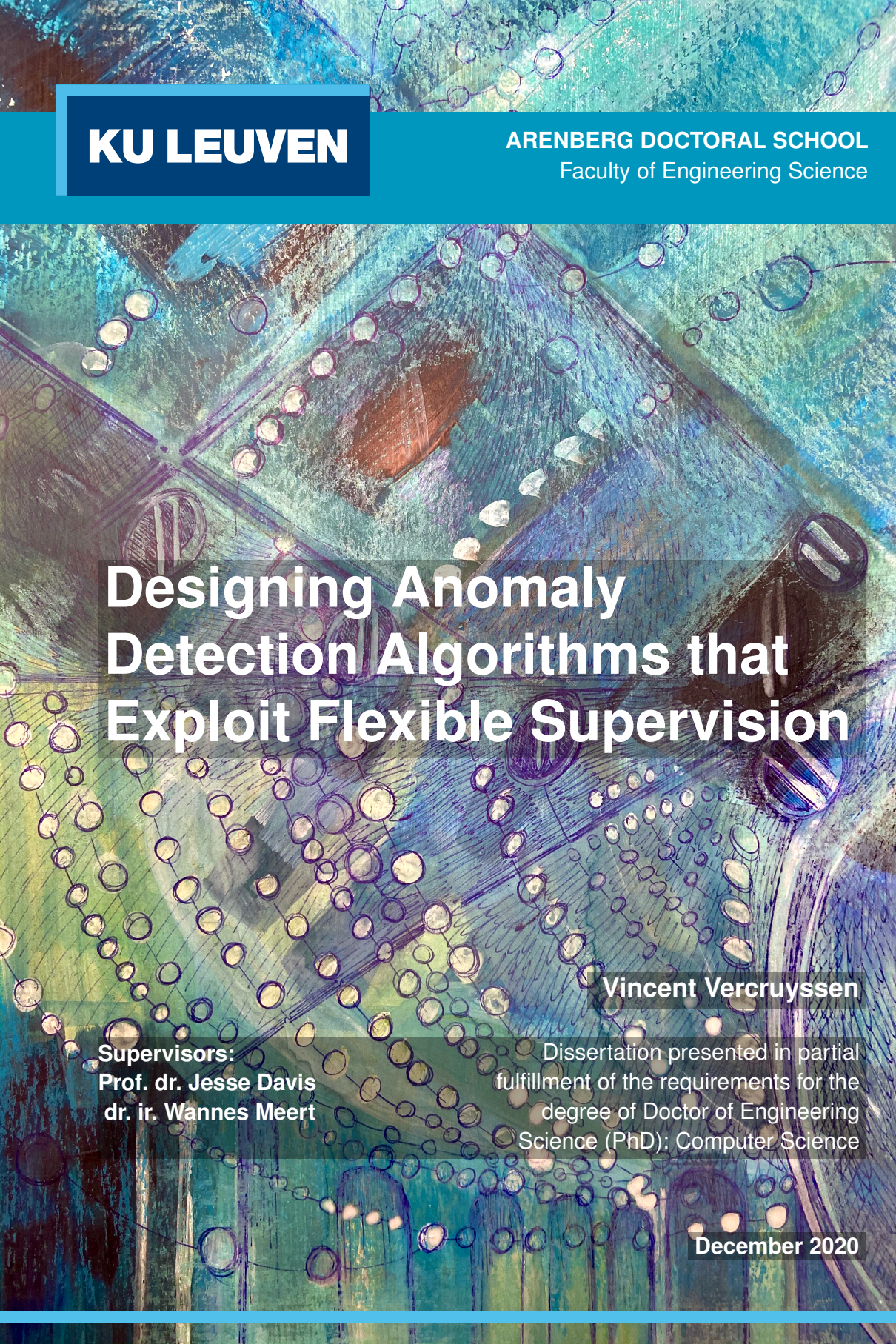


KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Designing Anomaly Detection Algorithms that Exploit Flexible Supervision

Vincent Vercruyssen

Supervisors:
Prof. dr. Jesse Davis
dr. ir. Wannes Meert

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

December 2020

Designing Anomaly Detection Algorithms that Exploit Flexible Supervision

Vincent VERCRUYSSEN

Examination committee:
Prof. dr. ir. Jean Berlamont, chair
Prof. dr. Jesse Davis, supervisor
dr. ir. Wannes Meert, supervisor
Prof. dr. ir. Luc De Raedt
Prof. dr. ir. Jan Aerts
Prof. dr. Bart Goethals
(Universiteit Antwerpen)
Prof. dr. Arthur Zimek
(University of Southern Denmark)
Prof. dr. Varun Chandola
(University at Buffalo)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

December 2020

Acknowledgements

The cosmos is within us. We are made of star-stuff. We are a way for the universe to know itself.

— Carl Sagan, Cosmos: A Personal Voyage

How to begin an ending?

Mijn interesse in wetenschap is pas recentlijk opengebloeid. Tijdens de laatste jaren van mijn studie handelsingenieur om precies te zijn. Rond die tijd kreeg ik van mijn grootvader het boek *De ongelovige Thomas heeft een punt* van Johan Braeckman en Maarten Boudry cadeau. Wat een geweldige introductie tot gezond skepticisme en de positieve kracht van wetenschap! Carl Sagan en zijn memorabele TV serie *Cosmos*, later nieuw leven ingeblazen door de eigenzinnige Neil deGrasse Tyson, wakkerden mijn niewsgierigheid verder aan. Boeken en series over dé Wetenschap werden al gauw een vast onderdeel van mijn dagelijkse routine. Ik begon me stilaan te realiseren dat ik deel wou uitmaken van deze wondere wereld. Dat inzicht ging echter gepaard met het besef dat mijn studies me al die jaren hadden voorbereid op een heel ander type carière, één waarin wetenschap tot de periferie behoorde. Wat doe je dan? Die vraag heb ik me al te vaak gesteld en naar het antwoord heb ik lang gezocht. Na veel slapeloze nachten besloot ik om de gok te wagen en me in te schrijven voor de master Artificiële intelligentie aan de KU Leuven. Vanaf mijn eerste dag op de nieuwe schoolbanken, heeft het onderwerp me diep geboeid. Robots maken! Wie kan dat niet bekoren?

In het begin van februari 2015, in het hartje van de winter en met de eerste examenperiode van de master achter de rug, kreeg ik een mailtje van twee - mij toen nog ongekende - professoren, Jesse Davis en Luc De Raedt. “Wil ik doctoreren in hun onderzoeksgroep?” Het was de opportuniteit waar ik op gehoopt had, maar die ik niet had verwacht. Dit kon ik niet laten liggen. In de late zomer, na het afronden van mijn verhuis naar een apartement in het

hartje van de stad, was het tijd om aan het echte werk te beginnen: AI research. Tijdens een doctoraat sta je aan de frontlinie van de wetenschappelijke kennis en moet je terrein veroveren op de onwetendheid. Dat is soms mikken in het duister en je moet vaak geluk hebben als je iets wilt raken. Ik heb echter steeds kunnen rekenen op een peleton van mensen om me te ondersteunen. Zij verdienen een dankwoord, want zonder hun hulp had ik dit boek nooit kunnen schrijven.

Jesse Davis, thank you for your continued trust in me. As my supervisor, your candid feedback and astute insights have been the cornerstone of my work. You taught me how to conduct proper research and how to write it down, always challenging me to push my limits without overwhelming me.

Wannes Meert, je bent een absoluut rolmodel voor mij. Jouw AI kennis reikt van domeinoverschrijdende inzichten tot de kleinste details. Ik hoop nog veel te mogen leren van je.

Thank you members of my jury. Luc De Raedt and Jan Aerts, for your insightful feedback from the start. Bart Goethals, for a joyful and fruitful collaboration. Varun Chandola, for writing the seminal paper that started my journey. Arthur Zimek, for your work had a tremendous impact on my understanding of the field. Jean Berlamont, for chairing the jury.

I want to thank VLAIO and the Flemish Government for supporting my research through the *VLAIO-SBO grant HYMOP* and the *Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen* grant.

During my PhD, I had the pleasure to collaborate extensively with Len Feremans and Boris Cule from the Adrem Data Lab of the Universiteit Antwerpen. Dr. Len, samenwerken met jou was een plezier en geen job.

Friends and colleagues from the DTAI research group who made my days both at the office and after the hours, thank you. Vova, for being my mentor in all things computer science and indulging me in my incessant questions. Tias, ik kan nog steeds jongleren! Mohit, my New York roomie. Samuel and our ever absurd, yet no-nonsense conversations. Laurens en onze eenmalige wekelijkse workout routine in hartje Polen. Jan, zullen we straks een pizza bestellen? Lorenzo, my complement in anomaly detection.

Jessa, Tom, en Maaike, met wie ik een kantoor heb mogen delen afgelopen twee jaar. Maaike, je bracht veel nieuwe energie in ons kantoor. Tom, je was mijn partner-in-crime zowel op kantoor als in het buitenland. Jessa, jij bent mijn vriend sinds het begin.

Muziek heeft me gepassioneerd en vreugde gegeven op momenten dat wetenschap het niet kon. Het was en is een waar genoegen om samen met de wonderlijke mensen van Closer to Home, Theater Flanel, Karthago, en ook Muze-cal het

podium te kunnen delen. Tom, mijn leerkracht en muzikale samenzweerder, ik heb nog veel te leren van jou.

Een PhD kan vaak eenzaam en rechtlijnig zijn, ware het niet voor enkele bijzondere mensen. Tim, mijn meest-inspirerende, muzikale vriend, het is een geluk jou te kennen. Charlotte, Michiel, Philippe, Kaija, Ina, Joren, Bastiaan, en David, ik kan me geen betere vrienden bedenken om de dingen die er écht toe doen mee te beleven. Pedro, from romie to homie, from the Dekenstraat to wherever we live. Jeroen, mijn filosofische vriend voor het leven. Dries, levensgenieter van dienst, met jou is het steeds feest. Jonas en Sander, the dynamic duo, jullie passie voor wetenschap heeft het vuur in mij aangewakkerd. You guys rock (literally)! Gertjan, je bent eersteklas, op en top. Nigel, Stefan, en de UA'ertjes, ook zovele jaren na het afstuderen voelt iedere keer dat ik jullie zie een beetje aan als thuiskomen. Thank you Mama Jesté, illustratrice, for the cover design.

Ook mijn familie was er steeds voor mij. Vake, met jou kan ik nog steeds de interessantste discussies voeren. Elske, je was stevast enthousiast voor mij. Opa en Bonnie, danku dat jullie er voor mij waren, van jongs af aan. Leo, onze familie is niet compleet zonder jou.

Mama, Papa, en Inge, jullie zijn mijn vurigste supporters al vanaf het prille begin, doorheen de moeiteloze en de moeilijke momenten. Het doet me zoveel plezier dat ik dit moment met jullie kan delen.

Tanine, you make me happy like nobody else in this world can.

En daarmee kom ik tot het einde van dit dankwoord en het begin van mijn eigenlijke werk. Er zijn nog zovele anderen die een plaatsje verdienen in bovenstaande paragrafen, te veel om hier te vermelden. Weet dat ik jullie allen in gedachten heb tijdens dit schrijven. Nu ik terugblik op mijn doctoraat, kan ik eerlijk zeggen dat er een aantal momenten waren waarop ik het wou opgeven. De twijfel over mijn onderzoek steekt soms nog de kop op en dat zal wellicht nog vaak gebeuren (Phyro van Ellis zou trots zijn). En doorheen de tijd is mijn naïve, romantische beeld van de wetenschap wat verdwenen. Maar leidt hier niet uit af dat het niet waardevol was. Integendeel. Ik een diep inzicht verworven in mijn vakgebied, een blik geworpen achter de schermen van academia, en mijn wereld mogen verruimen tot buiten de grenzen van de wetenschappelijke kennis. En laat dat nu juist wetenschap zijn: je eigen vooroordelen aan de kant schuiven om plaats te maken voor de realiteit.

Happy reading!

Vincent Vercruyssen
Leuven, December 2020

Abstract

Anomaly detection is the task of identifying observations in a dataset that do not conform the expected behavior. It is a crucial data mining task as in the real world, anomalous observations often correspond to real costs. For example, a machine that breaks, a fraudulent credit card transaction, or a patient experiencing irregular heart rhythms. With the advent of big data, manually sifting through millions of observations to detect the anomalies has become intractable. It is too time-consuming and costly. Instead, we need to design algorithms that automate this job for us.

Most existing anomaly detection algorithms work in a purely data-driven manner, meaning that they only look at the raw data to identify the anomalies. This requires making upfront assumptions about what may constitute anomalous behavior, such as anomalies that are infrequent or are substantially different from the normal observations. In many real-world applications, however, the anomalies do not conform to these assumptions. For example, some normal behaviors occur less frequently than the anomalous behaviors, such as a machine maintenance operation carried out only once in a while. The disconnect between the assumptions and reality results in a mismatch between what the detection algorithm predicts to be an anomaly and what is actually an anomaly. How can we design anomaly detection algorithms that can deal with such adverse effects?

The hypothesis of this dissertation states that *anomaly detection algorithms would derive considerable benefit from exploiting flexible expert supervision.* Flexible supervision refers to all forms of knowledge a domain expert has about a real-world application. By integrating this knowledge somehow in the detection algorithm, we could improve its performance. For example, the expert could correct the algorithm to stop flagging routine maintenance operations as anomalous. Currently, a handful of anomaly detection algorithms exists that can exploit such simple binary label information (an observation is anomalous or normal) obtained from the expert. However, flexible supervision

goes substantially beyond this classic binary label format.

This dissertation makes three scientific contributions, each related to exploiting flexible supervision in anomaly detection. The first contribution is an anomaly detection algorithm that exploits the knowledge the expert has about sporadically reoccurring patterns in the data, such as maintenance operations. If such a pattern does not occur when it is expected to, it gives rise to an absent pattern anomaly. In contrast to regular anomalies, absent pattern anomalies are identified by the absence of normal behavior, not by the presence of anomalous behavior. We introduce an algorithm that exploits a limited set of annotated occurrences of a pattern provided by the expert, to detect its suspicious absences.

The second contribution is an anomaly detection algorithm that exploits the knowledge contained in event logs. In an event log, the expert keeps track of all events that could be instrumental in identifying patterns in the data. We develop an algorithm that exploits the information contained in both event logs and continuous time series data (e.g., water consumption measurements in a retail store over time) to detect periods of anomalous behavior in the time series data (e.g., leaks or spills in the store).

The final contributions of this dissertation focuses on optimally exploiting the binary label information obtained either from the expert or available for a related dataset. For many real-wold problems, we have multiple, related datasets at our disposal. For example, if we are collecting data for multiple machines. The expert provides label information for only a subset of these datasets. We design a label propagation algorithm for augmenting the anomaly score of any unsupervised anomaly detector with a binary label information obtained from the expert through an active learning strategy. The use of an unsupervised anomaly detector allows one to derive an initial anomaly score for each instance in a dataset, while the label propagation can correct this initial score in a model-agnostic manner to better reflect the expert knowledge about anomalous or normal behavior. Finally, we introduce several algorithms for transferring label information between two different, yet related datasets. These algorithms compare the data distributions of the two datasets to determine whether they are similar which would justify transferring label information.

Beknopte Samenvatting

Anomalie detectie heeft tot doel om die observaties in een dataset te identificeren die niet overeenkomen met wat men zou verwachten. Het is een cruciale data mining taak gezien abnormale observaties veelal corresponderen met reële kosten. Bijvoorbeeld: een machine die kapot gaat, een fraduleuze kredietkaart transactie, of een patiént die een onregelmatig hartritme ervaart. De komst van big data heeft het onmogelijk gemaakt om manueel miljoenen observaties te inspecteren om de anomalieën eruit te halen. Dat is té kostelijk en té tijdsrovend. In plaats daarvan is het onze taak om computeralgoritmes te ontwerpen die deze taak voor ons oplossen.

De bestaande anomaliedetectie algoritmes werken veelal op een volledig data-gedreven manier. Concreet betekent dit dat ze proberen anomalieën te detecteren enkel op basis van de ruwe data. Dit vereist dat men vooraf specifieke assumpties maakt over hoe abnormaal gedrag er zou kunnen uitzien, zoals “anomalieën zijn infrequent of zijn substantieel verschillend van de standaard, normale observaties.” In de praktijk is het echter zo dat anomalieën niet altijd voldoen aan deze assumpties. Soms komt bepaald normaal gedrag minder frequent voor dan abnormaal gedrag. Een praktisch voorbeeld hiervan zijn sporadische onderhoudsoperaties. Deze discrepantie tussen de assumpties en de praktijk resulteert in een mismatch tussen wat het detectie algoritme denkt dat de anomalieën zijn en wat daadwerkelijk abnormaal is. Hoe kunnen we nu algortimes ontwikkelen die erin slagen om zulke tekortkomingen te omzeilen?

De centrale hypothese van dit proefschrift is dat *anomaliedetectie algoritmes substantieel gebaat zijn bij het exploiteren van flexibele expert supervisie.* Flexibele supervisie verwijst naar al de mogelijke vormen van domeinkennis waarover een expert beschikt voor een welbepaalde applicatie. Door deze kennis op een bepaalde manier te integreren in de detectiealgoritmes, kunnen we hun performantie verbeteren. De expert kan bijvoorbeeld aangeven dat onderhoudsoperaties normaal zijn zodat het algoritme deze ook zo classificeert. Op dit moment bestaat er slechts een handvol algoritmes dat enkel binaire

labelinformatie verkregen van de expert (een observatie is normaal of abnormaal) kan verwerken. Flexibele supervisie gaat echter een pak verder dan dit standaard binair label formaat.

Dit proefschrift maakt drie wetenschappelijke contributies die elk relateren tot het exploiteren van flexibele expert supervisie in anomaliedetectie algoritmes. De eerste contributie is een anomaliedetectie algoritme dat de kennis die de expert heeft over sporadisch voorkomende patronen in de data, zoals onderhoudsoperaties, exploiteert. Wanneer men verwacht een dergelijk patroon te zien in de data, maar het verschijnt niet, spreken we van een *absent pattern* anomalie. In tegenstelling tot reguliere anomalieën, kan men absent pattern anomalieën identificeren op basis van de afwezigheid van normaal gedrag, niet de aanwezigheid van abnormaal gedrag. We introduceren een algoritme dat een beperkte set van door de gebruiker geannoteerde patronen exploiteert om hun verdachte afwezigheid op te sporen.

De tweede contributie is een anomaliedetectie algoritme dat de kennis die vervat zit in event logs exploiteert. In een event log bewaart de expert al de events die nuttig zouden kunnen zijn om patronen in de data te identificeren. We ontwikkelen een algoritme dat de informatie vervat in zowel continue tijdsreeksdata (bv. waterconsumptie metingen in een supermarkt doorheen de tijd) als de bijhorende discrete event logs gebruikt om abnormaal gedraag in de tijdsreeksdata te identificeren (bv. waterlekken in de winkel).

De finale contributies van deze thesis richten zich op het optimaal uitbuiten van binaire labelinformatie ofwel verkregen van de expert ofwel beschikbaar voor een gerelateerde dataset. Voor veel praktische problemen uit de echte wereld, hebben we meerdere, doch gerelateerde datasets ter beschikking. De meeste bedrijven verzamelen bijvoorbeeld data voor meerdere machines tegelijkertijd. De expert voorziet binaire labelinformatie voor slechts enkele van deze datasets. We ontwerpen een labelpropagatie algoritme dat toelaat om de anomaliescores berekend door gelijk welk data-gedreven detectie algoritme aan te passen op basis van labelinformatie. De labels zijn verkregen door middel van een active learning strategie. Het gebruik van de data-gedreven detector laat toe om een initiële anomaliescore te berekenen voor iedere observatie in de dataset, terwijl de labelpropagatie ons in staat stelt om deze score op een model-agnostische manier te corrigeren. Tot slot introduceren we verschillende algoritmes om binaire labelinformatie over te dragen tussen twee verschillende, doch gerelateerde datasets. Deze algoritmes vergelijken de dataverdelingen van de twee datasets om te besluiten of ze gelijkaardig genoeg zijn om de labeltransfer te verantwoorden.

List of Abbreviations

tn	True negatives (normals)
fn	False negatives
tp	True positives (anomalies)
fp	False positives
R	Recall
P	Precision
TNR	True negative rate
ROC	Receiver operating characteristic
PR	Precision-recall curve
AUROC	Area under the ROC curve
AUPRC	Area under the PR curve
AP	Average precision
$\mathbf{RE}_{x\%}$	Required effort at $x\%$
PU	Positive and unlabeled (learning)
DTW	Dynamic time warping
i.i.d.	Independently and identically distributed

List of Symbols

$\mathcal{X}$	Input or feature space
$\mathcal{Y}$	Label space
$\mathcal{D}$	Domain
$\bullet_S$	Source $\bullet$
$\bullet_T$	Target $\bullet$
D	Dataset
x	Input instance
X	Set of input instances
y	Instance label
$\mathcal{T}$	Time series dataset
T	Continuous time series
E	Discrete event log
v	Measurement in a continuous time series
e	Event in a discrete event log
t	Timestamp
S	Dataset subsample or time series segment
$\mathcal{S}$	Set of dataset subsamples or time series segments
$\mathcal{P}$	Pattern
I	Itemset

I_s Sequential pattern

n, m, l Counts and lengths

$\mathcal{C}$ Set of clusters

C Cluster

c Cluster center

h Anomaly detection model

$\mathcal{H}$ Hypothesis space

a Anomaly score function

p Probability function

γ Contamination factor

δ Distance function

ν Normalization function

k Number of neighbors parameter

ψ Size or count parameter

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> *Look again at that dot. That's here. That's home. That's us. On it everyone you love, everyone you know, everyone you ever heard of, every human being who ever was, lived out their lives. The aggregate of our joy and suffering, thousands of confident religions, ideologies, and economic doctrines, every hunter and forager, every hero and coward, every creator and destroyer of civilization, every king and peasant, every young couple in love, every mother and father, hopeful child, inventor and explorer, every teacher of morals, every corrupt politician, every "superstar," every "supreme leader," every saint and sinner in the history of our species lived there-on a mote of dust suspended in a sunbeam.*
>
> — Carl Sagan, Pale Blue Dot

Present technology gives us an unprecedented ability to gather data about almost any imaginable real-world process. At the same time, increasing computing power enables us to carry out more complex analyses on these data, such as learning more powerful predictive models or sifting through more data to discover interesting nuggets of information. Hence, it is unsurprising that *data* is becoming the currency of the 21st century. By using the algorithmic tools of the field of Artificial Intelligence, we can exploit these huge volumes of data to solve complex tasks and make informed decisions.

In this dissertation, we focus on one centrally important data mining task: *anomaly detection*. The goal of anomaly detection is to find the observations in a dataset that deviate from the expected patterns of normal behavior [21]. Consider the data from five stores operated by a fictional retail company shown in Table 1.1. For each store, we know its size, whether it has an on-site butcher

	Size (in m^2)	Butcher shop?	Daily sales (in euro)	Anomaly?
Store 1: 01-09-2020	1.700	yes	24.000	no
Store 2: 01-09-2020	1.550	no	17.000	no
Store 3: 01-09-2020	1.620	yes	26.000	no
Store 4: 01-09-2020	1.460	no	15.500	no
Store 5: 01-09-2020	1.750	yes	16.000	yes

Table 1.1: An example of anomaly detection. Each row represents some data collected for a store of a fictional retail chain. The columns show which data are collected. Store 5 has significantly lower daily sales than the other stores that also have a on-site butcher shop. This is unexpected.

shop or not, and its daily sales on a particular day. The question we want to answer is whether any of theses stores' data is unexpected? If we divide the stores into two groups: those with an on-site butcher shop and those without a butcher shop, we can see that daily sales in store 5 are much lower (16.000 euros) than the other stores that also have an on-site butcher shop (24.000 euros and 26.000 euros) despite being a slightly larger store. This is clearly not in line with our expectations. This is an *anomaly*.

