.

Although our approach shows good results, it is limited to the KNN algorithm. Investigating similar approaches for other anomaly detection algorithms is an interesting future work. Another promising direction is to find good approximation bounds. Since anomaly detection algorithms rely on heuristic definition of outliers, relaxing this definitions might improve scalability without significantly affecting detection performance. Naturally, this type of analysis must be replicated for each anomaly definition in the literature, since results for different definitions might be difficult to generalize.

**Gravity-based Anomaly Detection.** We showed in Chapter 8 how our Gravity algorithm is competitive with respect to other state-of-the-art techniques. Nonetheless, several improvement directions could be investigated.

Being a distance-based algorithm, Gravity suffers from the same scalability issues as other algorithms in the same class. Thus, finding a good approach to scale Gravity to massive data sets is an interesting research challenge. Extensions to the base algorithm could also be investigated by analyzing its performance on a larger variety of benchmarks and use cases.

# Bibliography


[1] Georges Aad et al. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 1–29.

[2] Dimitris Achlioptas. "Database-friendly random projections: Johnson-Lindenstrauss with binary coins". In: *Journal of computer and System Sciences* 66.4 (2003), pp. 671–687.

[3] Charu C Aggarwal. "Outlier analysis". In: *Data mining*. Springer. 2015, pp. 237–263.

[4] Charu C Aggarwal. "Outlier ensembles: position paper". In: *ACM SIGKDD Explorations Newsletter* 14.2 (2013), pp. 49–58.

[5] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. "On the surprising behavior of distance metrics in high dimensional spaces". In: *ICDT*. Vol. 1. Springer. 2001, pp. 420–434.

[6] Charu C Aggarwal and Saket Sathe. *Outlier ensembles: An introduction*. 2017.

[7] Charu C Aggarwal and Saket Sathe. "Theoretical foundations and algorithms for outlier ensembles". In: *Acm Sigkdd Explorations Newsletter* 17.1 (2015), pp. 24–47.

[8] Charu C Aggarwal and Philip S Yu. "Outlier detection for high dimensional data". In: *ACM Sigmod Record*. Vol. 30. 2. ACM. 2001, pp. 37–46.

[9] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. "Ganomaly: Semi-supervised anomaly detection via adversarial training". In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 622–637.

[10] Fabrizio Angiulli and Fabio Fassetti. "Detecting distance-based outliers in streams of data". In: *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM. 2007, pp. 811–820.

[11] Fabrizio Angiulli and Clara Pizzuti. "Fast outlier detection in high dimensional spaces". In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2002, pp. 15–27.

[12] Fabrizio Angiulli and Clara Pizzuti. "Outlier mining in large high-dimensional data sets". In: *IEEE transactions on Knowledge and Data engineering* 17.2 (2005), pp. 203–215.

[13] Fabrizio Angiulli et al. "A distributed approach to detect outliers in very large data sets". In: *European Conference on Parallel Processing*. Springer, Berlin, Heidelberg. 2010, pp. 329–340.

[14] Arvind Arasu et al. "Linear road: a stream data management benchmark". In: *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment. 2004, pp. 480–491.

[15] Andrea Arcuri and Gordon Fraser. "On parameter tuning in search based software engineering". In: *International Symposium on Search Based Software Engineering*. Springer. 2011, pp. 33–47.

[16] Antonio Arranz et al. "DADICC: Intelligent system for anomaly detection in a combined cycle gas turbine plant". In: *Expert Systems with Applications* 34.4 (2008), pp. 2267–2277.

[17] R Arshady. "Suspension, emulsion, and dispersion polymerization: A methodological survey". In: *Colloid and polymer science* 270.8 (1992), pp. 717–732. DOI: 10.1007/BF00776142.

[18] Ira Assent et al. "AnyOut: Anytime Outlier Detection on Streaming Data." In: *DASFAA (1)*. 2012, pp. 228–242.

[19] Aya Ayadi et al. "Outlier detection approaches for wireless sensor networks: A survey". In: *Computer Networks* 129 (2017), pp. 319–333.