There are many real-world examples and independent studies that clearly illustrate why anomaly detection is such an important task [66, 11]. For one, normal patterns should be learned from data. Furthermore, anomalies are unique and usually unknown, but should still be recognized. Intrusion detection systems monitor huge volumes of network traffic data and try to filter out the *malicious* traffic indicative of attempts to access, attack, or otherwise damage the network [128, 117]. Banks employ intricate algorithms to sift through millions of credit card transactions in order to identify *fraudulent* transactions [1]. Anomaly detection algorithms are used for prognostics and health management in industrial environments, where the goal is to monitor industrial equipment continuously to detect any *abnormalities* in their behavior [65]. In order to track the health of patients, it is possible to equip them with long-term ECG-monitoring wearables that continuously record their heart rate. Any *changes* in the morphology of the heartbeats can be indicative of serious underlying problems and should be identified immediately [17]. Finally, modern-day chemical processes have become increasingly complex because of interactions between so many different components. A small error somewhere can rapidly cause large problems or even halt the process. Early detection of such faults with an anomaly detection system can prevent these costly breakdowns [91].

In the simplified example shown in Table 1.1, we can easily inspect each store manually to see whether anything is out of the ordinary. However, in reality a retail company will collect sales data on an hourly basis resulting in thousands if not millions of observations. Moreover, large retailers will operate

hundreds if not thousands of stores. Manually inspecting data on that scale is infeasible. Therefore, we need advanced anomaly detection algorithms that can automatically analyze the data.

1.1 Designing Anomaly Detection Algorithms

The first anomaly detection algorithms stem from the field of statistics [157]. These approaches assume that the normal data follow some statistical distribution and that anomalies lie outside of this distribution [58, 118].[1] These techniques arose mainly from the need to clean a dataset in order to remove unwanted outliers [90, 86]. Erroneous measurements can arise in dataset for a variety of reasons such as faulty sensors, laggy network connections, glitches and bugs in software programs, corrupted save files, unaccounted test conditions, lack of sleep, and a variety of other reasons. These outliers should be removed since they can distort the subsequent statistical analyses of the data.

As demonstrated by the example in Table 1.1, anomaly detection clearly goes substantially beyond removing outliers from the data. The low daily sales of store 5 are not the result of a faulty measurement, nor do we want to simply remove this observation from the data. Rather, our goal is to reveal similar observations because they may lead to actionable insights. In this case, the retail company can try to find out why the sales were low in that particular store. Did the computer system not record certain sales? Or does the store have a stock shortage causing low sales?

Starting almost 20 years ago, anomaly detection as a field has evolved from statistical approaches to more data-mining-oriented approaches [157]. There are roughly two reasons for this shift. First, the scale-up of databases shifted the focus from statistically interpretable detection methods to algorithmic scalability. Second, the statistical methods are not always flexible enough to accurately detect anomalies in complex, real-world applications. The shift has resulted in a trove of anomaly detection algorithms [121, 113, 13, 134, 62, 103, 52, 77, 78, 85, 5, 110, 87].[2]

[1] Anomalies were termed *outliers* because they were considered to be observations literally *lying outside* of the normal data distribution.

[2] This is not an exhaustive list, rather the tip of the iceberg.

	4 types of flexible supervision			
	Binary label	**Recurring maintenance pattern**	**Event logs**	**Related dataset**
Store 1	?	Usually every 2 months	<cleaning>	-
Store 2	?	Usually every 3 months	-	Similar dataset store 10
Store 3	?	Usually on Mondays	<sales,cleaning>	-
Store 4	?	Once a year	-	-
Store 5	normal	Usually around 10am	<maintenance>	-

Table 1.2: Four types of flexible supervision the domain expert can provide for the example of Table 1.1. Each column contains one type: binary label information, knowledge of when the maintenance operation is supposed to occur, an event log, and the availability of a related dataset.

1.2 Exploiting Flexible Supervision

A crucial limitation of the current algorithms is that the vast majority is purely unsupervised [41, 15, 53, 132, 38, 137, 130]. Because these algorithm only consider the raw data to identify the anomalies, they must make specific assumptions upfront about what may constitute anomalous behavior. For example, anomalies are infrequent and somehow different from the normal observations [60]. This allows an observation's anomalousness to be determined by contrasting it to the bulk of the data. In practice, most observations are indeed normal. In network intrusion detection, for example, most network traffic is benign [117].

Many real-world applications do not conform to these assumptions about anomalies [145]. Consider again the example in Table 1.1. Suppose it turns out that the butcher shop in store 5 is currently undergoing a routine, scheduled maintenance operation, which means that the butcher shop is not open to customers. This information would explain the store's low daily sales. Such maintenance operations can happen less frequently than some anomalous behaviors, thus violating the assumption made by the data-driven algorithms. How can we adapt the detection models to reflect this knowledge? Clearly, we do not want them to flag an anomaly each time a store undergoes maintenance in the future. One option is to have a domain expert from the retail company manually correct each prediction made by the algorithm. However, this is not a good idea for two reasons. First, this task is tedious and labour-intensive. Second, it is repetitive and therefore not a smart investment of resources. A solution is to design algorithms that learn a model both from data and a limited amount of binary label information (an observation is normal or anomalous) provided by the expert [135, 55, 126, 94, 31]. Table 1.2 illustrates this for the retail example. The first column of the table contains the labels provided by the domain expert where a question mark indicates that no label was provided.

However, the existing algorithms can only handle feedback in the form of binary labels [55, 126, 94, 31]. Meanwhile, the domain expert might possess additional types of knowledge about the application that can improve the detection algorithms, but that do not present themselves neatly in the format of binary labels. The columns in Table 1.2 contain four such types of knowledge applied to the retail example. First, binary label information consists of the user indicating whether an observation is normal or anomalous (or unknown). This type of supervision is common and easy to interpret. Second, the expert sometimes has knowledge of sporadically occurring patterns in the data, such as maintenance operations. Knowing when such a pattern normally occurs in the data is helpful to identify anomalous behavior (e.g., when a scheduled maintenance operation does not take place). Third, event logs store additional information, such as when a cleaning or maintenance operation takes place, which can be useful for an anomaly detector to pick up on specific patterns in the data (e.g., a drop in daily sales corresponds to maintenance). Finally, there might exist a dataset related to the data at hand and containing information that can be used to improve the current anomaly detection algorithm (e.g., labeled data of three similar stores operated by the retailer during the past years). We will collectively refer to this additional knowledge as *flexible supervision*.

While the field of anomaly detection has seen a boom in recent decades, researchers have not looked at flexible supervision for anomaly detection. Concretely, we identified three shortcomings related to this gap in the literature. First, there is no algorithm that can detect when a sporadically occurring pattern is suspiciously absent. Second, there is no algorithm that integrates the information contained in both the data and the accompanying event logs in a coherent way that naturally fits existing anomaly detection algorithms. Third, there are no algorithms that can augment any unsupervised anomaly detector with label information provided either for the dataset itself or a related dataset relevant to the problem at hand.

1.3 Dissertation Statement

This dissertation hypothesizes that *anomaly detection algorithms would derive substantial benefit from being able to exploit flexible forms of supervision.* We design a number of novel algorithms that address real-world problems. Each time, we answer two questions. First, *which* form of supervision is relevant to the problem at hand? And second, *how* can we develop anomaly detection algorithms that effectively exploit this supervision? Concretely, we address the three aforementioned shortcomings.

1. How can we exploit information about sporadically occurring patterns in a dataset to detect when the pattern is suspiciously absent?

2. How can we represent the information contained in both the data and the accompanying event logs such that it can be exploited by existing anomaly detection algorithms?

3. How can we use the (limited) label information provided for a dataset or a related, yet different, dataset to improve the predictions made by any unsupervised anomaly detection algorithm?

1.4 Contributions

This dissertation presents a number of novel algorithms that extend the state-of-the-art in the field of anomaly detection. The algorithms are part of three contributions.

1.4.1 Contribution 1: Exploiting Pattern Information

Designing an algorithm to detect absent pattern anomalies. The first contribution of this dissertation takes place against the backdrop of time series anomaly detection. A time series is a series of measurements of a variable over time, such as the hourly sales data of a retail store. Time series anomaly detection aims to identify periods of anomalous behavior, such as an hour characterized by an abnormal sales volume.

We introduce a novel anomaly detection setting where the user[3] is interested in detecting the absence of a sporadically occurring pattern within the context it is expected to appear. We formalize this type of anomaly as the *absent pattern anomaly*. In contrast to the classic anomaly detection paradigm, where the *presence* of unexpected behavior identifies an anomaly, the *absent* pattern anomaly is identified by the *absence* of a behavior that is expected to reoccur. In real-world applications, absent patterns are linked to serious problems. For example, the retailer expects the pattern connected to a maintenance operation to show up in the data every now and then. If that is not the case, it is an indication that the maintenance did not take place which could lead to problems manifesting down the line.

The user provides *flexible supervision* by annotating several instances of the pattern. The user provides no labels as to what is anomalous nor as to the context that is predictive of the pattern's occurrence. The supervision can simply be viewed as feedback that the pattern is relevant. We design an algorithm that exploits this supervision to detect suspicious pattern absences. First, it detects all occurrences of the pattern in a time series based on the expert supervision. Then, it learns how the pattern is distributed within the time series as well as what the relevant context is within which it appears. An experimental evaluation of our algorithm shows that it works well on both real-world and synthetic datasets.

Contribution 1 has been published at a peer-reviewed conference:

[3]We will sometimes refer to the *user* or *(domain) expert*, i.e., the person who is knowledgeable about the application and is interested in the output of the algorithms.

V. Vercruyssen, W. Meert, and J. Davis (2020). "Now you see it, now you don't" Detecting Suspicious Pattern Absences in Continuous Time Series. In *Proceedings of the 2020 SIAM International Conference on Data Mining (SDM)*, Cincinnati (US), pp.127-135.

1.4.2 Contribution 2: Exploiting Event Log Information

Designing an algorithm to transform mixed-type time series into a representation suitable for anomaly detection algorithms. The existing time series anomaly detection algorithms focus either on continuous time series or on discrete event logs but not on the combination thereof. However, in many practical applications, the patterns extracted from the event log reveal contextual and operational information important for the detection of anomalous behavior in the continuous time series data. Logging of maintenance or cleaning operations, for example, can help to identify them as normal behavior (see the example in Table 1.1).

The user provides *flexible supervision* by supplying event logs containing events that could explain the behavior observed in the continuous time series. The user does not know which events correspond to normal or anomalous patterns. We design a frequent-pattern-based algorithm that transforms the combination of univariate or multivariate continuous time series and discrete event logs into a representation that is suitable for applying anomaly detection algorithms. As such, they can exploit the information in the event logs. Our algorithm leverages frequent pattern mining techniques to construct this embedding. An extensive experimental evaluation shows the advantages of our framework over the state-of-the-art.

Contribution 2 has been published at a peer-reviewed conference:[4]

L. Feremans*, V. Vercruyssen*, B. Cule, W. Meert, and B. Goethals (2019). Pattern-Based Anomaly Detection in Mixed-type Time Series. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, Würzburg (Germany), pp. 240-256.

[4]Len Feremans and Vincent Vercruyssen contributed equally to this article.

1.4.3 Contribution 3: Exploiting Binary Label Information

Designing a general algorithm to augment unsupervised anomaly detectors with binary label information. The previous contributions dealt with non-standard forms of flexible supervision. The third contribution of this dissertation, on the other hand, deals with classic binary label information (an observation is anomalous or not). While in rare cases it might be possible to learn an effective model for anomaly detection in a purely data-driven manner, the assumptions underpinning such algorithms generally do not exactly line up with what the expert considers to be anomalous behavior. Our goal here is to complement the data-driven models with additional knowledge queried from the expert in the form of binary labels.

We contribute a general algorithm for extending any unsupervised anomaly detection algorithm with a limited number of labels provided by the domain expert. Starting from an unlabeled dataset, the approach is able to gradually incorporate expert-provided feedback to improve its performance. It does so by propagating the label information through the dataset and using it to correct an initial data-driven anomaly score for each observation. The labels are acquired through an active learning strategy. Our algorithm is evaluated on the real-world anomaly detection problem of detecting abnormal water consumption in a retail store. We collaborated with the Colruyt Group, a large Belgian retail company, to deploy the algorithm in practice and track its performance over the span of several months.

Contribution 3 has been published at a peer-reviewed conference:

V. Vercruyssen, G. Verbruggen, K. Maes, R. Bäumer, W. Meert, and J. Davis (2018). Semi-supervised Anomaly Detection with an Application to Water Analytics. In *IEEE International Conference on Data Mining (ICDM)*, Singapore, pp. 527-536.

Introducing the transfer learning for anomaly detection task and designing several algorithms to tackle it. This dissertation aims to address the problem of label scarcity generally encountered in anomaly detection settings. Our previous contribution clearly showed that anomaly detection algorithms derive substantial benefits from exploiting a limited amount of label information. However, this requires the expert to interact with the anomaly detection system and to provide feedback on its predictions. In most practical applications, such feedback is difficult and costly to obtain. Consider, for example, that the retail company operates hundreds of stores and wants to construct an anomaly

detection model for each store. The expert would have to provide labels for each store!

The availability of multiple related, yet different, datasets (e.g., a dataset for each store operated by the retailer) also entails an opportunity. The information contained in one dataset (the source) serves as a form of *flexible supervision* to improve the anomaly detection model in another dataset (the target). For example, if the user provides labels for a source retail store, these labels could be useful for a target store provided the stores are similar enough. However, we do not know beforehand which information is relevant to improving our target model or how to exploit it in our algorithm.

Our contribution is to cast this problem within the paradigm of transfer learning. We design multiple transfer learning algorithms tailored to anomaly detection. These algorithms first determine which information should be transferred from the source data to the target data. Then, they exploit the transferred information to improve the target anomaly detection model. Our experimental evaluation shows the benefit of applying these algorithms to real-world applications.

Contribution 4 has been published at a peer-reviewed conference and workshop:

V. Vercruyssen, W. Meert, and J. Davis (2020). Transfer Learning for Anomaly Detection through Localized and Unsupervised Instance Selection. In *Proceedings of the* 34^{th} *AAAI Conference on Artificial Intelligence*, New York City (US), vol. 34(4), pp. 6054-6061.

V. Vercruyssen, W. Meert, and J. Davis (2017). Transfer Learning for Time Series Anomaly Detection. In *Proceedings of the Workshop and Tutorial on Interactive Adaptive Learning at ECML PKDD*, Skopje (Macedonia), pp. 27-37.

1.5 Other Research Contributions

To present a coherent story, this dissertation focuses on the research related to context-aware anomaly detection based on publications where I am the first author. However, this is only a subset of the research I contributed to throughout my doctoral studies. This section summarizes these contributions and the resulting publications at peer-reviewed conferences.

1.5.1 Anomaly Detection Contributions

Quantifying the Confidence of Anomaly Detectors. Anomaly detection algorithms usually return a real-valued anomaly score indicating how anomalous an example is. By setting a threshold on the scores, they can be turned into binary predictions. While this is straightforward, different algorithms assign scores in distinct ways, making it often difficult to interpret or compare them. Moreover, small perturbations in the training data might cause the scores to change. This can quickly erode a user's trust in the predictions. One way to overcome these issues is to provide a measure of how *uncertain* - or reversely, *confident* - an anomaly detector is in its example-wise predictions which would allow the user to asses a detectors' reliability. We propose a two-step approach to obtain this confidence measure. First, the anomaly scores are transformed into anomaly probabilities in a Bayesian manner. Second, the probabilities are transformed into confidence scores by estimating the probability that an instance's score will be above the threshold based on a theoretical sample of the data. The threshold is derived using the contamination factor (see Chapter 2). We analyze the convergence behavior of the confidence measure theoretically and demonstrate its effectiveness on a large benchmark of datasets.

This contribution has been published at a peer-reviewed conference [109].

Estimating the Class Prior in the PU Active Learning Setting. In anomaly detection, the contamination factor is the proportion of anomalies in a dataset. Its counterpart in the field of positive and unlabeled (PU) learning is the *class prior*, i.e., the proportion of positive instances in a dataset [8]. Knowing the class prior enables learning better classifiers, so one important task is to estimate it from the data. If we have access to some labeled instances that are an unbiased subsample of all positive instances in the dataset, this task is relatively straightforward. In practice, however, labels are often acquired via an active learning strategy which guides the expert to provide those labels that are most informative for the classifier. As a result, the labeled instances are a biased subsample of the positive class. Anomaly detection naturally fits this PU setting as almost all provided labels will be for the normal (positive) instances. We design an algorithm that is capable of computing the class prior in the PU setting when the labels are acquired through an active learning strategy. The key insight is that whether an instance's label is observed or not depends on whether there exists a more informative instance (according to the active learning strategy) in the dataset. We show empirically that our algorithm accurately recovers the true class prior on a large benchmark of datasets.

This contribution has been published at a peer-reviewed conference [108].

Quantifying the Robustness of Anomaly Detectors. Naturally occurring variations in the training data can have an impact on the scores output by any anomaly detector. Ideally, an anomaly detector should be *robust* to small perturbations in the training data, meaning that its predictions should not change much if the training data change a little. Such perturbations are common to anomaly detection because anomalies are characterized by their unpredictability. Hypothetically, if we could collect a dataset multiple times, it would each time contain different anomalies and normal observations. We design a ranking stability measure that quantifies the robustness of any anomaly detector's predictions by looking at how consistently it ranks examples in terms of their anomalousness. The measure is tangential to the classic performance measures used for anomaly detection, such as area under the curve [15]. We demonstrate experimentally that it is able to capture stability in a way that we would expect it to. The stability measure could be used to decide between using different anomaly detectors that otherwise perform similarly.

This contribution has been published at a peer-reviewed workshop [107].

Developing a Pattern Mining and Anomaly Detection Framework. Analyzing time series data leads to the discovery of interesting patterns and anomalies. In recent years, numerous algorithms have been developed to discover interesting patterns in time series data as well as detect periods of anomalous behavior. However, these algorithms are often challenging to apply in real-world settings. We propose a framework, consisting of generic transformations, that allows to combine state-of-the-art time series representation, pattern mining, and pattern-based anomaly detection algorithms. The framework naturally handles a mix of multi-dimensional continuous time series and event logs. Additionally, we developed an open-source, interactive software tool that assists both pattern mining and domain experts to select algorithms, specify parameters, and visually inspect the results, while shielding them from the underlying technical complexity of implementing our framework.

This contribution has been published at a peer-reviewed workshop [46].

1.5.2 Sports Analytics Contributions

Predicting Soccer Passes. My final contribution is in the field of sports analytics. Given the advances in camera-based tracking systems, many soccer teams are able to record data about the players' position during a game. Analyzing these data is challenging, since they are fine-grained, contain implicit relational information between players, and contain the dynamics of the game.

We propose the use of qualitative spatial reasoning techniques to address these challenges, and test our approach by learning a model for pass prediction over a real-world soccer dataset. Experimental evaluation shows that our approach is capable of learning meaningful models.

This contribution has been published at a peer-reviewed workshop [142].

1.6 Structure of the dissertation

The structure of this dissertation is as follows:

Chapter 2 introduces the necessary background on anomaly detection. It explains the concepts necessary to understand the following chapters as well as place the contributions in the broader scientific scope.

Chapter 3 introduces the concept of absent pattern anomalies and discusses how to detect them in continuous time series.

Chapter 4 introduces a framework for mapping any univariate or multivariate time series and discrete event log to an embedding suitable for applying anomaly detection algorithms.

Chapter 5 introduces a general algorithm for updating the predictions of any unsupervised anomaly detector with label information provided by the human.

Chapter 6 explains how label scarcity in anomaly detection applications can be addressed from the paradigm of transfer learning.

Chapter 7 summarizes the contributions of the dissertation ans suggests a number of possible research questions for future work.

Chapter 2

Background

This chapter provides the necessary backdrop for the contributions of the dissertation. We start with defining what an *anomaly* is (Section 2.1). Then, we formalize the canonical anomaly detection problem and discuss its variants (Section 2.2), followed by an overview of the common taxonomies for anomaly types (Section 2.3). Next, we review the different anomaly detection paradigms based on the availability of label information, as well as a selection of specific detection algorithms relevant to the purpose of this dissertation (Section 2.4). In the following two sections, we provide the background on time series anomaly detection (Section 2.5) and transfer learning within the context of anomaly detection (Section 2.6). We end the chapter with a discussion of the common evaluation measures for anomaly detection algorithms (Section 2.7).

2.1 Defining Anomalies

The literature has referred to abnormal observations in a dataset as: *noise*, *outliers*, *novelties*, *discordant data objects*, *contaminants*, *anomalies*, and other terms. These terms have different, albeit somewhat overlapping, interpretations. Here, we discuss the differences between them and introduce a concrete definition of *anomalies*, the term we adopt for this dissertation.

The term *noise*, for our purposes, is connected with the use of statistical or signal processing techniques to model information in a dataset. It is the information that cannot be explained or captured by the model and likely the result of random processes that are unavoidable, such as deviations due to the use of

faulty measuring devices or human errors. Hence, noise is often associated with data cleaning, i.e., removing unwanted observations from the dataset [149].

Outliers are observations in the dataset that are literally outlying. They are "different" or "far away" from the other observations and bear limited or no resemblance to the bulk of the data. One of the earliest - yet still relevant - definitions is provided by Hawkins [60]:[1]

> "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."

The definition suggests the existence of a data generating mechanism for the normal observations. The link with the early statistical approaches to anomaly detection is clear: observations are either the result of this mechanism (the null-hypothesis) or not (the alternative hypothesis) and we can use a statistical test to find the correct answer [157]. Furthermore, the definition presumes the existence of a large set of normal observations against which we can compare the outliers. In other words, we can learn a model of normal behavior from the data such that outliers have a low probability of being generated by this model.

Anomalies are more general than outliers. First, in many real-world applications, anomalous behavior arises as a result of specific data generating mechanism, such as leaks in water consumption data (see Chapter 5). This corresponds to the idea of a *minority class* consisting either of observations that are exceptional to the bulk of the data, or that have a well-defined distribution [137]. Second, some normal behaviors, such as maintenance operations, are more exceptional than anomalous behavior. Thus, which observations are anomalous depends on the specific application. To reflect this insight, we define anomalies as: [2]

> An anomaly is an observation that deviates too much from what the domain expert desires within the context of a specific application.

In practice, there is a need for algorithms that can flexibly deal with anomalies that are not outliers. Such algorithms should try to exploit any (tacit) domain knowledge the expert has about the application.

Finally, *novelties* are observations that have not been previously observed in the data. They are, for example, new types of network intrusion attacks or new ways that an operating machine can fail. Novelty detection is closely related to anomaly detection in that the novelties are de facto exceptional to the bulk of

[1] Other contemporary definitions are variations of the same idea [6, 57].
[2] This definition is in part based on the definitions proposed in [157].

the data collected *so far*. From a modelling perspective, novelties are a type of anomaly whose behavior cannot be modeled.

2.2 Defining the Anomaly Detection Problem

In this section, we first define the canonical anomaly detection problem and its variants. Then, we define anomaly scores and the contamination factor. Finally, we explain how the contamination factor can be used to transform anomaly scores into binary predictions.