[20] Marco Balduini and Emanuele Della Valle. "FraPPE: a vocabulary to represent heterogeneous spatio-temporal data to support visual analytics". In: *International Semantic Web Conference*. Springer. 2015, pp. 321–328.

[21] Marco Balduini, Emanuele Della Valle, and Riccardo Tommasini. "SLD revolution: A cheaper, faster yet more accurate streaming linked data framework". In: *European Semantic Web Conference*. Springer. 2017, pp. 263–279.

[22] Marco Balduini, Sivam Pasupathipillai, and Emanuele Della Valle. "Cost-aware streaming data analysis: Distributed vs single-thread". In: *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*. 2018, pp. 160–170.

[23] Marco Balduini et al. "Citysensing: Fusing city data for visual storytelling". In: *IEEE MultiMedia* 22.3 (2015), pp. 44–53.

[24] Marco Balduini et al. "Social listening of city scale events using the streaming linked data framework". In: *International Semantic Web Conference*. Springer. 2013, pp. 1–16.

[25] Vic Barnett, Toby Lewis, et al. *Outliers in statistical data*. Vol. 3. 1. Wiley New York, 1994.

[26] Stephen D Bay and Mark Schwabacher. "Mining distance-based outliers in near linear time with randomization and a simple pruning rule". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2003, pp. 29–38.

[27] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 281–305.

[28] Kevin Beyer et al. "When is "nearest neighbor" meaningful?" In: *International conference on database theory*. Springer. 1999, pp. 217–235.

[29] Fábio Bezerra, Jacques Wainer, et al. "Anomaly detection algorithms in business process logs". In: *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS), volume AIDSS, Barcelona, Spain*. 2008, pp. 11–18.

[30] Albert Bifet et al. "Efficient online evaluation of big data stream classifiers". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 59–68.

[31] Christoph Boden, Tilmann Rabl, and Volker Markl. "Distributed Machine Learning-but at what COST". In: *Machine Learning Systems Workshop at the 2017 Conference on Neural Information Processing Systems*. 2017.

[32] Richard J Bolton and David J Hand. "Statistical fraud detection: A review". In: *Statistical science* (2002), pp. 235–249.

[33] Markus M Breunig et al. "LOF: Identifying Density-Based Local Outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data - SIGMOD '00* 29.2 (2000), pp. 93–104. ISSN: 0163-5808. DOI: 10.1145/342009.335388. URL: http://portal.acm.org/citation.cfm?doid=342009.335388.

[34] Francesco Calabrese et al. "Real-time urban monitoring using cell phones: A case study in Rome". In: *IEEE Transactions on Intelligent Transportation Systems* 12.1 (2011), pp. 141–151.

[35] Francesco Calabrese et al. "The geography of taste: Analyzing cell-phone mobility and social events." In: *Pervasive*. Vol. 10. Springer. 2010, pp. 22–37.

[36] Francesco Calabrese et al. "Urban computing and mobile devices". In: *IEEE Pervasive Computing* 6.3 (2007), pp. 52–57.

[37] Guilherme O Campos et al. "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study". In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 891–927.

[38] Raymond L Chambers. "Outlier robust finite population estimation". In: *Journal of the American Statistical Association* 81.396 (1986), pp. 1063–1069.

[39] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), p. 15.

[40] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[41] Jinghui Chen et al. "Outlier detection with autoencoder ensembles". In: *Proceedings of the 2017 SIAM international conference on data mining*. SIAM. 2017, pp. 90–98.

[42] Yixin Chen et al. "Outlier detection with the kernelized spatial depth function". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.2 (2009), pp. 288–305.

[43] Sanket Chintapalli et al. "Benchmarking streaming computation engines: Storm, flink and spark streaming". In: *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE. 2016, pp. 1789–1792.

[44] Joël Coste, Jacques Fermanian, and Alain Venot. "Methodological and statistical problems in the construction of composite measurement scales: a survey of six medical and epidemiological journals". In: *Statistics in medicine* 14.4 (1995), pp. 331–345. DOI: 10.1002/sim.4780140402.

[45] Alessandro D'Alconzo et al. "A distribution-based approach to anomaly detection and application to 3G mobile traffic". In: *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE. 2009, pp. 1–8.

[46] Armin Daneshpazhouh and Ashkan Sami. "Entropy-based outlier detection using semi-supervised approach with few positive examples". In: *Pattern Recognition Letters* 49 (2014), pp. 77–84.

[47] Dipankar Dasgupta et al. “Negative selection algorithm for aircraft fault detection”. In: *Artificial immune systems* (2004), pp. 1–13.

[48] Mayur Datar et al. “Locality-sensitive hashing scheme based on p-stable distributions”. In: *Proceedings of the twentieth annual symposium on Computational geometry*. ACM. 2004, pp. 253–262.

[49] Jesse Davis and Mark Goadrich. “The relationship between Precision-Recall and ROC curves”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 233–240.

[50] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[51] Emanuele Della Valle and Marco Balduini. “Listening to and visualising the pulse of our cities using Social Media and Call Data Records”. In: *International Conference on Business Information Systems*. Springer. 2015, pp. 3–14.

[52] Rémi Domingues et al. “A comparative evaluation of outlier detection algorithms: Experiments and analyses”. In: *Pattern Recognition* 74 (2018), pp. 406–421.

[53] Lian Duan et al. “Cluster-based outlier detection”. In: *Annals of Operations Research* 168.1 (2009), pp. 151–168.

[54] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[55] Nick Duffield et al. “Rule-based anomaly detection on IP flows”. In: *INFOCOM 2009, IEEE*. IEEE. 2009, pp. 424–432.

[56] William Eberle and Lawrence Holder. “Anomaly detection in data represented as graphs”. In: *Intelligent Data Analysis* 11.6 (2007), pp. 663–689. DOI: 10.5555/1368018.1368024.

[57] Hilmi E Egilmez and Antonio Ortega. “Spectral anomaly detection using graph-based filtering for wireless sensor networks”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 1085–1089.

[58] Manzoor Elahi et al. “Efficient clustering-based outlier detection algorithm for dynamic data stream”. In: *Fuzzy Systems and Knowledge Discovery, 2008. FSKD'08. Fifth International Conference on*. Vol. 5. IEEE. 2008, pp. 298–304.

[59] Andrew F Emmott et al. “Systematic construction of anomaly detection benchmarks from real data”. In: *Proceedings of the ACM SIGKDD workshop on outlier detection and description*. ACM. 2013, pp. 16–21.

[60] Sarah M Erfani et al. “High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning”. In: *Pattern Recognition* 58 (2016), pp. 121–134.

[61] Charles W Eriksen. “Discrimination and learning without awareness: a methodological survey and evaluation.” In: *Psychological review* 67.5 (1960), p. 279. DOI: 10.1037/h0041622.

[62] Eleazar Eskin. “Anomaly detection over noisy data using learned probability distributions”. In: *In Proceedings of the International Conference on Machine Learning*. Citeseer. 2000.

[63] Eleazar Eskin et al. “A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data”. In: *Applications of data mining in computer security* 6 (2002), pp. 77–102.

[64] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Kdd*. Vol. 96. 34. 1996, pp. 226–231.

[65] Vladimir Estivill-Castro. "Why so many clustering algorithms: a position paper". In: *ACM SIGKDD explorations newsletter* 4.1 (2002), pp. 65–75.

[66] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern recognition letters* 27.8 (2006), pp. 861–874.

[67] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From data mining to knowledge discovery in databases". In: *AI magazine* 17.3 (1996), pp. 37–37.

[68] Mathieu Garchery and Michael Granitzer. "On the influence of categorical features in ranking anomalies using mixed data". In: *Procedia Computer Science* 126 (2018), pp. 77–86.

[69] Markus Goldstein and Seiichi Uchida. "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data". In: *PloS one* 11.4 (2016), e0152173.

[70] Nico Görnitz et al. "Toward supervised anomaly detection". In: *Journal of Artificial Intelligence Research* 46 (2013), pp. 235–262.