2.2.1 Canonical Anomaly Detection

A domain consists of an d-dimensional input space $\mathcal{X} := \mathbb{R}^d$, an output space $\mathcal{Y} := \{\texttt{normal} = -1, \texttt{anomaly} = +1\}$ indicating whether an observation is normal or anomalous, and an associated joint probability distribution p.

Definition 1 (Domain)**.** *A domain $\mathcal{D} := p(x, y)$ is defined as a joint probability distribution over the input-label space pair $\mathcal{X} \times \mathcal{Y}$.*

By sampling observations from a domain's distribution, we obtain a dataset.

Definition 2 (Dataset)**.** *A dataset $D = \{(x_i, y_i)\}_{i=1}^n$ is a set of n tuples with $x \in \mathcal{X}$ an input instance and $y \in \mathcal{Y}$ its label.*

X denotes the set of all instances in D. These instances are usually *independent and identically distributed* (i.i.d.), meaning that they are all randomly and independently sampled from the same underlying domain $\mathcal{D}$ according to the joint probability distribution. For some data types, such as time series data, this is not the case (see Section 2.5).

The canonical anomaly detection task is now defined as follows:

Problem 1 (Canonical anomaly detection)**.** *Given a dataset D, construct a model $h\colon \mathcal{X} \to \mathcal{Y}$ that maps each instance x in the dataset onto its true label y.*

The model h is part of the hypothesis space $\mathcal{H}$. Its output could either be an arbitrary score indicating the anomalousness of an instance or a binary label predicting whether the instance is an `anomaly` or `normal`.

Difference with Standard Binary Classification. While Problem 1 is reminiscent of the standard setting of learning a (binary) classifier, it is fundamentally different. In the standard setting, each of the different classes in $\mathcal{Y}$ has its own specific data distribution. All instances with a certain class label are generated from the distribution associated with that class. In the anomaly detection setting, however, *only* the normal instances are defined by a specific distribution. The anomalies' only characterization is that they are not part of the normal data distribution. The intuition is that anomalies cannot be predicted. For most real-world processes, if we were to restart the process and gather data anew, the anomalies would be completely different (e.g., a machine can fail in numerous, unpredictable ways). Thus, we cannot use the traditional classification algorithms to solve the anomaly detection problem.

2.2.2 Variants of the Canonical Problem

The most obvious dimension along along which the canonical problem can vary is the availability of label information. This is because labels are difficult to obtain, requiring the domain expert to devote substantial time and resources. Unsurprisingly, the most commonly encountered variant of the canonical problem is the *unsupervised anomaly detection* problem, meaning that the training dataset contains only unlabeled instances whose true label could belong to either class (normal or anomaly). To improve the detection models learned from such data, many practitioners usually exploit one additional piece of information, namely the contamination factor (see infra).

The polar opposite is *supervised anomaly detection*. In this case, the labels of all the instances in the training dataset are known when constructing the detection model. This setting is not often encountered in practice given the large costs associated with gathering labels. Consider that real-world anomalous behavior is by nature infrequent and often corresponds to real costs. For example, a manufacturer is not likely to let his machines fail just to generate an example of anomalous behavior. It is far more realistic to gather instances of normal behavior and use these to learn a model.

If the training dataset contains *only* normal instances, we are dealing with *one-class anomaly detection*. If, on the other hand, the training dataset contains a mix of normals and anomalies and we have access to a limited amount of labels, we are in the *semi-supervised anomaly detection* setting. These two settings are often conflated and used interchangeably.[3] However, they warrant

[3]For the remainder of this dissertation, when referencing semi-supervised anomaly detection, we explicitly assume this definition unless otherwise stated.

their own definition because they give rise to different strategies for learning a detection model.

In-sample Detection and Out-of-sample Prediction. One important, yet often-overlooked distinction relates to *how* one conducts anomaly detection. The most obvious way is to do *out-of-sample anomaly prediction*, which entails constructing a model using a training dataset and subsequently applying the model to an unseen test dataset in order to predict which instances are anomalous. This setting is usually encountered in the context of a supervised, semi-supervised, or one-class anomaly detection problem.

More common, however, is *in-sample anomaly detection*. In this case, there is no separate test set and the detection model is used to detect anomalous observations in the training dataset. It is akin to a *post-mortem* analysis of the data with the goal of filtering out the anomalies. As a result, many anomaly detection methods, such as [113, 13, 62], do not explicitly learn a predictive model from the data.

Whether to do in-sample detection or out-of-sample prediction should be a conscious design choice that depends on the specific application. For example, out-of-sample prediction naturally fits novelty detection.

Anomaly Scores and Binary Labels. The canonical anomaly detection problem entails constructing a model that predicts whether each instance is normal or anomalous. In case limited or no label information is available, it is unclear how such a model could be constructed in the traditional machine learning sense, i.e., by training a classifier to minimize a loss function. The solution is to reformulate the canonical problem to computing a real-valued *anomaly score* for each instance in the dataset that correlates with its anomalousness:

Definition 3 (Anomaly score function)**.** *Given a dataset* $D = \{(x_i, y_i)\}_{i=1}^{n}$*, an anomaly score function* $a\colon \mathcal{X} \to \mathbb{R}$ *maps each instance* x *in the dataset onto a real-valued anomaly score* $a(x)$*.*

If the score is computed without access to any label information, it is denoted a_u. Otherwise, it is denoted a_l. In practice, anomaly scores convey the "degree of anomalousness" of an instance and they are used to rank instances from (more) normal to (more) anomalous. To turn the scores into binary predictions, we can choose a threshold anomaly score above which the model assigns the label `anomaly` to an instance. This threshold can be learned from the data

in case label information is available.[4] If no labels are available, we need an additional piece of information to set this threshold.

Contamination Factor. The contamination factor $\gamma \in [0, 1]$ is the fraction of anomalous observations in a dataset D:

$$\gamma \coloneqq \frac{|D_a|}{|D|} = \frac{n_a}{n_a + n_n} \tag{2.1}$$

where $D_a = \{x | x \in D, y = 1\}$ is the subset of true anomalous instances in the dataset, and n_a and n_n are respectively the true number of anomalies and normals in the data. The contamination factor is closely related to the *class prior* in the field of positive and unlabeled (PU) learning [8]. The latter is the fraction of positive instances in a PU dataset and - if we consider the positive instances to be the anomalies - equal to the contamination factor.

Given the anomaly scores output by a model, the contamination factor is normally used to determine the *threshold* anomaly score, which is chosen to be $(1 - \gamma)^{\text{th}}$ percentile anomaly score. It is the anomaly score such that only a fraction of γ of the instances in the dataset have a higher anomaly score (the suspected anomalies).

While it is difficult to estimate the contamination factor from the data, some inspiration can be drawn from techniques developed in the context of PU class prior estimation [8]. Mostly, however, it is assumed that the contamination factor is provided by the expert.[5] Aside from the computation of the threshold score, the contamination factor is sometimes used to calibrate the anomaly scores in a meaningful way (see Chapter 5). From a statistical point-of-view, the contamination factor relates to the threshold on the probability distribution that separates anomalies (instances with low probability under the distribution) from normals (instances with high probability).

2.3 Taxonomy of Anomaly Types

This section discusses the common anomaly taxonomies. Such taxonomies guide the development of detection algorithms [21], because each anomaly type exhibits certain identifiable characteristics that can be exploited to distinguish it from normal observations. The current consensus on anomaly types, however, is somewhat limited and arbitrary.

[4] In case of one-class anomaly detection, all instances in the training dataset are normal and the threshold is usually set to the highest anomaly score observed in the training data.

[5] The contamination factor can be seen as "soft" label information.

Point, Collective, and Contextual Anomalies. Anomalies are commonly divided into: *point*, *collective*, and *contextual* anomalies [21]. The distinction between point and collective anomalies only makes sense within the context of non-i.i.d. data, such as time series data. A point anomaly is a single instance that is anomalous by itself. A collective anomaly refers to a set of instances that together make up the anomalous behavior. When the data are i.i.d., all anomalies are necessarily *point* anomalies because the instances are independent.

A contextual anomaly refers to an instance that is anomalous with respect to a relevant context, which can be a subset of instances or a subset of features [70]. The challenge of anomaly detection is to select the right context that helps identify an anomaly. Depending on the chosen context, every anomaly is contextual. It seems that this taxonomy is not meaningful in contrasting contextual anomalies versus other types of anomalies. It is also incomplete because it applies only to non-i.i.d. data.

Local, Global, and Dependency Anomalies. Another idea is to separate anomalies into *local*, *global*, and *dependency* anomalies [157, 130]. These types are best understood as different flavors of contextual anomalies. An instance should be compared to a relevant reference set or context to determine its anomalousness. The context can be a subset of the data or, in some cases, construed on a subspace of the feature space [140].

First, local anomalies are instances that are outlying "relative to their local neighborhoods [...]" [13]. Global anomalies are instances that are outlying relative to the full dataset. They are an "edge case" of local anomalies with the reference set equal to the full dataset, and are scattered all across the input space [130]. Finally, dependency anomalies are instances that violate the dependency structure underlying the data [130, 116]. They are identified by contrasting the observed value of an instance with the expected value based on the dependency structure. This bears resemblance to concept of counterfactual observations.

Formalizing Anomaly Types. To formalize anomaly types in a more principled way, [124] starts from the key insight that anomaly detection always requires comparing an observation with a set of related observations on some derived property. The authors identify three core concepts: the *property set*, the *reference set*, and the *model function* [124]. An instance's property set is a subset of the (transformed) dataset used to derive a specific property for that instance. It is also referred to as the instance's context. An instance's reference set is a subset of the (transformed) dataset against which the derived property is compared to determine its degree of anomalousness. Finally, an

instance's model function derives a property for an instance using its context set and the dataset. The resulting property is usually a real value. The typical (unsupervised) anomaly detection algorithm now proceeds in roughly three steps. First, compute an instance's property and reference set. Second, compute the property of each instance using the model function. Finally, compare an instance's property to the properties of the instances in its reference set to derive the final anomaly score.

The three key concepts allow us to formally describe well-known anomaly types. Global anomalies are identified by setting the reference set equal to the full dataset. Local anomalies, on the other hand, are identified by setting the reference set equal to a subset of the dataset. Viewed through this lens, it immediately becomes clear that local and global anomalies are not well-separated concepts. Rather, such anomalies lie along a "locality" spectrum and detecting them involves gradually changing the scope of the reference set.

2.4 Anomaly Detection Paradigms

In this section, we briefly discuss how each of the four anomaly detection problems are typically solved (see Section 2.2.2). This discussion is not exhaustive, but rather focuses on the concepts relevant to this dissertation.

2.4.1 Supervised Anomaly Detection

Supervised anomaly detection relies mostly on classification methods that learn a model from the labeled training data. Because anomalies are not generated by a specific mechanism, the training data likely do not represent all types of anomalous behavior that can arise. A classifier learned from such data, will arrive at a decision boundary that does not generalize well.

The situation is different when the anomalies do follow a well-defined distribution. In that case, classification algorithms can be used successfully provided they explicitly correct for the inherent class imbalance, i.e., the number of normal observations is far greater than the number of anomalies ($\gamma \ll 0.5$). Real-world examples are listed in [21].

The unpredictability of anomalous behavior as well as the issues introduced by existing methods to tackle class imbalance result in supervised anomaly detectors often having difficulties achieving acceptable performance levels [29]. Hence, a sensible strategy is to transform the supervised anomaly detection problem into a semi-supervised or one-class problem [55].

2.4.2 One-class Anomaly Detection

One-class anomaly detection methods are based on finding a minimal description of the normal instances. Anything that does not fit this description is predicted to be an anomaly. While different algorithms construct the description in different ways, they all assume that the descriptions are constructed from normal data only. If the dataset is contaminated with anomalies, the learned descriptions are too general and will mask true anomalies.

A well-known one-class classification method originally designed for novelty detection, is the *one-class support vector machine* (OC-SVM) [121]. This method tries to find a maximum-margin separation between the normal instances and the origin. The more recent *support vector data description* (SVDD) algorithm aims to find the smallest hypersphere that encloses all normal instances [135]. The *one-class neural network* algorithm designs a neural network architecture that uses the OC-SVM loss function to learn a model [20].

Recently, the advances in deep learning have led to algorithms that fuse deep neural networks with one-class classification, such as *deep support vector data description* (DEEP-SVDD) [120]. This algorithm replaces the transformation function of the original SVDD with a neural network that is able to learn a more appropriate and/or expressive transformation. The deep learning aspect of such approaches can be thought of as replacing an explicit feature construction step. The hope is that, given enough data, the neural network learns an appropriate representation.

2.4.3 Semi-supervised Anomaly Detection

Semi-supervised anomaly detection algorithms assume that some labels are available for the training data. The authors of [55] argued convincingly that any semi-supervised anomaly detector should be grounded on an unsupervised model. The available labels only serve to correct or reinforce the predictions of this unsupervised base detector. Such an approach also works without labels and does not make the mistake of treating anomaly detection as an imbalanced classification problem.

SVDD-NEG is an extension of the SVDD algorithm that can process labeled anomalies. It requires them to lie outside of the encompassing hypersphere by constraining the original optimization problem [135]. The *semi-supervised anomaly detection* (SSAD) algorithm extends the loss function of the SVDD algorithm to enforce that normal and anomalous instances fall respectively

inside and outside the enclosing hypersphere [55]. SSAD can handle both labeled normals and anomalies.

Another notable semi-supervised anomaly detection algorithm is the *hybrid isolation forest* (HIF) algorithm which extends the unsupervised *isolation forest* (IFOREST) anomaly score of each instance with a component reflecting the distance to known normal and anomalous instances [94].[6] The *feedback guided anomaly detection* (FGAD) algorithm also extends IFOREST to the semi-supervised setting [126]. It does so by tuning weights on the edges of the learned trees such that the final scores correspond better to the known labels.

2.4.4 Unsupervised Anomaly Detection

Without label information, it is impossible to train a classifier. Unsupervised anomaly detectors get around this issue by deriving an anomaly score for each instance based on some assumptions of what anomalies look like (see Definition 3). Because there is no one-size-fits-all unsupervised anomaly detector [80] - different detectors work well for different applications and datasets - a true zoo of algorithms has been developed during the last decades. We start with defining some relevant concepts we will need to understand the methods.

Five concepts that are foundational to many unsupervised detectors are: an instance's k-nearest neighbor, the distance of an instance to its k-nearest neighbor, the reachability distance between two instances, and an instance's neighborhoods.

Definition 4 (k-nearest neighbor)**.** *The k-nearest neighbor of x, denoted as k-NN$(x; D)$, is the instance in dataset D such that $k-1$ instances in D are closer to x according to some distance function.*

Definition 5 (k-distance)**.** *The* k-dist$(x; D)$ *of an instance x is the distance between x and its k-NN$(x; D)$.*

Definition 6 (Reachability distance)**.** *The* k-reach-dist$(x_1, x_2; D)$ *between two instances x_1 and x_2 is defined as* $\max\{\text{k-dist}(x_1; D), \delta(x_1, x_2)\}$.

Definition 7 (k-neighborhood)**.** *The k-neighborhood of an instance is the set of k nearest neighbors of x in the dataset D and is denoted as $N_k(x; D) = \{x_i | x_i \in D, \delta(x, x_i) \leq k\text{-}dist(x; D)\}$.*

Definition 8 (ϵ-neighborhood)**.** *The ϵ-neighborhood of an instance is the set of instances in the dataset D that are within a distance ϵ of x. It is denoted as $N_\epsilon(x; D) = \{x_i | x_i \in D, \delta(x, x_i) \leq \epsilon\}$.*

[6] See Section 2.4.4 for details on the IFOREST algorithm.

The distance function δ can be chosen freely in all cases. If we set $\epsilon = \text{k-dist}(x; D)$, both neighborhoods are identical. It is possible to derive other neighborhood types based on, for instance, graph adjacency or temporal context [124].

Statistical Models. The earliest approaches to unsupervised anomaly detection are firmly rooted in statistics [157, 58, 57]. One early example is Grubbs' test for detecting outliers [58], which assumes that the data follow a Gaussian distribution. This test iteratively removes one instance from the dataset and each time tests the null-hypothesis (there are no outliers in the data) versus the alternative hypothesis (there is exactly one outlier). These approaches are parametric because they impose a specific statistical model that describes the data.

Non-parametric models, on the other hand, do not impose a model. A good example is kernel density estimation which estimates the probability density distribution from which the data are generated [104]. Anomalies, then, are observations with a low probability under the found distribution.

Data Mining Models. Over the years, the statistical techniques have given way to data mining approaches for three reasons. First, strict statistical tests are limited to a pre-defined set of distributions that cannot always adequately model the observed data [74]. Second, it is not always clear *which* statistical test should be applied to identify the anomalies [74]. Finally, the earliest data mining approaches were designed primarily with scalability in mind, something that the contemporary statistical techniques could not always offer [157].

Numerous surveys have described in detail various subsets of unsupervised anomaly detection algorithms and compared them on different benchmarks [53, 15, 38, 41, 130]. The following paragraphs describe in detail some of the most popular and best-performing algorithms according to these surveys. However, no exhaustive comparison of all these algorithms on an agreed upon benchmark has been done to date.

Density-based Models. Density-based models attempt to connect the anomalousness of an instance with the place it occupies in the input space. They assume that instances in low-density regions of the input space should get a high anomaly score. Two well-known techniques are *k nearest neighbor outlier detection* (KNNO) [113] and *local outlier factor* (LOF) [13]. Both techniques can be understood as (simple) density estimates centred on an instance x.

KNNO assigns as anomaly score simply the k-distance of an instance:[7]

$$a_u(x) \coloneqq \frac{\text{k-dist}(x; D)}{\nu(x)} \tag{2.2}$$

where the denominator $\nu(x)$ serves to normalize the anomaly score. The KNNO algorithm sets $\nu(x) = 1$. This particular formulation illustrates that the "anomaly property" of an instance (its k-distance) is compared against all other instances in the dataset (the reference set). Because of this, KNNO is considered a global anomaly detector. It would be almost trivial to set $\nu(x)$ adaptively, yielding a local KNNO detector.

A common variant of KNNO is *weighted* KNNO [4]:

$$a_u(x) \coloneqq \frac{\frac{1}{n}\sum_{x_i \in N_k(x;D)} \delta(x, x_i)}{\nu(x)} \tag{2.3}$$

where the anomaly score of an instance is the average of the distances to its k nearest neighbors. Again, $\nu(x) = 1$. The choice of k has important ramifications for the performance of the model. Weighted KNNO aims to mitigate this somewhat by taking an average. However, empirical studies show that the latter is routinely outperformed by the original KNNO [15].

LOF computes an anomaly score by comparing the local reachability density (lrd_k) for an instance x with the lrd_k's of its k nearest neighbors:

$$\text{lrd}_k(x) \coloneqq \frac{|N_k(x; D)|}{\sum_{x_i \in N_k(x;D)} \text{k-reach-dist}(x, x_i; D)} \tag{2.4}$$

$$a_u(x) \coloneqq \frac{1}{|N_k(x; D)|} \sum_{x_i \in N_k(x;D)} \frac{\text{lrd}_k(x_i)}{\text{lrd}_k(x)}. \tag{2.5}$$

By changing components of these functions, e.g., replacing $N_k(x; D)$ by $N_\epsilon(x; D)$, variants of LOF are obtained [123]. An example is *simplified* LOF which simply replaces the k-reach-dist$(x, x_i; D)$ with $\delta(x_1, x_2)$ in the local reachability density. By rearranging the terms of Equation 2.5, we get:

$$a_u(x) = \left(\frac{\text{lrd}_k(x)}{\nu(x)}\right)^{-1} \tag{2.6}$$

$$\nu(x) = \frac{1}{|N_k(x; D)|} \sum_{x_i \in N_k(x;D)} \text{lrd}_k(x_i) \tag{2.7}$$

[7] To be exact, the left side of the equation should read $a_u(x; k)$ as the score is dependent on the choice of k. However, for the sake of readability, we will use the shorthand notation $a_u(x)$ in this and subsequent equations when it is obvious what the dependencies are.

This formulation clearly shows that LOF is a local anomaly detection method because it compares the anomaly property of each instance (its lrd_k) with that of its nearest neighbors; the reference set equals $N_k(x; D)$.

Cluster-based Models. Cluster-based models divide the data in a set of clusters. A subsequent anomaly score is computed by considering how an instance relates to its assigned cluster. The *cluster-based local outlier factor* (CBLOF) algorithm [62] starts by clustering the data with the k-means clustering algorithm, resulting in a set of k clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$. Each instance x is assigned to a cluster by minimizing some distance function δ between x and the cluster center. The cluster to which x belongs is denoted as $C(x)$ and its center as $c(x)$. Next, CBLOF uses a heuristic to divide the set of clusters into large (normal) clusters $\mathcal{C}_l$ and small (anomalous) clusters $\mathcal{C}_s$. Finally, it computes an anomaly score for each instance as follows:

$$a_u(x) \coloneqq \frac{|C(x)| \cdot \min_{C_j \in \mathcal{C}_l} \delta(x, C_j)}{\nu(x)} \tag{2.8}$$

with $\nu(x) = 1$, implying - despite what the name of the algorithm suggests - that CBLOF is not local. This becomes clear when the number of clusters is 1, in which case the anomaly score is simply the distance between each instance and the center of the data. To this end, [38] developed *unweighted* CBLOF which drops the $|C(x)|$ term from Equation 2.8. This does not change $\nu(x)$, however, so its locality is questionable.

The analysis of the CBLOF algorithm allows us make two interesting observations. First, the $C(x)$ term in Equation 2.8 serves as a crude way to assign an anomaly score to the context of an instance, assuming that an instance's cluster serves as its context. Anomalousness, it seems, is not only identified on the level of the instance, but also on the context level. Second, the solution to making CBLOF a truly local method is relatively clear from the formalization of Equation 2.8; it suffices to adopt an appropriate $\nu(x)$ based on the reference set of the instance (in this case, the cluster to which it belongs). For a more extensive follow-up of these ideas, see Chapter 5.

Isolation-based Models. Isolation-based anomaly detectors do not profile normal behavior, but instead assign an anomaly score based on how susceptible an instance is to being *isolated* using some data separation mechanism. By far the most well-known algorithm is the *isolation forest* (IFOREST) algorithm [85]. A more recent isolation-based detector is the *isolation using nearest neighbor ensemble* (INNE) algorithm [5].

Given a dataset, iFOREST builds an ensemble of t isolation trees. Each tree is build using a random subsample of size ψ of the data and recursively splits this subsample further using random axis-parallel partitions until each instance ends up in its own node or the tree height limit is reached. Finally, an anomaly score for each instance is computed as:

$$a_u(x) \coloneqq \frac{1}{t} \sum_{\text{trees}} (x \text{ leaf node height} + \nu(x \text{ leaf node size})) \tag{2.9}$$

$$\nu(\psi) \coloneqq 2H(\psi) - 2 \tag{2.10}$$

with $H(i)$ the i^{th} harmonic number. The algorithm is efficient and effective [41].

One downside of iFOREST is its reliance on axis-parallel splits (like most tree-based learners). [59] proposed an extension of the original algorithm that addresses this issue at the cost of increased computational complexity. INNE takes yet another road, building an ensemble of t detectors based on random subsamples of the data of size ψ. For each instance x in a subsample S a new hypersphere is constructed centred at x with a radius equal to the 1-dist(x, S). Intuitively, the larger this radius, the more isolated the instance is. INNE builds in total $t \times \psi$ hyperspheres. Then, during the evaluation phase, each instance is awarded an anomaly score based on the radii of the hyperspheres it belongs to. While INNE is computationally more complex than iFOREST, it is more versatile.

Other Model Blueprints. There exist numerous other ways to model unsupervised anomaly detectors. We highlight three examples. Probabilistic methods, such as *Gaussian mixture models*, estimate the probability density function of the dataset and identify anomalies by their low probability under the distribution [38]. Reconstruction-based methods, such as *autoencoders*, try to summarize the data using latent representations and identify anomalies as those instances that cannot be reconstructed well from this representation. Self-supervised methods, such as *attribute-wise learning for scoring outliers* [106], work by applying transformations to the (normal) data that are subsequently predicted using a classifier. Anomaly scores are based on the classifier's predictions [150].

2.5 Time Series Anomaly Detection

A time series is a series of chronologically ordered observations of a variable of interest. Most real-world processes naturally lend themselves to being recorded

in this format. For example, measuring bacterial growth cycles in wet labs or tracking the minute-to-minute performance of the equipment utilized in the Large Hadron Collider at CERN [36]. In industry, the increasing accessibility of sensor technology is resulting in huge volumes of data. The bottleneck has become analyzing this mass of time series data.

2.5.1 Time Series Data

The most common type of time series dataset is a continuous time series:

Definition 9 (Continuous time series)**.** *A continuous time series* $T = \{(t_1, v_1), \ldots, (t_n, v_n)\}$ *is a sequence of real-valued measurements where* $v_i \in \mathbb{R}$ *and each measurement has a distinct timestamp* t_i*.*

The individual measurements are non-i.i.d. since each value depends on the previously observed values. Although it is not required, we will assume in this dissertation that the continuous time series are sampled regularly, $t_{i+1} - t_i$ is constant, and do not contain missing values.

Another common type of time series dataset is the discrete event log:

Definition 10 (Discrete event log)**.** *A discrete event log* $E = \{(t_1, e_1), \ldots, (t_n, e_n)\}$ *is a sequence of discrete events where* $e_i \in \Upsilon$*, with* Υ *a finite domain of discrete event types, and each event has a distinct timestamp* t_i*.*

Unlike continuous time series, multiple events can co-occur at the same timestamp, i.e. $t_i \leq t_{i+1}$. Events can also occur sparsely. Practical examples of discrete event logs are, for example, the history of all credit card transactions of an individual or the history of all maintenance operations executed on a factory machine.

Mixed-Type Time Series Data. It is possible to monitor multiple variables of interest simultaneously, such as a company recording the temperature, pressure, and vibration of a gas-cooled chiller. A collection of N continuous time series and M event logs, all of length n, is called a *mixed-type time series dataset* $\mathcal{T}$. Thus, $\mathcal{T}$ has $M + N$ dimensions. Typical time series representations are special cases of this: when $N = 1$ and $M = 0$ it is a *univariate time series*; and when $N > 1$ and $M = 0$ it is a *multivariate time series*. A single time series in $\mathcal{T}$ is denoted as $\mathcal{T}^i$.

Time Series Segment.

Definition 11 (Time series segment)**.** *A time series segment $S^i_{t,l}$ is a contiguous subsequence of a time series $\mathcal{T}^i$ and contains all measurements for which $\{(t_i, v_i) \vee (t_i, e_i) | t \leq t_i < t + l\}$.*

Additionally, we can define a segment over all $N + M$ dimensions of $\mathcal{T}$ simultaneously using the same values for timestamp t and length l for all series in $\mathcal{T}$, regardless of whether they are continuous time series or discrete event logs. Such a segment is denoted as $S_{t,l}$.

Most time series anomaly detection algorithms make use of *sliding window* to extract segments. One incrementally slides a window of length l over the time series, each time extracting the measurements within the window as a separate segment. Usual values for the increment are 1, l, or $\frac{l}{2}$. In case l is fixed, it is referred to as a *fixed-size sliding window*. The set of all time series segments extracted in this manner from $\mathcal{T}$ is denoted as $\mathcal{S}$. The set of all time series segments extracted from time series $\mathcal{T}^i$ is denoted as $\mathcal{S}^i$

2.5.2 Time Series Anomaly Detection

Problem 2 (Time series anomaly detection)**.** *Given a time series dataset $\mathcal{T}$ with dimension $N + M \geq 1$, construct a model $h \colon (\mathbb{R}, \Upsilon) \to \mathcal{Y}$ that maps each time series segment $S_{t,l}$ onto its true label label y.*

The segments are extracted using the fixed-size sliding window method. The granularity of the model, formalized by the window length l, depends on the application. Some critical applications might require almost real-time prediction of anomalous behavior (e.g., monitoring a patient's vital signs), while in other applications lower granularities are acceptable (e.g., monitoring water consumption in a retail store). Because of the non-i.i.d. nature of time series, anomalous behavior can span multiple successive segments.

Just like the canonical anomaly detection problem can be further refined depending on the availability of labels, time series anomaly detection can be categorized as a unsupervised, semi-supervised, one-class, or supervised problem. The difference with i.i.d. data is how the labels are awarded. Either each individual measurement is labeled or the expert awards a single label to a larger segments of the time series. The latter entails that either the whole segment contains normal behavior or that some anomalous behavior occurred during the segment. Note that this resembles multiple-instance learning because it unknown which measurement(s) in the segment the label refers to [16].

The inherent structure in time series data introduces complexities that are not observed in the classic i.i.d. datasets that often form the basis of anomaly

detection research [21, 15]. Phenomenons such as concept drift, seasonality, change points, degradation and noise of the measuring equipment, are all challenges that are linked uniquely to the temporal aspect of time series [29].

2.5.3 Time Series Anomaly Detection Models

Time series anomaly detection models are roughly organized along three paradigms. We briefly familiarize the reader with each paradigm. Our goal is not to give an in-depth overview of all existing time series anomaly detection algorithms (see surveys by [29, 22]).

Residual-Based Models. The first major paradigm exploits the non-i.i.d. nature of time series data and the resulting predictability of a value at time t_i based on previously observed values [29, 51]. Normal behavior is predictable, anomalous behavior not so much. Algorithms in this paradigm follow the same blueprint. First, they predict the next measurement based on a model learned from historic data. Then, they compute the difference between the predicted and the actual measurement, i.e., the *residual*. Finally, the degree of difference corresponds to the degree of anomalousness. A threshold can be chosen such that residuals higher than the threshold are flagged as anomalies. After computing the residuals, some methods aggregate them over longer periods. This allows for the detection of anomalous behavior at different granularities.

The initial residual-based approaches were relatively uncomplicated. Statistical methods, for example, fit a distribution over the collection of all individual measurements. Any new measurement that deviates too much from the distribution mean an anomaly. This approach can be seen as residual-based with the first moment of the fitted distribution being the predicted value for each new observation. However, the temporal structure in the data is discarded entirely.

Regression models, on the other hand, do not discard the temporal structure. One of the earliest approaches was the parsimonious *auto-regressive moving average* (ARMA) method which predicts the observed value at each timestamp as a linear combination of previous observations and error terms [12]. More intricate versions of the ARMA approach have been proposed, such as ARIMA or SARMA [29]. Each new version extends ARMA to model additional time series phenomena, such as seasonality.

Recently, neural-network-based approaches have claimed their place as the state-of-the-art for time series forecasting (i.e., predicting future measurements) [154]. With the advent of deep learning, more complex network architectures have

been tried and tested, such as *autoencoders*, *convolutional neural networks*, *recurrent neural networks*, *long-short-term memory networks*, and *hierarchical temporal memory networks* [30, 19]. Each new architecture tackles additional or different problems brought on by the nature of time series data, such as the existence of long-term patterns.

Instead of predicting the exact value of new observations, we can instead focus on the quantifying the change between consecutive measurements. One can use techniques, such as *Markov models*, to model transition probabilities between observed states and identify the anomalies as unexpected (low probability) transitions.

Projection-Based Models. The second major paradigm exploits the (implicit) assumption that by dividing a time series into a collection of time series segments, the dependency structure is broken and the individual segments are independent among themselves. This allows the use of classic anomaly detection algorithms originally designed for i.i.d. data, to be applied directly to the time series segments. Each segment is treated as an input instance and the anomaly detection model assigns an anomaly score or label per segment. Mostly, the sliding window technique is used to divide a series [71].

Oftentimes, the time series segments are first projected onto a new representation before being fed into the detection model. The projection is likely to entail a dimensionality reduction, especially when the original time series are multivariate [29]. Projections range from collecting simple statistics to extracting complex patterns and features. In case the time series consists entirely of discrete events, techniques from the field of pattern mining can be applied [22]. It is also possible to learn a relevant representation using, for instance, autoencoders [29].

One advantage of projection-based detection is that it can easily accommodate contextual or label information. Indeed, if some labels are available for the extracted segments, one can plug in a semi-supervised anomaly detection technique (see Chapter 4).

Pattern-Based Models. Aside from the residual-based and projection-based techniques, there exist a number of pattern-based techniques. These operate under the assumption that normal behavior takes the shape of specific patterns or rules that can be learned/extracted from the time series. Anomalies are identified by their deviation from the found patterns or rules. The patterns or rules can be extracted with or without label information. If labels are available, they can be used to decide on the usefulness of certain patterns.

The MatrixProfile is an efficient algorithm for computing the euclidean distance between each fixed-length time series segment of T and its nearest neighbor, i.e., the non-overlapping segment in T to which the euclidean distance is minimal [152]. Anomalies are detected as the windows with the highest 1-distance. Despite being computationally efficient, this technique is limited in its effectiveness (see Chapter 3).

Multiple detection techniques based on *frequent pattern mining* exist [63, 64]. These techniques are developed with discrete event logs in mind. They use frequent pattern mining algorithms to extract itemsets (or sequential patterns) from the time series data, which requires dividing the time series into segments. Then, they compute an anomaly score per segment roughly based on how many of the set of found frequent patterns matches the observed data in that segment.

Conclusion. The three paradigms all exploit the structure inherent in time series data to detect periods of abnormal behavior. Often, some form of post-processing is necessary after computing the anomaly scores or binary labels. For example, residual-based methods compute a single anomaly score per timestamp. Aggregation of these scores over multiple timestamps might lead to more robust predictions that are easier to communicate to the domain expert.

Time series data, and especially data streams, are sometimes characterized by concept drift [18].[8] This phenomenon occurs when the underlying distribution of the data changes. Real-world conditions change over time and they affect the processes generating the time series data. For example, a restructuring of a retail store will change how it consumes gas, water, and electricty. If the time series data contain concept drift, the detection model should be adapted to the new distribution.

2.6 Transfer Learning and Anomaly Detection

A central idea in this dissertation is that real-world anomaly detection benefits from supervision provided by the domain expert. However, if the quantity of data collected and problems tackled increases, how can the domain expert keep up providing the necessary label information?

One answer is to look at transfer learning. Transfer learning aims to extend the generalizability of a classifier beyond the particular dataset it was trained on to different, yet related, datasets. For example, a bank that has build a machine learning system to identify credit card fraud in country A, is now looking to

[8] The terms *concept drift* and *concept shift* are sometimes used interchangeably.

expand its operations to country B. How can the bank adapt is machine learning system such that it generalizes to the new country with its different population?

In this section, we briefly discuss the important ideas behind transfer learning. The current literature is characterized by inconsistent terminology; different formalizations, conflicting nomenclature, and interchangeability of some definitions [76]. We will follow the conventions laid out in two recent surveys [75, 76]. We succinctly discuss the common transfer learning problem settings as well as common methodological paradigms. This will provide the backdrop against which transfer learning for anomaly detection is introduced.

2.6.1 Transfer Learning

Definition 1 defined a domain as a joint probability distribution over an input-label space pair. Different domains are characterized by different input spaces, label spaces, and probability distributions. Transfer learning always deals with a *source* domain $\mathcal{D}_S$, *from* which information is transferred, and a *target* domain $\mathcal{D}_T$, *to* which information is transferred. If there are more than two source or target domains, we are dealing with multi-domain transfer learning. In each domain, we execute a *task*. Formally, a task consists of a label space and decision function learned from data [156]. Transfer learning is now formalized as [156]:

Definition 12 (Transfer learning)**.** *Given a source domain $\mathcal{D}_S$ (and task), a target domain $\mathcal{D}_T$ (and task), and the corresponding datasets $D_S = \{(x_i, y_i)\}_{i=1}^n$ and $D_T = \{(x_i, y_i)\}_{i=1}^m$, improve the target domain model $h_T \colon \mathcal{X} \to \mathcal{Y}$ using the information implied in the source domain.*

Transductive transfer learning entails that source and target tasks are the same, while *inductive* means that the tasks are different, but related [102]. This terminology is vague, however, as other authors use the term transductive transfer when the goal is to predict the labels of the given target instances in D_T, and inductive transfer when the goal is to predict the labels of yet unseen target instances [76]. For our purposes, the task is always anomaly detection (both in source and target domain) and the goal is to construct a model capable of predicting the label of both seen and unseen target instances.

2.6.2 Domain Adaptation

Definition 12 is a general definition and many distilled versions of the problem have been introduced [156, 102, 75, 76]. Each version structures or constrains the problem in a such a way as to make it more tractable from a methodological

point-of-view. Here we stick with one particular instantiation of transfer learning, namely *domain adaptation*, which assumes that the input space, label space, and task of each domain is the same. Because of the input and label spaces being equal, different domains are now defined as different probability distributions over the *same* input-label space: $\mathcal{D}_S \coloneqq p_S(x, y)$ and $\mathcal{D}_T \coloneqq p_T(x, y)$.

A nuance embedded in the transfer learning definition with important practical repercussions, is the question of "which model are we changing?" We can try to *improve the target model* h_T using source information or we can try to *adapt the source model* h_S to better fit the target domain. The latter interpretation naturally links with the concept of *generalizability* of a machine learning model. Within the framework of empirical risk minimization, the goal is to learn a classifier that has a small generalization error. This means that it will mostly make correct predictions for unseen instances drawn from the same underlying distribution. Transfer learning, then, aims to minimize the *cross-domain* generalization error when the unseen instances are drawn from a different distribution. This point-of-view has resulted in some useful theoretical bounds for transfer learning, for example [115].

2.6.3 Domain Adaptation Settings

The domain adaptation problem varies along two dimensions: the availability of label information in source and target domain, and the nature of the differences between the source and target probability distributions.

The availability of labels in both source and target domain comes in four forms: fully supervised, semi-supervised, one-class, and no labels. This brings about 16 unique transfer learning scenarios. The difficulty of transfer (how successful we are at minimizing the cross-domain generalization error) increases when less labels are available. If the target domain is fully labeled, transfer learning only makes sense insofar the target domain contains only a few instances or is a biased subsample from the underlying distribution. Otherwise, we could just use the available labeled target data to train a target classifier that is perfectly capable of minimizing the generalization error. If there are no labels available in either domain, transfer learning becomes extremely challenging in two ways. First, it is unclear what information is useful to transfer. Second, additional assumptions about the structure of the domains are needed to facilitate transfer.

If the difficulty of the transfer learning problem increases, the probability of *negative transfer* also increases. Negative transfer entails that the transferred information hurts the performance of the target classifier, increasing the cross-domain generalization error. Negative transfer is likely to occur if the transfer

learning problem is unstructured, preventing the analyst from making useful assumptions that could constrain the problem.

One of the most insightful ways to categorize different domain adaptation settings is by looking at how the source and target domain differ [75, 76].

Prior Shift. Prior shift, also termed label shift or target shift, refers to a change in the class prior distributions between the source and target domain. Decomposing the joint probability distribution of each domain, we get $p_S(x, y) = p_S(x|y)p_S(y)$ for the source and $p_T(x, y) = p_T(x|y)p_T(y)$ for the target domain. Prior shift entails that $p_S(y) \neq p_T(y)$ while $p_S(x|y) = p_T(x|y)$. As an example, imagine a machine that uniformly produces two types of dark chocolate bars: one containing around 50 percent cacao solids and the other around 70 percent. During quality control, 5 percent of the bars are rejected because their cacao levels are too low or too high. At some point the company decides to increase the machine's output. Due to this speed-up, the machine cannot always keep up and now 7 percent of the bars has to be rejected. Thus, $p(y)$ has changed while $p(x|\text{normal bar})$ and $p(x|\text{anomalous bar})$ - with variable x representing the cacao solids percentage of a chocolate bar - are unaffected by the speed-up.

Covariate Shift. The often-studied covariate shift refers to a change in the marginal distributions between source and target domain. Decomposing the joint probability distribution, covariate shift corresponds to $p_S(x) \neq p_T(x)$ while the class-conditional distributions (the posteriors) are unchanged $p_S(y|x) = p_T(y|x)$. In our chocolate-bar example, 5 percent of the bars are sub par. The company expands to a different country whose customers prefer lower cacao levels in their chocolate. As a result, the machine starts producing more bars with 50 percent cacao than with 70 percent. Thus, the observed $p(x)$ changes, while $p(y|x)$, i.e., what constitutes a normal and anomalous bar, remains unaffected.

Concept Shift. Finally, concept shift, also termed concept drift or conditional shift, refers to a change in the class-conditional distributions between source and target domain. Decomposing the joint probability distribution, concept shift corresponds to $p_S(y|x) \neq p_T(y|x)$ while the marginal distributions are the same $p_S(x) = p_T(x)$. In our chocolate machine example, further changes in consumer taste have made the 70 percent cacao bars undesirable, the taste is simply too intense. However, the machine has not been updated to reflect this change. All the 70 percent bars are rejected during quality control. In other words, $p(y|x)$ has changed but not $p(x)$.

Combined Shifts. Each type of shift (prior, covariate, concept) entails a change in the joint probability distributions between the domains. Breaking down the exact nature of the change permits the development of transfer learning methods tailored to that change. In some applications, multiple shifts occur simultaneously, resulting in a harder domain adaptation problem. In that case, additional information is needed to do successful transfer learning. That could be, for example, label information in both domains to correct a concept shift, or other assumptions about the structure of the domains.

The question arises how to measure the divergence (or shift) between the source and target domains? Prominent examples of domain dissimilarity metrics are: the *Kullback-Leibler divergence* [139], the *Wasserstein distance* [76], the *Renyi divergence* [139], the *Symmetric Difference Hypothesis* [9], and the *Mean Maximum Discrepancy* [56]. These are *global* metrics, meaning that they gloss over small, local differences between the domains, instead capturing larger differences between the domains as a whole.

2.6.4 Domain Adaptation Solution Paradigms

The literature has come up with three paradigms to tackle the question *how* information can be transferred from source to target [76]. Extended overviews of the three paradigms can be found in recent surveys [75, 76].

Instance-based. Instance-based methods, also known as instance-transfer or sample-based methods, work by assigning weights to the source instances such that the resulting weighted source joint probability distribution aligns better with the target domain distribution. Weights are assigned depending on the nature of the difference between the source and target domain. For instance, if the there is only a prior shift, the weights can be chosen such that the class distributions in both domains match. In case of a covariate shift, the source instances that "overlap" with target instances receive higher weights than those that do not (assuming that the target data are within the support of the source distribution). The hope is that a classifier trained on the weighted source data will generalize better to the target domain.

There are three ways to compute the instance weights [76]. First, indirect estimators estimate the probability distributions (joint, marginal, or prior) in each domain directly, either through parametric methods or non-parametric methods, such as kernel density estimators. Then, they compute the weight of an instance as the ratio of the estimated distributions evaluated at that instance. Second, the weights can be estimated directly through framing the alignment

as an optimization problem with the aim to minimize divergence (using the aforementioned similarity metrics) between the domains. Finally, some direct weight estimators do not make use of optimization or optimize them jointly with the classifier.

Feature-based. Feature-based methods assume that there exists a transformation of the original input space (rotation, scaling, mapping...) such that in the transformed space, the source and target distributions are better aligned. The transformation tries to match either the marginal and/or conditional distributions depending on the type of domain shift. By reducing the domain dissimilarity, a model that is learned and applied in the transformed space, will achieve a lower cross-domain generalization error.

Various feature-based methods exist [76]. Subspace mapping methods, such as *subspace alignment* [47], try to find common structure in the domains while reducing the impact of domain-specific noise. Other techniques assume the existence of a *manifold* (a parameter-space in which each parameter setting leads to the generation of a specific domain) and try to find it. The best-case scenario is to find a representation that is domain-invariant, meaning that all the domain-specific noise is removed. This is the goal of methods such as *transfer component analysis* (TCA) [101].

Deep-learning-based methods for domain adaptation fit both the feature-based and the model-based paradigm. Through the use of autoencoders, latent representations for source and target domain can be found. By combining autoencoders with classifiers and domain critic networks (adversarial networks), it is possible to find a latent representation that aligns the source and target domain while also being discriminative with respect to the domain classes [49].

Model-based. Model-based techniques, also referred to as hyperparameter transfer, are based on the assumption that the models learned for related domains share parameters and/or prior distributions over hyperparameters [102]. This idea makes intuitive sense; if a particular model achieves a low generalization error for the source domain then that model, trained on the target data, is likely to have a low generalization error for the target domain. Another way of looking at it is that the source domain helps shrink the search space of all possible (hyper)parameters of the target model.

Deep-learning-based methods also fit this category [133]. A popular idea is to pre-train a deep network on a large source dataset and fine-tune the last layers on a (small) target dataset. If the data are related, such as image datasets, the early network layers extract domain-invariant features while the final layers

capture the domain-specific effects. Hence, transfer learning can help mitigate the artificial neural networks' need for large amounts of training data.

2.6.5 Domain Adaptation for Anomaly Detection

This dissertation proposes to tackle *label scarcity* in anomaly detection by employing transfer learning techniques, which forces us to think how transfer learning might play out in an anomaly detection scenario.

Because labels are scarce, the target dataset likely contains no label information, while the source dataset has label information for all or some instances. Regarding the types of shift that could occur between anomaly datasets; they are defined mostly by the infrequent and unpredictable nature of anomalies. Covariate shift is likely to happen due to different anomalous observations. For example, two machines on a factory floor can break in completely different ways. Normal observations can also differ between datasets. For example, if one of the machines starts producing a different batch of products.

Prior shift refers to a change in the proportion of anomalies in the data. It can occur, but is probably small. Anomalies are rare and make up only a tiny fraction of the dataset. Therefore, even though the absolute amount of anomalous instances can vary strongly between datasets, the relative proportions will always be small. Anomaly detection problems are almost always defined by a large class imbalance.[9]

Finally, concept shift entails that a fraction of the observations that are normal (anomalous) in the source domain become anomalous (normal) in the target domain. Whether concept shift occurs, depends on the specific application.

2.7 Evaluation of Anomaly Detection Models

This section briefly describes the most-used quantitative evaluation metrics for assessing the performance of anomaly detection models, as well as their strengths and weaknesses. More detailed overviews are provided by [15, 38].

Confusion matrix. The confusion matrix compares the *true* labels with *predicted* labels to obtain the number of *true positives* (true anomalies predicted

[9] Class imbalance might not be the right term, because it invokes the idea of two distinct data generating mechanisms underlying the normal and anomalous observations, while anomalies could be characterized by their *lack* of such a mechanism.

as anomaly), the number of *true negatives* (true normals predicted as normal), the number of *false positives* (true normals predicted as anomaly), and the number of *false negatives* (true anomalies predicted as normal).

Accuracy. Accuracy is defined as:

$$\text{accuracy} \coloneqq \frac{tn + tp}{tn + fp + fn + tp} = \frac{tn + tp}{n} \tag{2.11}$$

where n is the number of instances in the dataset.