[71] Aurea Grané and Helena Veiga. "Wavelet-based detection of outliers in financial time series". In: *Computational Statistics & Data Analysis* 54.11 (2010), pp. 2580–2593.

[72] Jim Gray. *Benchmark handbook: for database and transaction processing systems*. Morgan Kaufmann Publishers Inc., 1992.

[73] Frank E Grubbs. "Procedures for detecting outlying observations in samples". In: *Technometrics* 11.1 (1969), pp. 1–21.

[74] Sudipto Guha et al. "Robust random cut forest based anomaly detection on streams". In: *International Conference on Machine Learning*. 2016, pp. 2712–2721.

[75] Manish Gupta et al. "Outlier detection for temporal data: A survey". In: *IEEE Transactions on Knowledge and Data Engineering* 26.9 (2014), pp. 2250–2267.

[76] James A Hanley and Barbara J McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." In: *Radiology* 143.1 (1982), pp. 29–36.

[77] Milos Hauskrecht et al. "Outlier detection for patient monitoring and alerting". In: *Journal of biomedical informatics* 46.1 (2013), pp. 47–55.

[78] Douglas M Hawkins. *Identification of outliers*. Vol. 11. Springer, 1980.

[79] Simon Hawkins et al. "Outlier detection using replicator neural networks". In: *International Conference on Data Warehousing and Knowledge Discovery*. Springer. 2002, pp. 170–180.

[80] Victoria Hodge and Jim Austin. "A survey of outlier detection methodologies". In: *Artificial intelligence review* 22.2 (2004), pp. 85–126.

[81] Junho Hong, Chen-Ching Liu, and Manimaran Govindarasu. "Integrated anomaly detection for cyber security of the substations". In: *IEEE Transactions on Smart Grid* 5.4 (2014), pp. 1643–1653.

[82] Shin-Ying Huang, Jhe-Wei Lin, and Rua-Huan Tsaih. "Outlier detection in the concept drifting environment". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2016, pp. 31–37.

[83] Hesam Izakian and Witold Pedrycz. "Anomaly detection in time series data using a fuzzy c-means clustering". In: *2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*. IEEE. 2013, pp. 1513–1518. DOI: 10.1109/IFSA-NAFIPS.2013.6608627.

[84] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988. ISBN: 013022278X. DOI: 10.5555/46712.

[85] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. "Data clustering: a review". In: *ACM computing surveys (CSUR)* 31.3 (1999), pp. 264–323.

[86] Fabian Keller, Emmanuel Muller, and Klemens Bohm. "HiCS: High contrast subspaces for density-based outlier ranking". In: *2012 IEEE 28th international conference on data engineering*. IEEE. 2012, pp. 1037–1048.

[87] Maurice G Kendall. "A new measure of rank correlation". In: *Biometrika* 30.1/2 (1938), pp. 81–93.

[88] Safa Khazai et al. "Anomaly detection in hyperspectral images based on an adaptive support vector method". In: *IEEE Geoscience and Remote Sensing Letters* 8.4 (2011), pp. 646–650.

[89] Edwin M Knorr and Raymond T Ng. "A Unified Notion of Outliers: Properties and Computation." In: *KDD*. 1997, pp. 219–222.

[90] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. "Distance-based outliers: algorithms and applications". In: *The VLDB Journal - The International Journal on Very Large Data Bases* 8.3-4 (2000), pp. 237–253.

[91] Anna Koufakou et al. "A scalable and efficient outlier detection strategy for categorical data". In: *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*. Vol. 2. IEEE. 2007, pp. 210–217.

[92] Anna Koufakou et al. "Fast parallel outlier detection for categorical datasets using MapReduce". In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE. 2008, pp. 3298–3304.

[93] Kira Kowalska and Leto Peel. "Maritime anomaly detection using Gaussian Process active learning". In: *Information Fusion (FUSION), 2012 15th International Conference on*. IEEE. 2012, pp. 1164–1171.

[94] Jay Kreps, Neha Narkhede, Jun Rao, et al. "Kafka: A distributed messaging system for log processing". In: *Proceedings of the NetDB*. 2011, pp. 1–7.

[95] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. "The (black) art of runtime evaluation: Are we comparing algorithms or implementations?" In: *Knowledge and Information Systems* 52.2 (2017), pp. 341–378.

[96] Hans-Peter Kriegel, Arthur Zimek, et al. "Angle-based outlier detection in high-dimensional data". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 444–452.

[97] Hans-Peter Kriegel et al. "LoOP: local outlier probabilities". In: *Proceedings of the 18th ACM conference on Information and knowledge management*. 2009, pp. 1649–1652.

[98] Gautier Krings et al. "Urban gravity: a model for inter-city telecommunication flows". In: *Journal of Statistical Mechanics: Theory and Experiment* 2009.07 (2009), p. L07003.

[99] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. "Generic and scalable framework for automated time-series anomaly detection". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 1939–1947.

[100] Alexander Lavin and Subutai Ahmad. "Evaluating Real-Time Anomaly Detection Algorithms–The Numenta Anomaly Benchmark". In: *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*. IEEE. 2015, pp. 38–44.

[101] Aleksandar Lazarevic and Vipin Kumar. "Feature bagging for outlier detection". In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM. 2005, pp. 157–166.

[102] Aleksandar Lazarevic et al. "A comparative study of anomaly detection schemes in network intrusion detection". In: *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM. 2003, pp. 25–36.

[103] Xiaolei Li et al. "Temporal outlier detection in vehicle traffic data". In: *IEEE International Conference on Data Engineering*. IEEE. 2009, pp. 1319–1322.

[104] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml.

[105] Shih-Wei Lin et al. "An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection". In: *Applied Soft Computing* 12.10 (2012), pp. 3285–3290.

[106] Richard P Lippmann et al. "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation". In: *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*. Vol. 2. IEEE. 2000, pp. 12–26.

[107] Duo Liu et al. "Network traffic anomaly detection using clustering techniques and performance comparison". In: *Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on*. IEEE. 2013, pp. 1–4.

[108] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest". In: *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE. 2008, pp. 413–422.

[109] Elio Lozano and E Acufia. "Parallel algorithms for distance-based and density-based outliers". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE. 2005, 4–pp.

[110] Vijay Mahadevan et al. "Anomaly detection in crowded scenes". In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 1975–1981.

[111] Henrique O Marques et al. "On the internal evaluation of unsupervised outlier detection". In: *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*. ACM. 2015, p. 7.

[112] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co., 2015.

[113] John McHugh. "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory". In: *ACM Transactions on Information and System Security (TISSEC)* 3.4 (2000), pp. 262–294.

[114] Frank McSherry, Michael Isard, and Derek G Murray. "Scalability! But at what {COST}?" In: *15th Workshop on Hot Topics in Operating Systems (HotOS {XV})*. 2015.

[115] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[116] Janaina Mourão-Miranda et al. "Patient classification as an outlier detection problem: an application of the one-class support vector machine". In: *Neuroimage* 58.3 (2011), pp. 793–804.

[117] Shanmugavelayutham Muthukrishnan. "Data streams: Algorithms and applications". In: *Foundations and Trends® in Theoretical Computer Science* 1.2 (2005), pp. 117–236.

[118] Kazuyo Narita and Hiroyuki Kitagawa. "Outlier detection for transaction databases using association rules". In: *Web-Age Information Management, 2008. WAIM'08. The Ninth International Conference on*. IEEE. 2008, pp. 373–380.

[119] Hoang Vu Nguyen, Hock Hee Ang, and Vivekanand Gopalkrishnan. "Mining outliers with ensemble of heterogeneous detectors on random subspaces". In: *International Conference on Database Systems for Advanced Applications*. Springer. 2010, pp. 368–383.

[120] Keith Noto, Carla Brodley, and Donna Slonim. "FRaC: a feature-modeling approach for semi-supervised and unsupervised anomaly detection". In: *Data mining and knowledge discovery* 25.1 (2012), pp. 109–133.

[121] Travis E Oliphant. "Python for scientific computing". In: *Computing in Science & Engineering* 9.3 (2007), pp. 10–20.