The main problem underlying the evaluation of anomaly detection models, is the inherent class imbalance of anomaly detection datasets. The anomaly class is usually vastly underrepresented compared to the normal class. This effectively rules out using a metric such as *accuracy* because it is too susceptible to the imbalance. For example, if a dataset contains 5% anomalous observations and a detection model always predicts normal, its accuracy is 95%. Obviously, the model did not learn anything though.

The impact of class imbalance on accuracy can be easily understood by rewriting Equation 2.11 as:

$$\text{accuracy} = \overbrace{\frac{tn + fp}{n}}^{w_{\text{norm}}} \cdot \underbrace{\frac{tn}{tn + fp}}_{\text{TNR}} + \overbrace{\frac{fn + tp}{n}}^{w_{\text{anom}}} \cdot \underbrace{\frac{tp}{fn + tp}}_{\text{recall}} \tag{2.12}$$

Accuracy is a weighted sum of the *true negative rate* (TNR) and the *recall* (R) indicating the ability of the model to correctly label respectively the normals and the anomalies. If there is no class imbalance, the weight terms are equal because the true number of normals $(tn + fp)$ and anomalies $(fn + tp)$ are equal. Thus, a model that always predicts `normal` (TNR $= 1$ and R $= 0$) has an accuracy of 0.5. In case of class imbalance, however, it is easy to see that $w_{\text{norm}} \gg w_{\text{anom}}$. As a result, the recall obtained by a model does not matter as long as the TNR is high enough. In the extreme case of always predicting the normal label, the accuracy becomes equal to the class imbalance.

F1 score. Another often-used metric is the *F1 score*. It is the harmonic mean between *precision* (P) and *recall*. The F1 score is defined as:

$$\text{F1 score} \coloneqq \frac{tp}{tp + \frac{1}{2}(fp + fn)}. \tag{2.13}$$

In in-sample anomaly detection (there is no test set), it is standard practice to use the contamination factor γ to convert the anomaly scores of a model into a

binary labeling. Supposing that the dataset contains n_n *true* normal and n_a *true* anomalous instances; if $\gamma = \frac{n_a}{n}$ and the model predicts $\gamma \cdot n$ instances to be anomalous then the number of *predicted* anomalies is also n_a and the number of *predicted* normals is n_n. Because $fn + tp = n_a$ and $fp + tp = n_a$, we derive that $fp = fn$. The F1 score can now be rewritten in two ways:

$$\text{F1 score} = \frac{tp}{tp + fp} = \text{recall} \tag{2.14}$$

$$\text{F1 score} = \frac{tp}{tp + fn} = \text{precision} \tag{2.15}$$

We can conclude that the F1 score does not yield additional information over reporting precision or recall in case of in-sample anomaly detection using the correct contamination factor.

Area Under the Curve. Two measures that are often used in anomaly detection are the *area under the receiver operating characteristic* (AUROC) and the *area under the precision recall curve* (AUPRC), also referred to as *average precision* (AP) [53, 38]. Each of these measures requires a real-valued anomaly score or probability in the range $0 - 1$ for each instance in the test set. This allows to rank the instances from normal to anomalous.

The AUROC is computed by measuring the area under the receiver operating characteristic (ROC). More intuitively, it corresponds to the probability that the model ranks a randomly chosen true anomaly higher than a randomly chosen true normal.[10] Similarly, the AUPRC is computed by measuring the area under the precision recall (PR) curve. Because interpolation is tricky in the precision-recall space [32], computing the AUPRC is not straightforward. A common approximation is to compute the AP:

$$\text{AP} \coloneqq \sum_n (\text{R}_n - \text{R}_{n-1})\text{P}_n \tag{2.16}$$

where R_n and P_n are respectively the recall and precision obtained by setting the threshold for converting real-valued anomaly scores into binary predictions equal to the score of the instance at the n^{th} position in the ranking. By summing over all possible ranks (and thus thresholds), we obtain the average precision.

It is argued that PR curves - and by extension the AUPRC - are more informative than ROC curves when there is class imbalance in the data [32]. To see why this might be the case, consider that the ROC curve plots the recall against the

[10] This can be easily verified experimentally.

false positive rate (FPR). The PR curve, on the other hand, plots recall against precision. These quantities are defined as:

$$\mathrm{R} \coloneqq \frac{tp}{tp + fn} = \frac{tp}{n_a} \tag{2.17}$$

$$\mathrm{FPR} \coloneqq \frac{fp}{fp + tn} = \frac{fp}{n_n} \tag{2.18}$$

$$\mathrm{P} \coloneqq \frac{tp}{tp + fp}. \tag{2.19}$$

with n_n and n_a respectively the true number of normals and anomalies in the data. Because the R and FPR are computed over the anomalies and normals independently, a changing class imbalance characterized by a changing ratio between n_n and n_a, does not affect the computation of the R and FPR (assuming that the model does not start making predictions differently). Precision, on the other hand, *is* affected by a changing class imbalance because of the denominator computed over both normals (fp) and anomalies (tp). If class imbalance increases, the number of fp will increase disproportionally versus the number of tp such that precision will drop. Hence, it seems that AUROC is less affected by a changing class imbalance than AUPRC. This might just be an indication that ROC curves are overly optimistic under class imbalance [32], however, when comparing the performance of an algorithm over multiple datasets with different class imbalances, the high susceptibility of precision to this skew will make it difficult to justifiably compare the AUPRC of a single model over multiple datasets. Let alone, multiple models.

Chapter 3

Detecting Absent Pattern Anomalies in Continuous Time Series

Given its large applicational potential, one important task in the field of anomaly detection is time series anomaly detection. The goal is to identify portions of the data characterized by the presence of unexpected or abnormal behavior [21]. Existing approaches focus on analyzing whether the currently observed behavior differs from previously seen, normal behavior. Figure 3.1b illustrates the canonical anomaly detection problem where the grey-shaded region highlights a pattern (i.e., a collection of points) that substantially differs from the other patterns present in the data such as the highlighted green bell-shaped pattern in Figure 3.1a. In contrast, this chapter addresses detecting a radically different type of anomalous behavior: the *absence* of a pattern. The red-shaded region in Figure 3.1c shows a portion of the time series where we could reasonable expect to see an occurrence of the bell-shaped pattern. Detecting such an anomaly is challenging as the observed measurements in the red-shaded region also correspond to typical, normal behavior in the time series.

In many real-world use cases, absent patterns often correspond to significant anomalies. Figure 3.2 shows an illustrative real-world resource monitoring use case. It involves monitoring a retail store's water usage over several weeks. The store's water distribution system contains a device that automatically performs self-cleaning. Although the water usage during a self-cleaning action is always slightly different, the highlighted segments in Figure 3.2 show that

a recognizable pattern emerges in the data. A failure of the cleaning action neither causes any immediate problems nor any noticeable effects in the water usage data. However, after a while a sudden breakdown will occur that halts the water distribution and a large amount of water will be disposed through an unmonitored valve. Other real-world use cases, for instance, are found in network monitoring where CPU load is used to monitor data backup operations and recurring fail-over system tests.

Despite the prevalence of this type of anomaly in practice, little attention has been paid toward developing algorithms that can detect the suspicious absence of a pattern. This chapter formalizes the problem of absent pattern detection and introduces a tested algorithm for identifying absent patterns. The standard anomaly detection approaches struggle in this setting as it *violates the common assumption of anomaly detection* that frequent behavior is normal and infrequent behavior is abnormal. By definition, the most common behavior of the sporadically occurring pattern is absence. Furthermore, an absent pattern is not necessarily replaced by anomalous behavior. A successful approach to this task must address two challenging sub-problems:

1. **Identify the pattern of interest and all its occurrences.** This poses two challenges. First, the pattern does not appear all that frequently, in the Figure 3.2 time series snippet of 64 days it only occurs six times. In contrast, most pattern mining algorithms focus on identifying frequent patterns. Additionally, there will be many other patterns in the data that occur more frequently than the pattern of interest. Second, the pattern appears at slightly different times during the day and its shape can vary.

2. **Learn how the pattern's occurrence is distributed within the time series.** In Figure 3.2, it is not immediately obvious what characteristics of the data are predictive of the pattern's occurrence. Its appearance does not follow a fixed time or interval schedule, there are gaps of three days, nine days, ten days, etc. between occurrences of the pattern. Similarly, the immediate context (i.e., the previous or next's days values) are not predictive of the pattern's occurrence. *This is a more challenging variation of contextual anomaly detection* than is seen in most research because there is no direct link between the time series and the context.

Clearly, this is a problem setting that requires a data-driven solution. For instance, the retail company needs to monitor over 500 stores and the behavior of each store's cleaning device is unique and might change over time. Furthermore, feedback is given directly by local domain experts and should be processed without them requiring intimate knowledge of the approach.

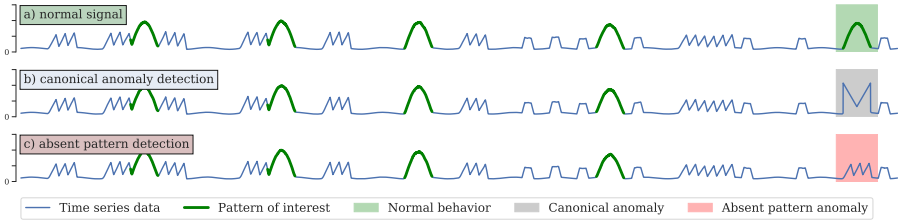


Figure 3.1: The figure contrasts canonical anomaly detection with *absent pattern detection* on a fictional time series fragment. The **top** plot shows the fragment without any anomalies. The **middle** plot shows the fragment with a single anomaly which is picked up by canonical detection approaches because it looks different from the rest of the observed behavior, The **bottom** plot shows the fragment with an *absent pattern anomaly*. None of the classic approaches can detect it because the behavior observed in its place is not anomalous.

Contributions of this Chapter

The work outlined in this chapter, makes the following five contributions:

The **first** contribution is the introduction of a novel type of time series anomaly, the *absent pattern*, and the formalization of the problem of absent pattern detection.

The **second** contribution is a strategy for detecting all occurrences of a specific pattern in a time series. The strategy consists of the user labeling a small number of occurrences and a PU learning model trained on said occurrences capable of identifying the remaining unlabeled occurrences.

The **third** contribution is FZapPa (Finding Zapped Patterns): an algorithm for identifying when a pattern occurrence should have occurred but did not. It makes the assumption that any pattern occurrence only depends on the previous occurrence and uses the Weibull distribution to model these occurrences. It automatically selects the relevant context descriptor to model the occurrences.

The **fourth** contribution is an extensive experimental evaluation of FZapPa on three real-world water usage datasets and three real-world power usage datasets. FZapPa is compared against the state-of-the-art time series anomaly detection algorithms and a detailed analysis is presented highlighting how existing methods are not adequate to identify *absent patterns*.

The **final** contribution is an implementation of FZapPa available at: `https://github.com/Vincent-Vercruyssen/absent_pattern_detection`

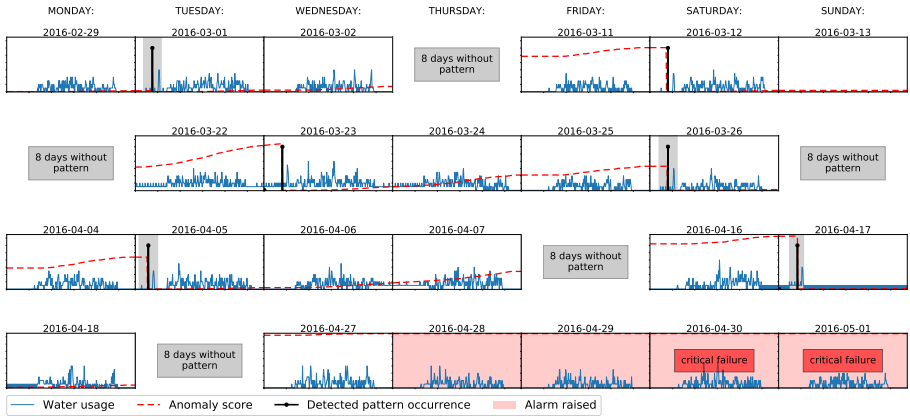


Figure 3.2: Water usage in one retail store over a period of three months in 2016. The store has an automatically-operated water softener installed to control the water quality. Its self-cleaning operation results in a signature pattern that can be observed in the data. The highlighted segments show all days with such a pattern. The softener works fine until April 17 and we observe the signature pattern on that day. The following two weeks, however, the softener fails to perform self-cleaning and the pattern is not observed. Finally, the system breaks down 12 days after the last self-cleaning operation. We apply FZapPa to detect the absent pattern occurrence between the April 18 and May 1. The black spikes indicate where the PU-classifier has detected an occurrence of the pattern. The dashed, red curve shows the anomaly score that is computed by FZapPa at each time step. Two days before the critical failure, the anomaly score is at its maximal value and FZapPa issues a warning for an imminent failure. Note that immediately after a pattern occurrence is detected in the data, FZapPa's anomaly score drops to 0, only to increase again during the periods the pattern is absent.

Scientific Publications

The content of this chapter is based on the following publication [144]:

V. Vercruyssen, W. Meert, and J. Davis (2020). "Now you see it, now you don't" Detecting Suspicious Pattern Absences in Continuous Time Series. In *Proceedings of the 2020 SIAM International Conference on Data Mining (SDM)*, Cincinnati (US), pp.127-135.

Problem Statement

The problem addressed in this chapter is formalized as follows:

Given: A continuous time series $T = \{(t_i, v_1), \ldots, (t_n, v_n)\}$ that contains a sporadically reoccurring pattern $\mathcal{P}$, where a pattern is a specific shape of length m with $m \ll n$.

Do: Identify anomalous periods of T. That is, find places in T where the pattern $\mathcal{P}$ is expected to occur but does not.

Sporadically reoccurring means that $\mathcal{P}$ occurs multiple times in T but that the number of times it occurs is much smaller than $\lfloor \frac{n}{m} \rfloor$.

3.1 Methodology

Outline of FZapPa. FZapPa[1] partitions the time series into a sequence of overlapping segments $S_{1,l}, S_{2,l}, \ldots, S_{n-l,l}$, where segment $S_{i,l}$ is a contiguous subsequence of T starting at time t_i with length l and $l \ll n$. For each segment $S_{i,l}$, FZapPa computes an anomaly score, which is the probability that a pattern is expected to appear in $S_{i,l}$ but is absent given information about its previous occurrence and the current context:

$$a_u(S_{i,l}) = p(absent_i | S_{i,l}, \ldots, S_{j,l}, \ldots, S_{k,l}) \tag{3.1}$$

where $S_{k,l}$ is most recent segment where the pattern was detected and $i > j > k$. A high score indicates a period that is likely to be anomalous. This probability can be communicated to the domain expert or thresholded to a specific percentile of the distribution to generate a discrete alarm. The threshold can be tuned from feedback or set by the expert.

FZapPa has three components. First, it must be able to detect occurrences of the pattern in T. It views this as a PU learning problem, because this step has only access to a few known example occurrences of the pattern. The trained PU model is used to detect the remaining occurrences. Second, it automatically learns a context of T that is fed to a Weibull distribution to model the distribution of pattern occurrences within T. Finally, at test time, FZapPa uses the trained context and the Weibull distribution to assign an anomaly score to the current segment, which can be communicated to the expert or thresholded to generate a discrete prediction (i.e., anomaly or no anomaly).

[1] We loosely use the term *zapped* instead of *absent*.

FZapPa exploits the flexible supervision of the expert, namely, a few labeled occurrences of the pattern to identify suspicious absences of the pattern.

The problem statement refers to a single continuous time series T. Theoretically, however, the outlined approach also extends to a multivariate continuous time series $\mathcal{T}$.

3.1.1 Detecting the Pattern Occurrences

Detecting occurrences of patterns in the data is important because:

1. At training time, we need to identify all occurrences of the pattern in the training data in order to accurately model its distribution in the (time series) data.
2. At test time, it is paramount to recognize each pattern occurrence to avoid raising a false alarm, i.e., reporting an absent pattern when the pattern was in fact present.

However, the pattern's relative rarity presents two complications. One, automatically discovering which pattern in the data is relevant would be challenging as the relevant pattern does not correspond to traditional measures of interestingness such as frequency. Two, even given examples of the pattern, its rarity, its subtle variations and its similarity to other data means that the standard approach of detecting an occurrence by computing the current segment's distance to a prototypical exemplar of the pattern is likely to produce false detections.

To address these problems, we view pattern detection as a PU learning problem [8]. The goal is train a classifier to predict whether or not a given segment of the time series contains the pattern. We assume that an expert has provided a small number of segments that are known to contain the pattern, which can be viewed as positive examples. The labeled positive patterns are assumed to adhere to the selected completely at random assumption, which is the standard assumption made in PU learning [8]. However, for all other segments it is unknown whether the pattern is present. That is, each one may or may not contain the pattern, which gives rise to the PU setting.

For our pattern detector, we use the inductive bagging Svm classifier [98], which is designed for PU data. We construct training data by generating $n - \hat{m}$ time series segments from T with $\hat{m}$ the average length of the known patterns using a fixed-size sliding window. We convert each segment into a feature vector containing the three types of features:

Summary statistics: max, min, mean, median, standard deviation, skewness, kurtosis, and entropy.

Time-based features: two cyclical features that indicate the time-of-day: $\sin(2\pi t_i/24)$ and $\cos(2\pi t_i/24)$, where t_i is the hour in which the segment starts. This ensures for example that a segment at 1am is equidistant to both a segment at 11pm and a segment at 3am.

Shape features: the known pattern occurrences provide valuable information about the shape of the underlying pattern $\mathcal{P}$. Therefore, we construct one feature for each known pattern occurrence whose value is the segment's *dynamic time warping* (DTW) distance to the pattern. This captures the segment's similarity to the known occurrences while allowing a certain amount of deviation [10].

A segment containing a known pattern occurrence is labeled as a positive example, the remaining segments are unlabeled. Importantly, the PU classifier cannot be used for performing anomaly detection, it can merely predict if the pattern is present in a segment. Simply because a pattern is not detected in a segment does not mean that the segment is anomalous: the vast majority of segments are not expected to contain the pattern.

The learned model makes a prediction for each segment in the training data, and our final set of detected pattern occurrences consists of all segments that are predicted to belong to the positive class. The use of a sliding window to obtain the time series segments results in the classifier predicting a pattern occurrence in multiple subsequent segments of T. We require the classifier to predict an occurrence in at least q consecutive segments of T to achieve a high confidence in a pattern detection and eliminate any spurious matching. The value of q is set in a data-driven way to be the maximum possible value at which all known pattern occurrences are still detected.

3.1.2 Modeling the Pattern Occurrences

Predicting when a pattern is absent requires learning how the pattern occurrences are distributed within T. FZapPa attempts to learn a context based on (indirect) properties of T that is predictive of when the pattern will occur in the data. Building the model proceeds in two steps. First, it searches for a novel descriptor, or feature, $\mathcal{X}$ of T that describes the context when the pattern appears. We can rewrite the right-hand side of Equation 3.1:

$$p(absent_i|\mathcal{X} = x_i) \quad \text{with} \quad x_i = f(S_{i,l}, \ldots, S_{j,l}, \ldots, S_{k,l}).$$

The descriptor $\mathcal{X}$ is defined by a function f that has as inputs the previous segments and as output the value x. This function can be instantiated as directly observing a value (e.g., the first measurement $x_i = v_i$ or time stamp $x_i = t_i$) or a constructed descriptor. In this chapter, we consider descriptors that can be constructed from the segments since the last occurrence of the pattern in segment $S_{k,l}$ up to the current segment $S_{i,l}$. Specifically, we consider the cumulative sum $x_i = \sum_{k \leq j \leq i} v_j$ and the time interval $x_i = t_i - t_k$. This naturally generalizes to considering all previous pattern occurrences.

Second, for each type of descriptor $\mathcal{X}$, we take the values x from the segments in which the pattern occurs and fit a single Weibull probability density function on the descriptive descriptor values x:

$$f(x; \alpha, \beta) = \begin{cases} \frac{\beta}{\alpha} \left(\frac{x}{\alpha}\right)^{\beta-1} e^{-(x/\alpha)^{\beta}} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where $\alpha > 0$ is the shape parameter of the distribution and β is the scale parameter. The parameters α and β are determined by minimizing the negative log-likelihood over all x values. The Weibull distribution has a number of useful properties for modeling the distribution of the pattern [99]. It offers a flexible shape parameter that can represent different types of behavior (e.g., both decreasing and increasing occurrence rates over x), and it takes into account that negative values cannot occur. These properties allow modeling of a variety of life behaviors, which makes it also a popular distribution for survival analysis and failure analysis.

Finally, we select the descriptor $\mathcal{X}^*$ with the corresponding density function that results in the best fit. Note that in contrast to other popular uses of the Weibull distribution, in our approach the variable $\mathcal{X}^*$ is not necessarily time. This allows FZapPa to identify the context that best explains the pattern occurrences (as measured by the best fit).

3.1.3 Predicting the Pattern Occurrences

At test time, FZapPa uses the learned bagging Svm PU classifier to predict whether the current segment ending at time point t_i contains the pattern. If it does, its anomaly score is zero. Otherwise, it uses the learned Weibull distribution and corresponding best-fitting descriptor to compute the probability that an occurrence of the pattern should have been observed by now, which can be converted into an anomaly score.

Given the best-fitting $f(x^*; \alpha, \beta)$ and the corresponding best descriptor $\mathcal{X}^*$, computing the anomaly score proceeds in three steps. First, for a given time

point t_i, we find its corresponding time series interval as the interval between t_i and the timestamp of the last known pattern occurrence. Second, we compute the descriptive value for this interval x_i^*. Third, we compute the probability of observing the descriptive value under the best-fitting model $p(\mathcal{X}^* < x_i^*) = F(x_i^*; \alpha, \beta)$, where $F(x^*; \alpha, \beta)$ is the cumulative distribution function for the Weibull distribution:

$$F(x^*; \alpha, \beta) = 1 - e^{-(x^*/\alpha)^\beta}.$$

Intuitively, this captures how unusual it is to observe x_i^* under the fitted model. The periods characterized by a high probability are the periods where our model suspects a pattern occurrence. Note that, as long as no new pattern occurrence is observed, $P(\mathcal{X}^* < x_i^*)$ can only increase over time because of the cumulative nature of the descriptors. This is desirable in practice as it corresponds to a model that gradually becomes more confident that a pattern is absent. Now, this probability can be used to compute Equation 3.1:

$$p(absent_i | \mathcal{X}^* = x_i^*) = 1 - P(\mathcal{X}^* < x_i^*).$$

In Figure 3.2, the dashed red line shows the anomaly score after fitting the model on the earlier-detected pattern occurrences. The descriptor selected by the model in this case is the cumulative water usage. During the final days of April, the anomaly score surpasses the threshold and an alarm goes off, two days before the critical failure.

3.2 Related Work

This section briefly comments the most related work within the context of this specific chapter. For more details on the discussed methods, see Chapter 2.

Standard Anomaly Detection. Although successful in many settings, the standard anomaly detection techniques are not particularly well-suited to the absent pattern setting because when the (sporadically occurring) pattern is absent, typically (frequent) normal behavior is observed in its place. Moreover, said techniques (e.g., KNNO, LOF, IFOREST...) were not originally designed for time series data.

Finding absent patterns can be thought of as identifying contextual anomalies, because the absence of a pattern depends on contextual factors (e.g., the time passed since the previous occurrence). Thus, one approach would be to rely on the expert hand-crafting the context, but this is often not realistic.