[122] Matthew Eric Otey, Amol Ghoting, and Srinivasan Parthasarathy. "Fast distributed outlier detection in mixed-attribute data sets". In: *Data Mining and Knowledge Discovery* 12.2-3 (2006), pp. 203–228.

[123] Themistoklis Palpanas et al. "Distributed deviation detection in sensor networks". In: *ACM SIGMOD Record* 32.4 (2003), pp. 77–82.

[124] Spiros Papadimitriou et al. "Loci: Fast outlier detection using the local correlation integral". In: *Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405)*. IEEE. 2003, pp. 315–326.

[125] Daehyung Park et al. "Multimodal execution monitoring for anomaly detection during robot manipulation". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 407–414.

[126] Heiko Paulheim and Robert Meusel. "A decomposition of the outlier detection problem into a set of supervised learning problems". In: *Machine Learning* 100.2-3 (2015), pp. 509–531.

[127] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[128] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. "Incremental local outlier detection for data streams". In: *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*. IEEE. 2007, pp. 504–515.

[129] Sutharshan Rajasegarar, Christopher Leckie, and Marimuthu Palaniswami. "Hyperspherical cluster based distributed anomaly detection in wireless sensor networks". In: *Journal of Parallel and Distributed Computing* 74.1 (2014), pp. 1833–1847.

[130] Sutharshan Rajasegarar et al. "Centered hyperspherical and hyperellipsoidal one-class support vector machines for anomaly detection in sensor networks". In: *IEEE Transactions on Information Forensics and Security* 5.3 (2010), pp. 518–533.

[131] Sutharshan Rajasegarar et al. "Quarter sphere based distributed anomaly detection in wireless sensor networks". In: *Communications, 2007. ICC'07. IEEE International Conference on*. IEEE. 2007, pp. 3864–3869.

[132] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. "Efficient algorithms for mining outliers from large data sets". In: *ACM Sigmod Record*. Vol. 29. 2. ACM. 2000, pp. 427–438.

[133] Shebuti Rayana. "ODDS library". In: *Stony Brook,-2016* 2017 (2016).

[134] Fei Ren et al. "Using density-based incremental clustering for anomaly detection". In: *Computer Science and Software Engineering, 2008 International Conference on*. Vol. 3. IEEE. 2008, pp. 986–989.

[135] Saket Sathe and Charu C Aggarwal. "Subspace Outlier Detection in Linear Time with Randomized Hashing". In: *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE. 2016, pp. 459–468.

[136] Thomas Schlegl et al. "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery". In: *International Conference on Information Processing in Medical Imaging*. Springer. 2017, pp. 146–157.

[137] Bernhard Schölkopf et al. "Estimating the support of a high-dimensional distribution". In: *Neural computation* 13.7 (2001), pp. 1443–1471.

[138] Erich Schubert and Arthur Zimek. "ELKI: A large open-source library for data analysis-ELKI Release 0.7. 5" Heidelberg"". In: *arXiv preprint arXiv:1902.03616* (2019).

[139] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. "Fast and Scalable Outlier Detection with Approximate Nearest Neighbor Ensembles". In: *DASFAA (2)*. 2015, pp. 19–36.

[140] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. "Generalized outlier detection with flexible kernel density estimates". In: *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM. 2014, pp. 542–550.

[141] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. "Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection". In: *Data Mining and Knowledge Discovery* 28.1 (2014), pp. 190–237.

[142] Mark Schwabacher, Nikunj Oza, and Bryan Matthews. "Unsupervised anomaly detection for liquid-fueled rocket propulsion health monitoring". In: *Journal of Aerospace Computing, Information, and Communication* 6.7 (2009), pp. 464–482.

[143] Nidhi Singh and Craig Olinsky. "Demystifying Numenta anomaly benchmark". In: *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE. 2017, pp. 1570–1577.

[144] Rasheda Smith et al. "Clustering approaches for anomaly based intrusion detection". In: *Proceedings of intelligent engineering systems through artificial neural networks* (2002), pp. 579–584.

[145] Lorne Swersky et al. "On the evaluation of outlier detection and one-class classification methods". In: *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1–10.