Time Series Anomaly Detection. A subset of dedicated time series anomaly detection algorithms focuses on detecting deviating temporal patterns. HotSax and its faster variant Wat [14] compute a discord score as the distance of a time series segment to the nearest non-self matching neighbor. Wcad [72] finds the most deviating segment according to a compression-based distance metric. A recent improvement is MatrixProfile [152], which is an *all-pairs-similarity-search* technique for time series. To detect anomalies with MatrixProfile, one selects the segments with the lowest similarity to the rest of the time series data. This method subsumes both HotSax and Wat. These techniques suffer from the same issues as the standard anomaly detection techniques: they only detect anomalous behavior that differs from typical or common behaviors.

Arrival Time Modelling. One technique akin to time series anomaly detection detects deviations in arrival times or frequency of arrival [40]. These methods are often used for intrusion detection or traffic analysis where events occur very frequently. Counting or frequency-based methods fail to detect small or moderate changes in behavior without long accumulation periods. A better approach is to look at the arrival times of patterns. The main assumption is that the time since the last event may be irregular, but it is not unbounded. An anomaly is then reported some time period beyond what is reasonable to expect for the next pattern. FZapPa is inspired by this idea but generalizes beyond interval times to learn the appropriate context that is predictive of when to expect a pattern. Additionally, they take discrete events as input and cannot handle continuous time series directly.

Health Monitoring. Prognostics involves predicting the time progression of a specific failure mode from its incipience to the time of component failure, thus the remaining useful lifetime [127]. Our approach relates to this line of research since the absence of a pattern might be an indication of or lead to a failure. We can differentiate between three types of models and data: (1) similarity models require run-to-failure history, (2) degradation models require known failure thresholds, and (3) survival models require life time data [127, 81]. All these approaches assume that the observed behavior can be directly associated with degradation and thus can be used as an indication of the time until failure. Furthermore they require a large number of labeled examples to learn from. Both assumptions do not hold in our setting.

Temporal Point Processes. Most adjacent to this work, is the study of temporal point processes. Given a stream of *discrete events* of various types (e.g., different consumers buying goods on Amazon), the goal is to model the

process generating the events. In [95], the authors model event streams with neural Hawkes processes. More recently, [96] extends this work to model the stream of events both forwards and backwards in time. Then, they impute missing event sequences with the learned model. While these techniques are certainly applicable in our setting to model the distribution of the pattern occurrences, they do require access to a sequence of discrete events. We face the added difficulty that we do not have access to these, rather our events are not discrete but slightly varying subsequences of a continuous time series and we only know for a handful of them when they occur.

3.3 Experiments

The empirical evaluation of FZapPa focuses on three questions:

Q1: how does FZapPa compare against existing anomaly detection techniques at detecting absent pattern occurrences?

Q2: can FZapPa automatically determine the appropriate context (i.e., the best description $\mathcal{X}^*$) that is predictive of the pattern's occurrence?

Q3: how does using our PU-learning based pattern detector affect FZapPa's performance compared to using standard pattern detectors?

3.3.1 Experimental Setup

Compared Approaches. We experimentally compare FZapPa with eight state-of-the-art anomaly detectors, divided into three categories:

Unsupervised anomaly detectors. We consider three state-of-the-art anomaly detection techniques: Knno [113], Lof [13], and iForest [85]. We also consider MatrixProfile, a state-of-the-art time series anomaly detection technique [152].

Semi-supervised anomaly detectors. Ssad is a semi-supervised extension of the one-class support vector machine algorithm for anomaly detection [55] and Ssdo updates a prior unsupervised anomaly score by propagating label information (see Chapter 5).

Absent pattern detectors. AverageMissing is a baseline that predicts absent pattern occurrences based on the average inter-arrival time between occurrences [40]. AverageMissing only works in our setting if we allow

it to use the pattern detection subroutine of FZapPa, because it was originally intended to work with discrete event streams [40].

Resource Usage Data. We use six real-world datasets, three water usage datasets obtained from a company and three power usage datasets from the UCI Irvine Machine Learning repository [39]. Each water dataset records water usage in a retail store, measured every five minutes, over the course of two years. In each store, a water softener is installed that controls the water hardness. To prevent malfunctioning, the softener regularly executes a self-cleaning operation, resulting in a signature pattern that can be observed in the water usage data (see Figure 3.2). A failure of this self-cleaning action is not noticeable in the data, except the absence of the pattern it causes, but it leads to a degradation of the system later in time.

Each power usage dataset records the power usage of a French household, recorded every minute, over the course of three years.[2] The UCI dataset contains the power usage (in kilowatt-hour) recorded every minute, over the course of 47 months (resulting in a total of $2,075,259$ measurements). In our experiments, we only use the first three full years of the recorded data. About 0.53% of the said data is missing and the missing values are replaced with base noise, i.e., the power consumption level when no appliances are being used in the house. Finally, the data are subsampled to obtain a measurement every five minutes. We inject a recurring pattern in the data based on a randomly selected shape from the *ItalyPowerDemand* UCR dataset.[3] Each pattern occurrence is a slightly altered version of this shape scaled to a length of around two hours. Two versions of each dataset are constructed: one where the occurrences are distributed in T based on time, and one where the distribution is based on cumulative usage. In the experiments, performance results are averaged over the two versions.

Experimental Setup. The experiment simulates how the absence detection model is used in real-world applications. For each of the six time series datasets T, the following steps are repeated for increasing values of i: (1) extract the first i months from T as training data and month $i+1$ as test data; (2) remove the final pattern occurrence from the test data to simulate an absent pattern; (3) randomly label 10 pattern occurrences in the training data and train the model; (4) detect the absent pattern occurrences in the test data. The first value of i is 6 and the experiment continues until the end of T is reached.

[2]Data available at: `https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption`

[3]Data available at: `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`

One week after the absent pattern occurrence, the test set ends, mimicking an intervention. Each experiment is repeated five times with different initializations of the expert-annotated patterns and the results are averaged over all runs.

Evaluation. The goal is to detect each day with an absent pattern occurrence (maximize recall) while reducing the number of days with a false alarm (maximize precision) given that an alarm requires an intervention and thus bears a physical cost. Each method outputs an anomaly score for each segment of T in the test set. Taking the maximum score over all segments in a day, yields a single score for that day. Thresholding this score results in a discrete prediction: alarm or no alarm. In the experiments, we vary the threshold to observe the precision and false alarm rate under different recall rates. There should be no alarm on the days prior to the day the pattern is absent, and any such alarm is a false positive. A true positive occurs if an alarm is raised within seven days of the absent pattern.[4]

Hyperparameters. Our hyperparameter strategy is designed to give maximal advantage to the baselines over FZapPa. For each of the baselines, we tune the hyperparameters and report the best-achieved results. For FZapPa, the parameters for the bagging Svm are set to the values recommended in the original paper [98] and the warping width for the DTW-distance is set to 0.1, which is a routinely used value that allows for some deviation between the pattern occurrences [138].

3.3.2 Results Q1: Absent Pattern Detection

Figure 3.3 shows the precision (top row plots) and the percentage test set days with a false alarm (bottom) as a function of recall. On each of the six datasets, FZapPa outperforms all the baselines: for any level of recall, it achieves a higher precision. Looking at false alarm rate, averaged over all datasets, at 90% recall, FZapPa issues only 5 ± 2 false alarms per 100 days whereas Lof and MatrixProfile, its two nearest competitors, generate respectively 14 ± 3 and 18 ± 4 false alarms. If one wishes to detect all anomalies, the baselines generate on average 42 ± 21 false alarms per 100 days. FZapPa issues 8 ± 3 false alarms per 100 days. Lof and AverageMissing, its two nearest competitors at this level of recall, issue 26 ± 8 and 23 ± 7 false alarms respectively. Thus, the baselines generate three times as many false alarms as FZapPa and issue a

[4] This number depends on the application being considered and should be chosen accordingly.

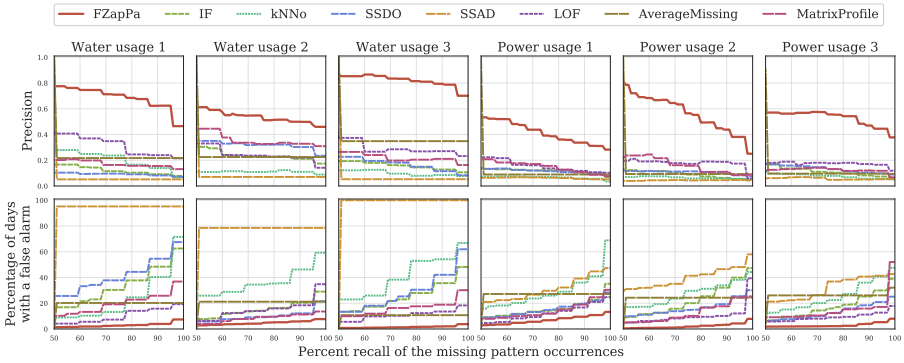


Figure 3.3: Comparison of FZapPa with the baselines at the absent pattern detection task. The plots show the precision (**top**) and the percentage of days in the test set that trigger a false alarm (**bottom**) as a function of recall. On each dataset, FZapPa achieves a substantially higher precision than all of its competitors regardless of the level of recall. Similarly, it has the fewest number of false alarms at each level of recall.

false alarm roughly every three days! Such rates are unacceptable in practice because they create distrust in the detection model.

The main reason the baselines perform poorly, is because they are optimized to detect behavior that is different from the normally observed behavior, not normal behavior that is *absent*. Note that the semi-supervised detectors include the labeled pattern occurrences during training. The following paragraphs discuss the different models in more detail.

Discussion of AverageMissing's Performance. At 90% recall, AverageMissing produces on average over all six datasets 22 ± 5 false alarms per 100 days, which increases to 22 ± 7 when the user requires 100% recall. At recall rates $>90\%$, the precision of AverageMissing, averaged over the water and power datasets, is always < 0.35 and < 0.10 respectively. FZapPa is a substantial improvement over AverageMissing because it both selects the best-fitting descriptor in a data-driven way and models the occurrences with the more appropriate Weibull distribution.

Discussion of Unsupervised Anomaly Detection Algorithms' Performance. The unsupervised techniques perform poorly when detecting absent pattern anomalies as can be concluded from the low precision scores seen in Figure 3.3.

These low precision scores lead to undesirably high false alarm rates. The explanation for the poor performance of these methods is two-fold. First, the absent pattern occurrences are not automatically replaced by *other* anomalous behavior. The pattern is sporadically occurring (infrequent) and its absence is likely to be replaced by frequently occurring normal behavior, which is not recognized as anomalous by said methods. Second, the standard techniques have no internal mechanism to deal with the fact that a pattern occurrence depends on a previous occurrence. The only way to model this type of dependency is if expert knowledge is available to hand-craft relevant features.

Discussion of Semi-supervised Anomaly Detection Algorithms' Performance. Like the unsupervised anomaly detection methods, the semi-supervised anomaly detection methods also show poor performance on the task of detecting absent patterns (see Figure 3.3). These techniques use the known pattern occurrences as examples of normal behavior when training. However, this does not mean that the resulting models can accurately detect an absent occurrence of the pattern because it is most likely replaced by frequently occurring normal behavior. SSAD in particular has trouble learning a good model when the provided labels are only of one class (i.e., the known occurrences).

Discussion of MatrixProfile's Performance. The MATRIXPROFILE method suffers from the aforementioned shortcomings of the traditional anomaly detection techniques as it is similar to applying KNNO with the Euclidean distance function and $k = 1$ directly to the time series segments of T (i.e., without converting them to feature vectors). The problem is compounded by a number of additional constraints on the usefulness of the MATRIXPROFILE technique for anomaly detection. First, if T contains two or more anomalies that are more similar than at least one random pair of normal segments they will not be identified as anomalies. To see why this is the case, consider that the MATRIXPROFILE computes the anomaly score for each time series segment as the Euclidean distance to its nearest neighbor in T. Second, MATRIXPROFILE is computed only with respect to the raw time series signal assuming that all the necessary information to detect anomalies is contained within the signal. Anomalies characterized by a combination of other features (e.g., context features such as time-of-day) will not be identified [145]. Third, the MATRIXPROFILE only computes the Euclidean distance. It does not work with the DTW distance which has been found to be useful for discovering information in time series data [37].

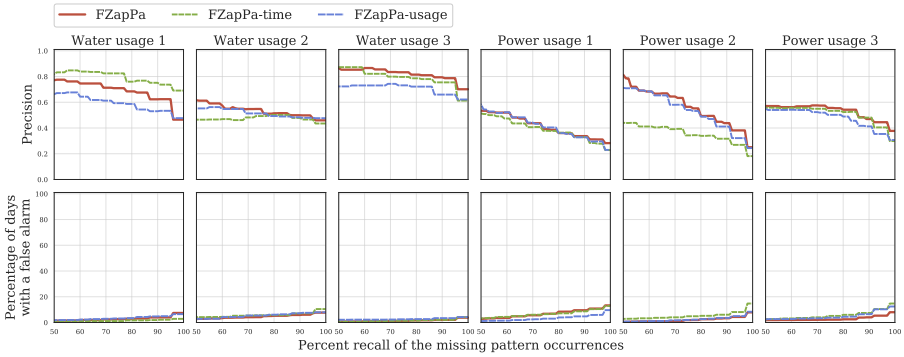


Figure 3.4: Precision (**top**) and the percentage of test set days with a false alarm (**bottom**) as a function of recall for FZapPa, FZapPa-Time, and FZapPa-Usage. In 5 out of 6 datasets, FZapPa raises a fewer or similar number of false alarms compared to FZapPa-Time and FZapPa-Usage. FZapPa maintains higher or equivalent precision compared to its variants for recalls $> 95\%$ in 5 of 6 datasets.

3.3.3 Results Q2: Impact of the Selected Descriptor

A key component of FZapPa is its ability to learn the appropriate context that is predictive of the pattern's occurrence. This entails performing the data-driven selection of the descriptor variable in the Weibull distribution. To illustrate the importance of learning the right context, we compare FZapPa with two variants:

FZapPa-Time is a variant where the descriptor is hand-selected to be the time interval between pattern occurrences.

FZapPa-Usage is a variant where the descriptor is hand-selected to be the cumulative sum of the measured water or power usage since the last detected pattern.

Different descriptors are appropriate for different types of data, necessitating the automatic selection of the descriptor. This is illustrated in Figure 3.4, which shows for each dataset the percentage of test set days with a false alarm (top row plots) and the precision (bottom) as a function of recall. With the exception of the Water usage 1 dataset, FZapPa always achieves a higher or similar precision than FZapPa-Time and FZapPa-Usage. For Power usage 3 and Water usage 3, *time* is an apt descriptor to capture the distribution of the pattern occurrences, while for Power usage 2 and Water usage 2, *usage* is the

better descriptor. In each case, FZapPa figures out the correct descriptor to use.

3.3.4 Results Q3: Choice of Pattern Detector

FZapPa internally uses a PU-classifier to detect the pattern occurrences. We empirically compare FZapPa with two variants that use well-known alternative approaches to the pattern detection subroutine of FZapPa:

FZapPa-Dtw takes the classic shape-based pattern-matching approach to occurrence detection [84]. For a segment, it computes the DTW cost to nearest known example of the pattern. If this cost is below a threshold, then the pattern is detected. The threshold is set to the minimal possible value such that all *known* pattern occurrences are detected.

FZapPa-Fv looks at the feature-vector representation of the segments that contain a known pattern occurrence [7, 84]. For each segment, it computes the Euclidean distance between its feature vector representation and the feature-vector representation of the closest segment containing a known pattern. Again, if this distance is below a threshold, then the pattern is considered to be detected.

Figure 3.5 shows for each dataset the percentage of days in the test set with a false alarm (top) and the precision (bottom) as a function of recall for the three methods. Regardless of dataset or metric, FZapPa achieves superior or equivalent performance compared to its variants. The lone exception is on the Water usage 3 dataset where using the DTW detector results in better precision at recalls of less than 70%. These results demonstrate the benefit of our novel PU learning approach to pattern detection in this setting compared to the more traditional approaches. The advantage stems from the PU learner being able to learn an accurate detection threshold.

3.4 Conclusion

In this chapter, we introduced the problem of detecting absent pattern anomalies in time series data. The user provides flexible supervision in the form of a limited set of annotated occurrences of a pattern of interest. She does not specify what the relevant context is for these occurrences or whether each occurrence is normal or anomalous. We designed FZapPa, an algorithm for detecting absent pattern anomalies. The algorithm works in two steps. First, it detects

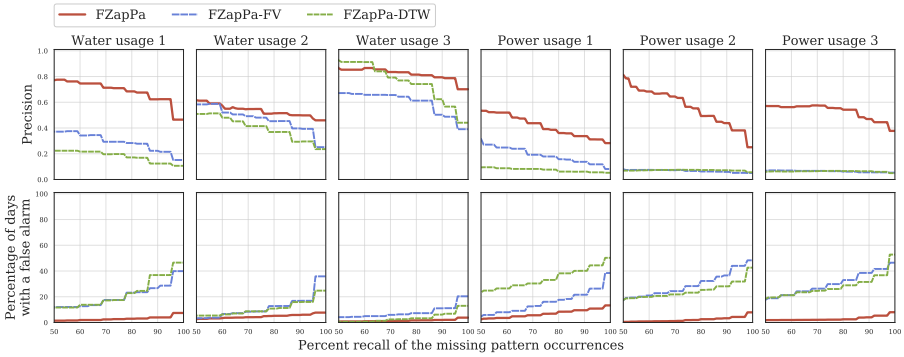


Figure 3.5: Precision (**top**) and the percentage of test set days with a false alarm (**bottom**) as a function of recall for FZapPa, FZapPa-Dtw, and FZapPa-Fv. On each dataset, FZapPa raises a fewer or similar number of false alarms compared to FZapPa-Dtw and FZapPa-Fv. FZapPa maintains higher or equivalent precision compared to its variants for all levels of recall in 5 out of 6 datasets.

all occurrences of the pattern based on the expert supervision. Second, it learns a model of how the pattern is distributed within the time series and uses this model to predict absent occurrences.

Chapter 4

Pattern-Based Anomaly Detection

Anomaly detection in time series is an important real-world problem, especially as an increasing amount of data of human behavior and a myriad of devices is collected, with an increasing impact on our everyday lives. We live in an "Internet of Things" world, where a network of devices, vehicles, home appliances and other items embedded with software and electronics are connected and exchanging data [50]. Although many organisations are collecting time series data, automatically analysing them and extracting meaningful knowledge, such as an understandable model that automatically flags relevant anomalies, remains a difficult problem, even after decades of research.

Exploring different benchmark datasets for time series anomaly detection, we found that these datasets often consist of univariate time series, where anomalies are global point anomalies [21]. In contrast, we focus on *collective* anomalies: a collection of points in the time series is anomalous depending on the surrounding *context*. For instance, most smartphones log many *continuous* times series from various sensors, such as the accelerometer, gyroscope, internal thermometer, and battery level. In addition, smartphones log *discrete* events in different operating system logs, such as applications starting or stopping, certain hardware components being turned on or off, or application-specific events. Such events are crucial in determining whether the behavior observed in the continuous time series, e.g., a spike in power usage, is anomalous. We argue that for many real-world applications one needs to extract information from both types of sources to successfully detect anomalies.

In the active research area for anomaly detection in continuous time series, much attention has been given to finding anomalies using continuous *n-grams*, DTW distance, and similarity to the nearest neighbors [100, 152], but not to integrating event log data. On the other hand, pattern mining based techniques for detecting anomalies have been developed for discrete event logs, but not for continuous time series. In this chapter, we propose how to circumvent the apparent mismatch between discrete patterns and continuous time series data. We introduce a pattern-based anomaly detection algorithm that can flexibly detect anomalies in mixed-type time series, i.e., time series consisting of both continuous sensor values and discrete event logs.

Given a time series dataset, the method leverages the mature field of *frequent pattern mining* research [153] to find frequent patterns in the data, serving as a template for the frequently occurring normal behavior. Then, the frequent patterns are used to map the time series data to a feature-vector representation. This newly found pattern-based embedding of the data combines the information in both the continuous time series and the discrete event logs into a single feature vector. Finally, a state-of-the-art anomaly detection algorithm is used to find the anomalies in the embedded space.

Contributions of this Chapter

The work outlined in this chapter, makes the following three contributions:

The **first** contribution is PBAD (Pattern-Based Anomaly Detection): an algorithm for transforming a combination of continuous time series and event logs into a format suitable for applying anomaly detection techniques.

The **second** contribution is an extensive experimental evaluation of PBAD on a benchmark of univariate time series, multivariate time series, and mixed-type time series (i.e., the combination of continuous time series and discrete event logs).

The **final** contribution is an implementation of PBAD available at: `https://bitbucket.org/len_feremans/pbad`

Scientific Publications

The content of this chapter is based on the following publication [45]:

L. Feremans*, V. Vercruyssen*, B. Cule, W. Meert, and B. Goethals (2019). Pattern-Based Anomaly Detection in Mixed-type Time Series. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, Würzburg (Germany), pp. 240-256.

Problem Statement

The problem addressed in this chapter is formalized as follows:

Given: A univariate, multivariate, or mixed-type time series $\mathcal{T}$ consisting of at least one continuous time series T ($N \geq 1$) and one event log E ($M \geq 1$).

Do: Identify anomalous periods of $\mathcal{T}$.

4.1 Pattern Mining Preliminaries

We provide the following definitions for frequent pattern mining [153], adapted to the context of mixed-type time series. The first type of pattern we consider is an *itemset*. For an itemset, no temporal order between items is required. An itemset I consists of one or more items $\mathrm{I}_j \in \Xi$, where Ξ is a finite domain of discrete values, that is, $I = \{\mathrm{I}_1, \ldots, \mathrm{I}_m\} \subseteq 2^{|\Xi|}$. An itemset I *occurs* in, or is covered by, a segment $S^i_{t,l}$ if all items in I occur in that segment in any order, that is,

$$I \prec S^i_{t,l} \Leftrightarrow \forall \mathrm{I}_j \in I : \exists \langle \mathrm{I}_j, t_j \rangle \in S^i_{t,l}.$$

Given the set of all segments $\mathcal{S}$ of a time series, we define *cover* and *support* as

$$\mathit{cover}(I, \mathcal{S}) = \{S^i_{t,l} | S^i_{t,l} \in \mathcal{S} \wedge I \prec S^i_{t,l}\} \text{ and } \mathit{support}(I, \mathcal{S}) = |\mathit{cover}(I, \mathcal{S})|.$$

The second type of pattern we consider is a *sequential pattern*. A sequential pattern I_s consists of an ordered list of one or more items, denoted as $I_s = (\mathrm{I}_1, \ldots, \mathrm{I}_m)$, where $\mathrm{I}_j \in \Xi$. A sequential pattern can contain repeating items, and, unlike n-grams, an occurrence of a sequential pattern allows gaps between items. We define that a sequential pattern I_s *occurs* in a segment $S^i_{t,l}$ using

$$I_s \prec S^i_{t,l} \Leftrightarrow \exists \langle \mathrm{I}_1, t_1 \rangle, \ldots, \langle \mathrm{I}_p, t_m \rangle \in S^i_{t,l} : \forall i, j \in \{1, \ldots, p\} : i < j \Rightarrow t_i < t_j.$$

The definitions of cover and support are equivalent to those of itemsets. Finally, an itemset or a sequential pattern is *frequent* if its support is higher than a

user-defined threshold on the minimal support (parameter *min_sup* in our algorithm).

Given a set of segments $\mathcal{S}$ and discretised continuous values, we can use existing itemset or sequential pattern mining algorithms to efficiently mine all *frequent* patterns in both continuous and discrete time series [153, 48]. However, even with the restriction that patterns must occur at least *min_sup* times, it remains a challenge to filter out *redundant* patterns. We will focus on algorithms that mine only *closed* or *maximal* patterns. An itemset I is not closed, and thus redundant, if and only if there exists an itemset Z, such that $I \subset Z$ and $support(I, \mathcal{S}) = support(I, \mathcal{S})$. Likewise, a sequential pattern I_s is not closed if there exists a sequential pattern Z_s, such that $I_s \sqsubset Z_s$ and $support(I_s, \mathcal{S}) = support(Z_s, \mathcal{S})$, where $\sqsubset$ is used to denote the subsequence relation.

4.2 Methodology

Outline of Pbad. PBAD computes for each time series segment of $\mathcal{T}$ an anomaly score. The method has four major steps. First, the time series is preprocessed. Second, frequent itemsets and sequential patterns are mined from the individual continuous time series or event logs $\mathcal{T}^i \in \mathcal{T}$. Third, the distance-weighted similarity scores between the frequent patterns and each time series segment are computed to construct a pattern-based embedding of time series $\mathcal{T}$. Fourth, the embedding is used to construct an anomaly detector to detect the anomalous periods of $\mathcal{T}$. We use the IFOREST algorithm in this Chapter because it is a state-of-the-art detector. The four steps are illustrated in Figure 4.1. In the following paragraphs, we outline each step in more detail.

4.2.1 Preprocessing

Preprocessing is the first step in PBAD shown in Algorithm 1, line 1-7. First, we *normalise* each continuous time series $\mathcal{T}^i$ in $\mathcal{T}$ to values between 0 and 1, because multivariate time series can have widely different amplitudes.

Then, we segment each time series in $\mathcal{T}$ using a *fixed-size sliding window* of length l. Frequent patterns are often limited in length, i.e., a length of 5 is already quite high. Therefore, it can be useful for certain datasets to also reduce the length of each extracted segment such that a frequent sequential pattern or itemset covers a large proportion of each segment. For example, we can reduce a continuous segment of length 100, using a *moving average*, by storing the average value for every 5 consecutive values. The resulting segment now has a

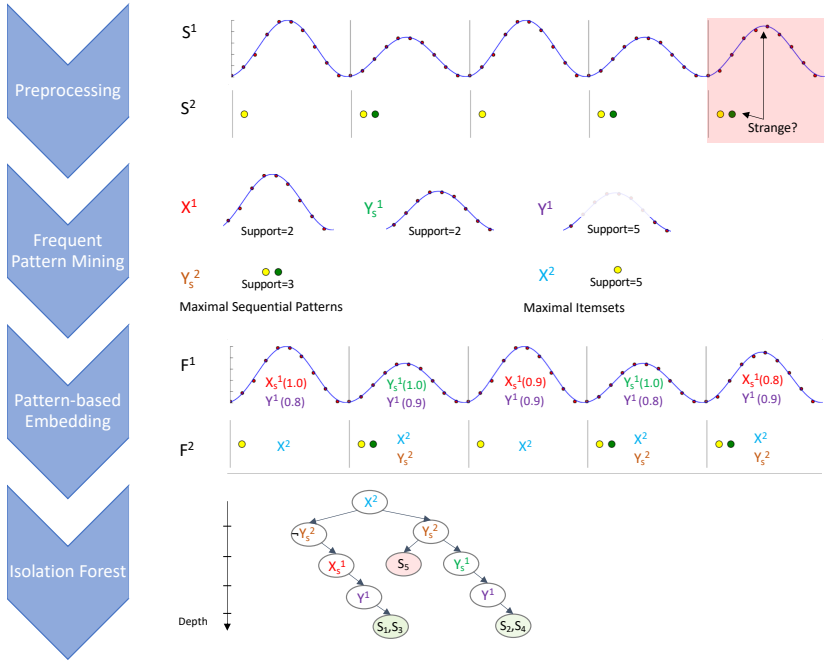


Figure 4.1: Illustration of major steps in pattern-based anomaly detection method in mixed-type time series. Note that the 1^{st} and 3^{th} segment, and the 2^{nd} and 4^{th} segments match the same set of patterns extracted from both the time series and the event log. The 5^{th} segment is anomalous because of the co-occurrence of a peak with a green event. The isolation forest step in PBAD marks it as such since its depth is only 2.

length of 20 and subsequently any matching sequential pattern or itemset of size 5 will cover a large part of the segment.

Frequent pattern mining only works on *discretised* data. Event logs are by nature discretised. For a continuous time series, each segment $S^i_{t,l} = (v_1, \ldots, v_l)$ must be discretised using a function f to yield a discrete representation $f(S^i_{t,l}) = (v_1{}', \ldots, v_l{}')$ where $v_j{}' \in \Xi$ and Ξ represents a chosen "alphabet" of discrete values of a restricted size. Examples of such functions include equal-width or equal-frequency binning, or aggregating segments and computing a symbol using *symbolic aggregate approximation* [83].

Algorithm 1: PBAD($\mathcal{T}$, l, *min_sup*, *is_maximal*, *max_sim*)

Input : Data $\mathcal{T}$, window length l, support threshold *min_sup*, *is_maximal* (`maximal` or `closed`), threshold on max Jaccard similarity *max_sim*

Result: Anomaly scores for each time series segment $S_{t,l}$

```
// 1. Preprocessing: create windows and discretize
```
$\mathcal{S} \leftarrow \varnothing$

foreach $\mathcal{T}^i \in \mathcal{T}$ **do**

  if $\mathcal{T}^i$ *is continuous* **then**

    $\mathcal{S}^i \leftarrow$ CREATE_SEGMENTS(DISCRETIZE(NORMALISE($\mathcal{T}^i$)), l)

  else

    $\mathcal{S}^i \leftarrow$ CREATE_SEGMENTS($\mathcal{T}^i, l$)

  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}^i$

```
// 2. Mine frequent itemsets and sequential patterns
```
$\mathcal{P} \leftarrow \varnothing$

foreach $\mathcal{S}^i \in \mathcal{S}$ **do**

  $\mathcal{P}^i \leftarrow$ MINE_FREQ_ITEMSETS($\mathcal{S}^i$, *min_sup*, *is_maximal*)

  $\mathcal{P}^i \leftarrow \mathcal{P}^i \cup$ MINE_FREQ_SEQ_PATTERNS($\mathcal{S}^i$, *min_sup*, *is_maximal*)

  `// Remove redundant patterns using Jaccard`

  $\mathcal{P}^i \leftarrow$ SORT $\mathcal{P}^i$ on descending support

  for $1 \leq i < |\mathcal{P}^i|$ **do**

    for $i + 1 \leq j \leq |\mathcal{P}^i|$ **do**

      if $J(I_i, I_j) \geq$ *max_sim* **then**

        $\mathcal{P}^i \leftarrow \mathcal{P}^i \setminus I_j$

  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}^i$

```
// 3. Compute pattern based embedding
```
$X \leftarrow$ matrix of 0 vals with $|\mathcal{P}|$ columns and $|\mathcal{S}|$ rows for each segment $S_{t,l}$

for $1 \leq i \leq |\mathcal{T}|$ **do**

  for $1 \leq j \leq |\mathcal{P}^i|$ **do**

    $idx \leftarrow$ global index of I^i_j in $\mathcal{P}$

    for $1 \leq t \leq |\mathcal{S}|$ **do**

      `// Weighted similarity between pattern` I^i_j `and segment` $S^i_{t,l}$ `for time series` i

      $X_{t,idx} \leftarrow 1.0 - \frac{\text{EXACTMINDIST}(I^i_j, S^i_{t,l})}{|I^i_j|}$

```
// 4. Compute anomaly scores using Isolation Forest
```
scores $\leftarrow$ SORT IFOREST(X) descending

4.2.2 Extracting Frequent Patterns

After preprocessing, PBAD mines frequent itemsets and sequential patterns from time series $\mathcal{T}$. Given the assumption that anomalous behavior occurs infrequently in the time series, the frequent patterns would characterise the frequently observed, normal behavior. To extract the frequent patterns, we leverage the mature field of frequent pattern mining. This has two main advantages. First, the existing techniques can be extended to mine patterns in different types of time series data (continuous and event logs). Second, the mined patterns are easily interpretable and can later be presented to the user to give intuitions as to why the classifier labeled a segment as normal or anomalous.

Extracting Frequent Itemsets and Sequential Patterns. After preprocessing, we have created a set of segments for each series. These can trivially be transformed to a transaction (or sequence) database, required by existing frequent pattern mining algorithms [153]. Said algorithms generate candidate patterns with growing length. Since support decreases with the length of either the itemset or sequential pattern, it is relatively straightforward for these algorithms to only enumerate patterns that are frequent by pruning the candidate patterns on *min_sup*. We always mine both itemsets and sequential patterns, but filter either on *closed* or *maximal* patterns depending on the status of the parameter *is_maximal* (line 9-11). The implementation of both closed and maximal itemsets and sequential pattern mining algorithms is available in the SPMF library [48].

Removing Overlapping Patterns. Frequent pattern mining algorithms can generate too many redundant patterns. To further reduce the set of patterns, we employ *Jaccard similarity* to remove itemsets and sequential patterns that co-occur in a large percentage of segments. Formally, we use a parameter *max_sim*, and remove all patterns with a high enough Jaccard similarity:

$$J(I_1, I_2) = \frac{|cover(I_1) \cap cover(I_2)|}{support(I_1) + support(I_2) - |cover(I_1) \cap cover(I_2)|} \tag{4.1}$$

If $J(I_1, I_2) \geq max\_sim$, we remove the pattern with the lowest support. We created a straightforward routine that compares all pattern pairs (line 12-17).

Dealing with Multivariate and Mixed-Type Time Series. For multivariate and mixed-type time series, we consider two strategies for pattern extraction: *early* and *late integration*. Under the *early integration* strategy, the items of all preprocessed series in $\mathcal{T}$ are combined into a single event sequence.

The frequent patterns are then mined over this new event sequence, resulting in patterns containing values from multiple dimensions. For example, the segments $S^1_{1,4} = (1, 2, 3, 4)$ and $S^2_{1,4} = (10, 10, 11, 11)$ spanning the same period in two time series $\mathcal{T}^1$ and $\mathcal{T}^2$, can be combined into a single event sequence $E_{1,4} = (\{1^1, 10^2\}, \{2^1, 10^2\}, \{3^1, 11^2\}, \{4^1, 11^2\})$. Frequent patterns can now be mined in this single event sequence, yielding candidate patterns such as the sequential pattern $I_s = (1^1, 2^1, 11^2, 11^2)$, meaning that value 1 followed by 2 in series $\mathcal{T}^1$ is followed by 2 occurrences of 11 in series $\mathcal{T}^2$.

In contrast, the *late integration* strategy mines patterns in each time series of $\mathcal{T}$ separately and takes the union of the resulting set of patterns. Now, each pattern is associated with exactly one time series. While it would be tempting to conclude that early integration is better since it can uncover patterns containing events from different dimensions as well as any order between these events, we prefer *late integration* in our experiments for two reasons. First, in practice, early integration leads to an exponential increase in the search space of possible patterns, i.e., *pattern explosion*, since the pattern mining algorithms consider every possible combination of values in each of the time series. Second, the anomaly detection classifier in PBAD is constructed on the union of pattern-based embeddings of each time series in $\mathcal{T}$. As such, it learns the structure between patterns from the separate time series.

4.2.3 Constructing the Pattern-Based Embedding

Having obtained a set of patterns for the time series in $\mathcal{T}$, PBAD maps $\mathcal{T}$ to a pattern-based embedding in two steps. First, it computes a similarity score between each segment $S^i_{t,l}$ and each pattern I^i mined from the corresponding time series $\mathcal{T}^i$ in $\mathcal{T}$. If $\mathcal{T}^i$ is continuous, PBAD computes a *distance-weighted similarity score*. If $\mathcal{T}^i$ is an event log, PBAD computes the exact match, i.e. 1 if $I^i \prec S^i_{t,l}$ and 0 otherwise. Second, it concatenates the similarity scores over all dimensions, yielding the feature-vector representation of the segment of $\mathcal{T}$. Since this process is repeated for each segment in $\mathcal{T}$, we end up with a pattern-based embedding of the full time series (line 18-23). We argue that normal time series segments are more frequent than the anomalous segments and as such normal segments match the set of frequent patterns better. As a result, they will be clustered together in the embedded space whereas the less frequent anomalous segments will have lower similarity scores and will be more scattered in the embedded space.

Computing the Distance-Weighted Similarity Score. The intuition behind a weighted similarity score can be illustrated with a simple example. For

instance, the sequential pattern $I^1 = (0.1, 0.5)$ clearly matches time series segment $S^1_{1,3} = (0.1, 0.55, 1.0)$ better than segment $S^1_{4,3} = (0.8, 0.9, 1.0)$. Thus, the similarity between a sequential pattern I^i of length m and a time series segment $S^i_{t,l}$ of length l depends on the *minimal* Euclidean distance between the pattern and the segment:

$$weighted\_dist(I^i, S^i_{t,l}) = \min_{E \subset S^i_{t,l}} \sqrt{\sum_{j=1}^{m} (E_j - I^i_j)^2} \tag{4.2}$$

where E is a subsequence of m elements from segment $S^i_{t,l}$. The optimisation yields the minimal distance by only observing the best matching elements in the pattern and the segment. Given the weighted distance between the sequential pattern and a segment, a similarity score is computed as follows:

$$sim(I^i, S^i_{t,l}) = 1.0 - weighted\_dist(I^i, S^i_{t,l})/|I^i| \tag{4.3}$$

If the distance between the pattern and the time series segment is small, the similarity increases. Since the patterns can have different lengths, the distance is normalised for the length of the pattern. Going back to the simple example, the similarity with segment $S^1_{1,3}$ is $1.0 - \sqrt{(0.1 - 0.1)^2 + (0.55 - 0.5)^2}/2 = 0.975$ while the similarity with segment $S^1_{4,3}$ is only $1.0 - \sqrt{(0.8 - 0.1)^2 + (0.9 - 0.5)^2}/2 = 0.597$.

Because a sequential pattern imposes a total order on its items, Equation 4.2 cannot be solved trivially. We design an exact algorithm for computing Equation 4.2 with the added ordering constraint. This algorithm matches every element in the pattern with exactly one unique element in the segment such that the sum of the distances between the matched elements is minimal. The approach is based on the *Smith-Waterman* algorithm for local sequence alignment. However, in contrast, our algorithm ensures that every element in the segment and pattern can only be matched once and enforces a match for every element in the pattern. Furthermore, it imposes no gap penalty for skipping elements. Finally, it returns an exact distance between the pattern and the segment. Since it is a *dynamic programming algorithm*, it is guaranteed to find the optimal alignment of the pattern and the segment that minimises the distance. The full ExactMinDist algorithm is detailed in the following paragraph.

The ExactMinDist Algorithm. Algorithm 2 outlines the exact distance computation between a pattern I with length m and time series segment S with length l. Here, we use the shorter notation S to denote a segment between two timestamps for the i^{th} time series, previously denoted as $S^i_{t,l}$. First,

Algorithm 2: ExactMinDist (I, S)

Input : Pattern I (`itemset` or `sequential pattern`), segment S
Result: Computed distance d

1 **if** I *is an* `itemset` **then**
2 $\quad I \leftarrow \text{SORT}(I)$; $S \leftarrow \text{SORT}(S)$
3 $m \leftarrow \text{LENGTH}(I)$; $l \leftarrow \text{LENGTH}(S)$
4 $w = l - m + 1$
`// 1. Construct the scoring matrix` H `using 0-based indexing:`
5 $H \leftarrow$ matrix of ∞ values with $l + 1$ rows and $m + 1$ columns
`// 2. Fill the first column of the matrix with 0:`
6 $H_{j,0} = 0$ **for** $0 \leq j < l + 1$
`// 3. Update the values in the matrix:`
7 **for** $0 \leq i < m$ **do**
8 $\quad$ **for** $i \leq j < i + w$ **do**
9 $\quad\quad H_{j+1,i+1} = \min \begin{cases} H_{j,i} + (S_j - I_i)^2 \\ H_{j,i+1} \end{cases}$
`// 4. The distance is in the last row and column of the matrix:`
10 $d = \sqrt{H_{j+1,i+1}}$

if the pattern is an itemset, the pattern and segment are sorted (line 1-2). Next, a scoring matrix with $l + 1$ rows and $m + 1$ columns is initialized with the first column zeros and the rest of the values ∞ (line 5-6). Intuitively, each row and each column, except the first row and column, correspond to one element in the segment and the pattern respectively. Then, each cell in the matrix is updated by setting its value to the minimum of: (1) the sum of the distance between the elements of the pattern and segment corresponding to that cell and the value in the cell diagonally up to the left, and (2) the value in the cell above (line 7-9). Finally, the distance that minimizes Equation 4.2 is stored in the last cell of the matrix.

The ExactMinDist algorithm has a time complexity of $\mathcal{O}(l \cdot m)$. However, we can further reduce the number of computations by only filling in the feasible regions of the matrix. This is possible because there is an ordering to the matched elements. For example, if the pattern has length 5 and the segment has length 7, the first element of the pattern can only be matched with one of the first three elements of the segment. This is expressed by w in Algorithm 2.

Dealing with Itemsets. PBAD also computes the similarity between itemsets and segments. In contrast to a sequential pattern, an itemset does not impose an order on its elements. However, we can still use the EXACTMINDIST algorithm to obtain the correct weighted distance and compute the similarity score if we first sort the elements of both the itemset and segment. This ensures that the EXACTMINDIST algorithm matches the correct elements.

Constructing the Embedding Under the Early Integration Strategy. In case of the early integration strategy, we must deal with patterns with mixed items from different continuous time series and event logs when computing the similarity score. For itemsets, we adapt Equation 4.2 and compute the minimal distance in each dimension separately and then sum all distances over all dimensions. The distance is computed either weighted, i.e., between an item and a continuous time series value, or binary, i.e., between an item and a discrete event log value. For sequential patterns, we consider the subsequence in each dimension separately and sum all distances. However, in this case we have to satisfy the total order constraint within each time series and between different time series. A brute-force way to compute this, is to generate all possible subsequences (with gaps) over each dimension that satisfy the local and global order constraints, induced by each sequential pattern, and take the subsequence that has the smallest distance. In practice, this is feasible since the length of the time series and patterns is limited to small numbers.

Time Complexity. The time complexity of constructing the pattern-based embedding of $\mathcal{T}$ is $\mathcal{O}(|\mathcal{P}| \cdot |\mathcal{S}| \cdot o)$ where $o = \mathcal{O}(l \cdot m)$ is required by the EXACTMINDIST algorithm, $|\mathcal{P}|$ the number of frequent patterns found, and $|\mathcal{S}|$ the number of segments in the time series. Under the late integration strategy, this complexity increases linearly with the number of dimensions of $\mathcal{T}$.

4.2.4 Constructing the Anomaly Detection Classifier

The final step of PBAD is to construct the anomaly detection classifier (line 24 in Algorithm 1). Given the pattern-based embedding of $\mathcal{T}$, any state-of-the-art anomaly detector can be used to compute an anomaly score for each segment of $\mathcal{T}$. PBAD uses the IFOREST classifier [136] since it has been empirically shown to be the state-of-the-art in unsupervised anomaly detection [38]. Anomalies are isolated quickly from the data, resulting in shorter path lengths in the tree, as illustrated in Figure 4.1.

4.3 Related Work

This section briefly describes the most related work within the context of this specific chapter. Existing pattern-based anomaly detection methods each differ in how they define patterns, support, anomaly scores, and what type of input they are applicable to. The original *frequent-pattern-based outlier factor* FPOF method [63] mines frequent itemsets for detecting anomalous transactions in a transaction database, using the traditional definition of support. Their outlier factor is defined as the number of itemsets that match the current transaction versus the total number of frequent itemsets. The more recent *minimal-infrequent-pattern-based outlier factor* MIFPOD method [64] mines minimal infrequent, or rare, itemsets for detecting outliers in data streams. Rare itemsets are not frequent, i.e., they do not satisfy the minimal support threshold, but have only subsets that are frequent. They define support as usual, but based on the most recent period, and not necessarily the entire event log. Finally, they compute the outlier factor as the number of rare itemsets that match the current transaction versus the total number of rare itemsets, similar to FPOF, but weighted by the sum of deviation in support of matching rare itemsets. Finally, the *pattern-based anomaly value* PAV method [24] uses linear patterns, i.e., two consecutive continuous values in a univariate time series. The final outlier factor is computed as the relative support of this single pattern in sliding windows of size two. In contrast to these methods, PBAD considers extensions specific to multivariate and mixed-type time series: it uses a distance-weighted similarity to bridge the gap between a discrete pattern and a continuous signal, employs a late integration scheme to avoid pattern explosion, removes redundant patterns using a threshold on Jaccard similarity, and derives an anomaly score using the IFOREST classifier and the pattern-based embedding of the time series.

The MATRIXPROFILE technique [152] is the state-of-the-art anomaly detection technique for continuous time series. This technique computes for each time series segment an anomaly score by computing the Euclidean distance to its nearest neighbour segment. In contrast, PBAD can also handle the combination of event logs and continuous time series. A host of *time series representation methods* and *similarity measures* have been developed [37] for time series classification. Time series *shapelets* are subsequences from a continuous time series and are used in combination with the dynamic time warping distance to classify time series segments [151]. Sequential patterns used by PBAD are different from shapelets, because we use non-continuous subsequences with missing elements, or gaps. Itemsets are even more different, because the order of values is discarded. Another key difference is that the enumeration process for shapelets is usually reduced by only considering subsequences of a specific

length [69], while we make use of the anti-monotonic property of support to enumerate patterns of varying length without constraints. Finally, itemsets and sequential patterns can also be extracted from discrete event logs.

Another approach, related to classification, is to employ a *minimal redundancy, maximal relevance* strategy to select *discriminative* patterns [25]. Currently, we employ an unsupervised technique, but for future work we could adopt a strategy for selecting patterns that are the most discriminative towards anomalies. Finally, deep learning techniques are becoming a popular choice for time series anomaly detection [92]. For instance, autoencoders could be used to learn an embedding of a mixed-type time series. A key difference with PBAD is that, unlike deep learning techniques, frequent patterns are easily interpretable.

4.4 Experiments

The empirical evaluation of PBAD focuses on three questions:

Q1: how does PBAD perform compared to the state-of-the-art pattern based anomaly detection algorithms on univariate time series data?

Q2: how does PBAD compare on multivariate time series data?

Q3: how does PBAD compare on mixed-type time series data?

We evaluate PBAD on three types of time series: real-world univariate time series, real-world multivariate time series, and synthetic mixed-type time series. Before discussing the results, we lay out the experimental setup.

4.4.1 Experimental Setup

Compared Approaches. We experimentally compare PBAD with five state-of-the-art approaches, divided into two categories:

Frequent pattern-based baselines. PAV is a multi-scale anomaly detection algorithm based on infrequent patterns, specifically bi-grams, for univariate time series [24]. MIFPOD is an anomaly detection method for event logs [64]. Their outlier factor is based on minimal infrequent, or *rare*, itemsets. FPOF computes an outlier factor based on the number of frequent itemsets that exactly match the current transaction for transactional databases [63]. We adapt FPOF and compute the outlier factor based on closed itemsets and reduce itemsets further using Jaccard similarity as in PBAD.