[146] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. "Fast anomaly detection for streaming data". In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Vol. 22. 1. 2011, p. 1511.

[147] Mahbod Tavallaee, Natalia Stakhanova, and Ali Akbar Ghorbani. "Toward credible evaluation of anomaly-based intrusion-detection methods". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.5 (2010), pp. 516–524.

[148] Mahbod Tavallaee et al. "A detailed analysis of the KDD CUP 99 data set". In: *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE. 2009, pp. 1–6.

[149] William R Thompson. "On a criterion for the rejection of observations and the distribution of the ratio of deviation to sample standard deviation". In: *The Annals of Mathematical Statistics* 6.4 (1935), pp. 214–219.

[150] Jaree Thongkam et al. "Support vector machine for outlier detection in breast cancer survivability prediction". In: *Asia-Pacific Web Conference*. Springer. 2008, pp. 99–109.

[151] Wojciech Tylman. "Anomaly-based intrusion detection using Bayesian networks". In: *Dependability of Computer Systems, 2008. DepCos-RELCOMEX'08. Third International Conference on*. IEEE. 2008, pp. 211–218.

[152] Owen Vallis, Jordan Hochenbaum, and Arun Kejariwal. "A novel technique for long-term anomaly detection in the cloud". In: *6th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 14)*. 2014.

[153] Chengwei Wang et al. "Statistical techniques for online anomaly detection in data centers". In: *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE. 2011, pp. 385–392.

[154] Yu Wang et al. "Online anomaly detection for hard disk drives based on mahalanobis distance". In: *IEEE Transactions on Reliability* 62.1 (2013), pp. 136–145.

[155] Geoffrey I Webb et al. "Characterizing concept drift". In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 964–994.

[156] James M Whitacre, Tuan Q Pham, and Ruhul A Sarker. "Use of statistical outlier detection method in adaptive evolutionary algorithms". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM. 2006, pp. 1345–1352.

[157] Ke Wu et al. "RS-Forest: A rapid density estimator for streaming anomaly detection". In: *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE. 2014, pp. 600–609.

[158] Drausin Wulsin et al. "Semi-supervised anomaly detection for EEG waveforms using deep belief nets". In: *2010 Ninth International Conference on Machine Learning and Applications*. IEEE. 2010, pp. 436–441.

[159] Yan Xia et al. "Learning discriminative reconstructions for unsupervised outlier removal". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1511–1519.

[160] Kenji Yamanishi and Jun-ichi Takeuchi. "A unifying framework for detecting outliers and change points from non-stationary time series data". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2002, pp. 676–681.

[161] Kenji Yamanishi et al. "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms". In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2000, pp. 320–324.

[162] Matei Zaharia et al. "Apache spark: a unified engine for big data processing". In: *Communications of the ACM* 59.11 (2016), pp. 56–65.

[163] Ke Zhang, Marcus Hutter, and Huidong Jin. "A new local distance-based outlier detection approach for scattered real-world data". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2009, pp. 813–822.

[164] Yang Zhang, Nirvana Meratnia, and Paul Havinga. "Outlier detection techniques for wireless sensor networks: A survey". In: *IEEE Communications Surveys & Tutorials* 12.2 (2010), pp. 159–170.

[165] Yang Zhang et al. "Statistics-based outlier detection for wireless sensor networks". In: *International Journal of Geographical Information Science* 26.8 (2012), pp. 1373–1392.

[166] Ying Zhang et al. "Combining motion and appearance cues for anomaly detection". In: *Pattern Recognition* 51 (2016), pp. 443–452.

[167] Yue Zhao, Zain Nasrullah, and Zheng Li. "PyOD: A Python Toolbox for Scalable Outlier Detection". In: *Journal of Machine Learning Research* 20.96 (2019), pp. 1–7. URL: http://jmlr.org/papers/v20/19-011.html.

[168] Chong Zhou and Randy C Paffenroth. "Anomaly detection with robust deep autoencoders". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2017, pp. 665–674.

[169] Arthur Zimek et al. "Subsampling for efficient and effective unsupervised outlier detection ensembles". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 428–436.