Time series dataset	$\|\mathcal{T}\|$	Labeled time period	Labeled normals	Labeled anomalies	Δt	Total length of the series
Ambient Temperature	7,267	395 *h*	347 *h*	48 *h*	1 *h*	302 *days*
Request Latency	4,017	66 *h*	14 *h*	52 *h*	5 *min*	13 *days*
New York Taxi	10,320	356 *h*	236 *h*	120 *h*	30 *min*	215 *days*
Water use store 1	292,632	1,217 *h*	829 *h*	388 *h*	5 *min*	1,016 *days*
Water use store 2	281,485	2,391 *h*	2,178 *h*	213 *h*	5 *min*	977 *days*
Water use store 3	169,253	1,595 *h*	1,488 *h*	107 *h*	5 *min*	587 *days*
Water use store 4	292,608	1,821 *h*	615 *h*	1,206 *h*	5 *min*	1,016 *days*
Water use store 5	364,032	574 *h*	504 *h*	70 *h*	5 *min*	1,264 *days*
Water use store 6	364,032	1,047 *h*	815 *h*	232 *h*	5 *min*	1,264 *days*
Lu+Si/Sq	11,152	371.7 *sec*	315.6 *sec*	56.1 *sec*	0.034 *sec*	371.7 *sec*
Lu/Sq	17,318	577.2 *sec*	539.7 *sec*	37.5 *sec*	0.034 *sec*	577.2 *sec*
Si/Sq	11,114	370.5 *sec*	325.6 *sec*	44.9 *sec*	0.034 *sec*	370.5 *sec*
Sq/Si	11,613	387.1 *sec*	349.8 *sec*	37.3 *sec*	0.034 *sec*	387.1 *sec*
Synthetic	105,120	8,760 *h*	8,629 *h*	131 *h*	5 *min*	365 *days*

Table 4.1: Characteristics of all the datasets used in the experiments. The labels and total length of the series are indicated in time units: days, hours, minutes, and seconds.

Time series anomaly detection methods. The MatrixProfile[1] is an anomaly detection technique based on all-pairs-similarity-search for time series data [152]. The anomalies are the time series discords.

Data. The benchmarks used in this chapter consist of 13 different real-world univariate and multivariate datasets. The characteristics of each are outlined in Table 4.1.

Experimental Setup. The experimental setup corresponds to the standard setup in time series anomaly detection [64]. Given a time series $\mathcal{T}$ with a number of labeled timestamps: (1) use a fixed-size sliding window to divide the time series into segments; (2) each segment that contains a labeled timestamp takes its label; (3) construct the anomaly detection model on the full time series data; (4) use the model to predict an anomaly score for each segment in the time series; (5) evaluate the predictions on the labeled segments by computing the AUROC and AP (estimate of the AUPRC).

Hyperparameters. Each method has the same preprocessing steps which includes setting an appropriate window size and increment to create the fixed-size, sliding window. Continuous variables are discretised using equal-width

[1]We denote it MP in the result tables.

Dataset	Window length	Window increment	Nr. bins
`Ambient temperature`	12 *h*	6 *h*	*99*
`Request latency`	1 *h*	30 *min*	*31*
`New York taxi`	6 *h*	3 *h*	*79*
`Water use store 1-6`	1 *h*	1 *h*	*10-50*
`Multivariate indoor exercise`	3 *sec*	0.5 sec	30
`Mixed-type synthetic data`	1 *h*	1 *h*	30

Table 4.2: The window length and window increment of the fixed-size sliding window used to divide each time series into segments, chosen in function of the sample rate Δt of the underlying dataset. Due to their nature, the water data measurements are already logged as discrete values, similarly for `Temp`, `Latency`, and `Taxi`.

binning. See Table 4.2 for details on setting preprocessing parameters. PAV has no parameters. MATRIXPROFILE has a single parameter, the window size. The parameters of FPOF and MIFPOD are chosen by an oracle that knows the optimal settings for each dataset. For PBAD, as a rule of thumb, we set minimal support relatively high, that is $min\_sup = 0.01$. The Jaccard threshold is set to 0.9. Intuitively, if two patterns cover almost the same segments, e.g., 90 out of 100 segments, using both patterns is both unnecessary and less efficient. For mining *closed itemsets* we use CHARM, and for mining *maximal itemsets* we use CHARM-MFI. For mining *closed* and *maximal sequential patterns* we use CM-CLASP and MAXSP respectively. CHARM and CM-CLASP are based on a vertical representation. MAXSP is inspired by PREFIXSPAN which only generates candidates that have at least one occurrence in the database [153, 48]. The sequential patterns should have a minimal length of 2, and by default we set pattern pruning to *closed*. We use the IFOREST classifier implemented in SCIKIT-LEARN with 500 trees in the ensemble. The implementation, datasets and experimental scripts for PBAD are publicly available.[2] We do not report detailed run-time results, however, on the selected datasets PBAD requires less than 30 minutes on a standard PC.

Before mining the patterns and computing an anomaly score, PBAD and the baselines preprocess the time series and divide them into segments using a fixed-size sliding window. Table 4.2 shows the values for the window length and increment for each dataset that were used throughout the experiments. If the increment is equal to the window length, the resulting segments do not overlap. The table also shows the number of bins used for discretizing each dataset if this is required by the pattern mining algorithms.

[2]Implementation of PBAD: `https://bitbucket.org/len_feremans/pbad/`.

	AUROC					AP				
dataset	MP	PAV	MIFPOD	FPOF	PBAD	MP	PAV	MIFPOD	FPOF	PBAD
Temp	0.240	0.590	0.997	**0.999**	0.998	0.014	0.040	0.917	**0.957**	0.917
Taxi	0.861	0.281	0.846	0.877	**0.879**	0.214	0.057	0.300	0.403	**0.453**
Latency	0.599	**0.608**	0.467	0.493	0.553	0.515	0.361	0.255	0.296	**0.382**
Water 1	0.656	0.482	0.514	0.825	**0.884**	0.499	0.301	0.328	0.812	**0.821**
Water 2	0.600	0.520	0.513	0.857	**0.945**	0.353	0.127	0.094	0.688	**0.862**
Water 3	0.536	0.457	0.544	**0.671**	0.605	0.126	0.121	0.079	**0.350**	0.233
Water 4	0.675	0.579	0.548	0.613	**0.721**	0.774	0.687	0.700	**0.817**	0.808
Water 5	0.444	0.581	0.455	0.790	**0.960**	0.199	0.243	0.111	0.671	**0.906**
Water 6	0.682	0.609	0.500	**0.874**	0.752	0.578	0.431	0.228	**0.692**	0.551
Average	0.588	0.523	0.598	0.778	**0.811**	0.364	0.263	0.335	0.632	**0.659**
Ranking	3.333	3.889	4.167	2	**1.611**	3.111	4.111	4.278	1.778	**1.722**

Table 4.3: The table shows the AUROC and AP obtained by each method on 9 univariate time series. PBAD outperforms the baselines in 5 of the 9 datasets.

4.4.2 Results Q1: Anomaly Detection in Univariate Time Series

For the univariate test case, we use nine real-world datasets. Three datasets are from the Numenta time series anomaly detection benchmark [2]. `Temp` tracks the ambient temperature in an office during 302 days where the goal is to detect periods of abnormal temperature. `Latency` monitors CPU usage for 13 days in a data center with the goal of detecting abnormal CPU usage. Finally, `Taxi` logs the number of NYC taxi passengers for 215 days in order to detect periods of abnormal traffic. The 6 remaining datasets are not publicly available. Each tracks on average 2.5 years of `water` consumption in a different store of a large retail company. The company wants to detect periods of abnormal water consumption possibly caused by leaks or rare operational conditions. Domain experts have labeled between 547 and 2 391 hours in each store.

Results and Discussion. Table 4.3 shows the AUROC and AP obtained by each method on each of the nine univariate time series datasets as well as the ranking of each method. PBAD outperforms the existing baselines for detecting anomalies in univariate time series data in five of the nine datasets.

In the experiments, MATRIXPROFILE sometimes performs close to random. Because MATRIXPROFILE detects anomalous segments as those with the highest distance to its nearest neighbour segment in the time series, its performance degrades if the data contain two or more similar anomalies. This is, for instance, the case for the `water` consumption data where each type of leak corresponds to a specific time series pattern.

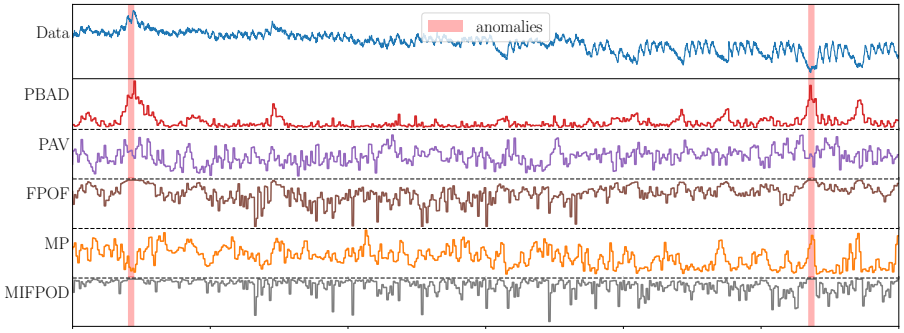


Figure 4.2: The figure plots 125 days of the `ambient temperature` time series including 48 hours labeled as anomalous. For each method, the scaled anomaly scores are plotted, a higher score indicating a more anomalous time series segment. Both PBAD and FPOF accurately identify the labeled anomalous segments.

We compared the impact of computing the distance-weighted similarity between a pattern and time series segment, versus computing an exact match, on the univariate datasets. In this case, using distance-weighted similarity, results in a higher AUROC and AP on eight of the nine datasets. Using the combination of both frequent itemsets and frequent sequential patterns instead of only itemsets or only sequential patterns results in higher AUROC on six of the nine datasets.

The application of PBAD and the baselines to detect abnormal temperatures in the `Temp` dataset is illustrated in Figure 4.2. The anomaly scores outputted by PBAD and FPOF peak on the days an actual anomaly was recorded, indicating that these methods successfully detect the abnormal periods. MIFPOD also accurately finds the anomalies. PAV has highly fluctuating anomaly scores with the scores for the actual anomalies somewhere in the middle of the pack. The AUROC of PAV on this is 0.584 (see Table 4.3), indicating that its predictions are almost random.

4.4.3 Results Q2: Anomaly Detection in Multivariate Time Series

For the multivariate test case, we use an indoor exercise monitoring dataset [34]. The data contain recordings of ten people each executing 60 repetitions of three types of exercises: squats (`Sq`), lunges (`Lu`), and side-lunges (`Si`). The (x, y, z) positions of 25 sensors attached to each person were tracked during execution,

	AUROC					AP				
Dataset	**MP**	PAV	MIFPOD	FPOF	PBAD	**MP**	PAV	MIFPOD	FPOF	PBAD
Lu+Si/Sq	0.472	0.571	0.819	0.966	**0.983**	0.283	0.255	0.430	0.862	**0.888**
Lu/Sq	0.604	0.671	0.775	**0.966**	0.940	0.082	0.110	0.131	0.662	**0.737**
Si/Lu	0.471	0.425	0.804	0.864	**0.907**	0.128	0.115	0.444	0.572	**0.573**
Sq/Si	0.484	0.504	0.482	0.903	**0.914**	0.094	0.092	0.087	0.391	**0.707**
Average	0.508	0.542	0.720	0.925	**0.936**	0.147	0.143	0.273	0.622	**0.726**
Ranking	4.5	4	3.5	1.75	**1.25**	4	4.5	3.5	2	**1**

Table 4.4: The table shows the AUROC and AP obtained by each method on 4 multivariate time series. Each dataset contains tracking of movement during indoor exercises where the normal exercise is listed first, and the anomalous exercise second. For instance, Lu/Sq contains 90 repetitions of the *lunge* exercise and 8 repetitions of the *squat* exercise.

resulting in a multivariate time series $\mathcal{T}$ of dimension 75.

We construct four multivariate time series datasets containing anomalies by randomly combining around 90 repetitions of one exercise type with 8-12 repetitions of a different type. Then, the goal is to accurately detect the minority exercise. Before applying the algorithms, we use the methodology outlined in [34] to preprocess the raw data and further reduce the number of dimensions of $\mathcal{T}$ to three. Note that the baseline algorithms are not naturally equipped to deal with multivariate time series. The straightforward solution is to compute an anomaly score for each time series separately and add the scores.

Results and Discussion. Table 4.4 shows the AUROC and AP obtained by each method on the four multivariate time series datasets as well as the ranking of each method. PBAD and FPOF outperform the other methods, with PBAD improving the AUROC and AP over FPOF with $1.3 \pm 2.7\%$ and $23.8 \pm 33.2\%$ respectively.

4.4.4 Results Q3: Anomaly Detection in Mixed-Type Time Series

Due to the lack of publicly-available, labeled mixed-type time series datasets, we construct a realistic synthetic data generator. The generator simulates the electricity production in a microgrid consisting of four energy resources: a small wind turbine, a solar panel, a diesel generator, and a microturbine. Each resource has a distinct behavior. The operator controlling the grid can take 12 discrete actions: turning on and off each of the energy resources, shutting

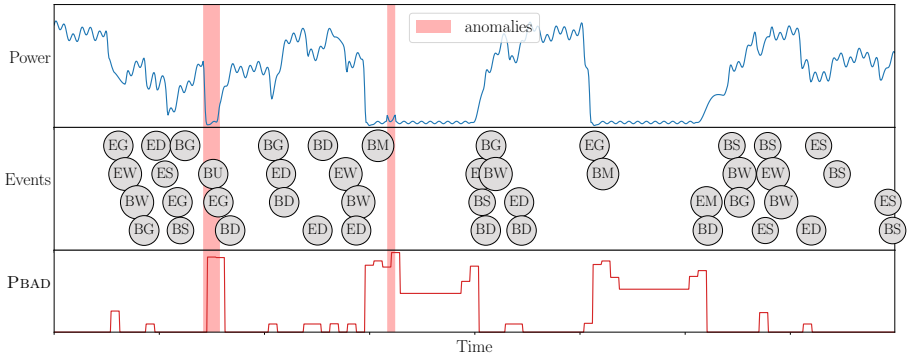


Figure 4.3: The figure shows five days of synthetic power grid data. The top plot shows continuous power output of the grid. The middle plot shows the discrete events, B and E indicate begin and end respectively, while W, S, D, G, M, and U refer to wind, solar, diesel, gas, maintenance, and shutdown respectively. The bottom plot shows the anomaly score of PBAD. The first anomaly corresponds to a discrete event (BU) that did not generate the expected power response.

down and starting up the grid, and starting and stopping grid maintenance. Then, a full year of data is generated in three steps. First, a control strategy determines every five minutes which actions to take and logs them. It is also possible to take no action. Second, the actions determine the power output of the grid at each time step, forming the continuous time series. Finally, 90 control failures are introduced in the system. These are actions not having the desired effect, e.g., starting the wind turbine does not lead to an increase in electricity production, actions without effect, and effects that are not logged. Using the generator, we generate 45 variations of mixed-type time series, each with different anomalies, and report averaged results. The full details of the generator are omitted for brevity, but can be found in the online Appendix to this paper on which this chapter is based [45].

Results and Discussion. We only ran PBAD on the mixed-type data to detect the control failures. MATRIXPROFILE and PAV cannot handle event logs, while MIFPOD and FPOF do not naturally handle mixed-type time series. However, we run PBAD three times: only on the continuous component of the synthetic data (AUROC $= 0.81 \pm 0.13$), only on the event logs (AUROC $= 0.63 \pm 0.07$), and on the full mixed-type series (AUROC $= 0.85 \pm 0.08$). This indicates that PBAD successfully leverages the information in the combined series to detect the anomalous data points. Figure 4.3 shows seven days of the power generated by the microgrid and the corresponding event log. The anomaly score generated

by PBAD is plotted underneath the data, illustrating that it can accurately detect the anomalies in this case.

4.5 Conclusion

In this chapter, we tackled the problem of anomaly detection in mixed-type time series data. The user provides flexible supervision in the form of a discrete event log accompanying the continuous time series data. The events could help explain the patterns observed in the continuous data. However, the user does not specify which events are important nor whether the events are normal or anomalous. We designed PBAD, an algorithm for mapping mixed-type time series data onto a representation suitable for applying existing anomaly detection algorithms. The algorithm works in two steps. First, it leverages frequent pattern mining techniques to mine frequent patterns from the time series data. Then, it computes the distance between each time series segment and the extracted patterns to obtain a feature vector representation.

Chapter 5

Semi-Supervised Anomaly Detection

Modern organizations use a variety of sensors to continuously monitor equipment and natural resources. Often, the obtained measurements are stored centrally in order to enable detailed monitoring and swift interventions whenever a fault or suboptimal behavior is detected. Typically, automated strategies are needed to monitor the collected data because it is not feasible for a person to monitor hundreds of machines, plants or buildings simultaneously. Such strategies have been successfully applied in many organizations, for example by using condition monitoring methods [114]. Originally, anomalous behavior in time series was detected by statistical methods like exponentially weighted moving average, or cumulative sum [26]. Often, this approach is too simplistic for complex signals and as a result more elaborate formulas were hand-crafted to differentiate between normal and abnormal behavior [141]. However, maintaining a formula-based system is nearly impossible as it needs to be modified whenever the monitored system changes. Paradoxically, even when confronted with a complex signal, many operators are able to discern patterns which indicate when a system is behaving abnormally. This insight has motivated applying machine learning to learn models of typical behavior by observing the system [79].

Anomaly detection is almost always posed as an unsupervised learning problem, the reason being that gathering labeled data is costly, difficult, and time-consuming. From a machine learning perspective, three challenges frequently arise in real-world anomaly detection tasks. First, it is difficult to treat anomaly detection as a supervised learning problem because few or no ground-truth examples of anomalous behavior are available. Furthermore, collecting all

possible types of anomalies upfront is infeasible in practice. Nor is it desirable to permit anomalous behavior for the sake of generating data as such behavior may have an expensive and negative impact (e.g., result in breaking a machine). Second, while unsupervised methods do not require labels, they make strong assumptions that are violated in practice, which makes for a less robust anomaly detector. The most widely used assumption is that infrequent behavior is anomalous, which is not always true. For instance, maintenance operations on a system can occur less frequently than the system wasting a resource. Consequently, some form of *flexible supervision* (labeling or tuning) is required in practice. Third, when analyzing the time series data, the signal requires a combination of heterogeneous descriptors.

In this chapter, we tackle the problem of automatically detecting periods of abnormal water consumption in a real-world setting at the Colruyt Group, a large retail company (>500 stores). This use case exhibits each of the aforementioned machine learning challenges. Water consumption has a complex pattern as it is a combination of fixed behavior (e.g., maintenance, opening hours that differ daily), stochastic behavior due to external influences (e.g., restroom visits, number of customers in the store) and combinations of both (e.g., meat preparations, cleaning). Furthermore, several normal behaviors (e.g., maintenance) occur less frequently than some abnormalities, violating the assumption underlying unsupervised anomaly detection. As part of Colruyt's policy for sustainable entrepreneurship, they employ energy monitoring systems that track the consumption of electricity, water, gas, etc. in their stores, office buildings and distribution sites. Reducing water usage has clear environmental benefits. For instance, 11% of Europe's population live in regions that are water stressed. Because of losses in the water supply network, as much as 30% of water is lost before it reaches the consumer in countries such as France and Spain [43]. Droughts have significantly impacted Europe in the past decades and it is expected to deteriorate [129]. Early detection of leakages, and malfunctions could help prevent these losses.

We propose a novel constrained-clustering-based approach for anomaly detection that works in both an unsupervised and semi-supervised setting. Starting from an unlabeled dataset, the approach is able to gradually incorporate expert-provided supervision to improve its performance. We evaluated our approach on real-world water monitoring time series data from supermarkets in collaboration with one of Belgium's largest retail companies. Empirically, we found that this approach outperforms their current detection system as well as several other baselines.

Contributions of this Chapter

The work outlined in this chapter, makes the following four contributions:

The **first** contribution is a constrained-clustering-based anomaly detection algorithm that works both in the unsupervised setting and some when labels are available.

The **second** contribution is SSDO (Semi-Supervised Detection of Outliers): a general algorithm for extending any unsupervised anomaly detection algorithm with a limited amount of label information. It corrects the unsupervised methods' anomaly scores by propagating known labels through the dataset.

The **third** contribution is the deployment of SSDO in a real-world water monitoring use case provided by one of Belgium's largest retailers, the Colruyt Group. This involved designing and implementing a robust application that could be used by Colruyt on a daily basis. It also entails an empirical study that shows the benefits of our proposed approach compared to other anomaly detection approaches as well as the company's baseline approach; and a discussion of the deployed system architecture.

The **final** contribution is an implementation of SSDO available at: `https://github.com/Vincent-Vercruyssen/anomatools`

Scientific Publications

The content of this chapter is based on the following publication [145]:

V. Vercruyssen, G. Verbruggen, K. Maes, R. Bäumer, W. Meert, and J. Davis (2018). Semi-supervised Anomaly Detection with an Application to Water Analytics. In *IEEE International Conference on Data Mining (ICDM)*, Singapore, pp. 527-536.

Problem Statement

The problem addressed in this chapter is formalized as follows:

Given: A continuous time series $T = \{(t_1, v_1), \ldots, (t_n, v_n)\}$, where t_i is a time stamp and v_i is the water consumption at time t_i for a retail store.

Do: Identify periods of anomalous consumption in T.

5.1 Methodology

Outline of the Approach. SSDO computes for each time series segment $S_{t,l}$ of T an anomaly score. The method has three major steps. First, each segment $S_{t,l}$ is transformed to an instance x in feature vector format. Second, an initial anomaly score is assigned to each instance based on its position with respect to the data distribution found using constraint-based clustering. Third, if some time series segments (and thus the corresponding instances) have known labels, there is a label propagation phase. Each instance's anomaly score is updated based on nearby instances' known labels. The approach employs an active learning strategy to acquire more labels, and the process is repeated whenever additional data or labels are provided.

Section 5.2 will discuss the practical details about applying the SSDO algorithm to the water usage monitoring use case. The transformation function maps the raw consumption data to feature vectors. We chose features that describe different characteristics of the signal (e.g., statistical properties, motifs) as well as contextual variables (e.g., day of week) in order to capture the heterogeneity of the time series signal. Initially no labels were available. They were gradually obtained through the active learning strategy. SSDO improves with the feedback and is increasingly able to distinguish between more subtle differences in behavior.

5.1.1 Constrained-based-clustering Anomaly Score

Here, we develop a new method for computing an anomaly score using clustering. We employ clustering because it: (a) allows deriving an anomaly score in a purely unsupervised manner, and (b) allows assigning an anomaly score not only to each instance but also the cluster to which it belongs. If labels are available, we are able to incorporate them to guide the clustering in the form of constraints. We do so by including one *cannot-link* constraint between each anomalous and normal instance. We do not consider *must-link* constraints, because there is no guarantee that either all normal behavior or all anomalous behavior is similar. For example, there can be different types of anomalies and they should not be forced to appear in the same cluster. We employ the *constrained k-means* clustering algorithm (COP k-means) [147], which reduces to standard k-means if no constraints are provided. Note that any constrained clustering algorithm that defines a center for each cluster could also be used. Applying COP k-means to the data yields a set of k clusters $\mathcal{C} = \{C_1, \ldots, C_k\}$.

A standard assumption in unsupervised anomaly detection is that anomalies look "different" from normal instances. From a geometric perspective, this means

they will be far away from normal instances. From a statistical perspective, this means they will lie in a low-density region of the input space. This assumption gives us three intuitions that could be helpful in identifying anomalies: one on the instance level, and two on the cluster level. First, the more an instance deviates from its cluster center, the more likely it is to be anomalous. Second, the more a cluster center deviates from other cluster centers, the more likely it is to represent anomalies. Third, the smaller a cluster is, the more likely it is to contain anomalies. If we consider an instance's cluster to be the *context* to which it belongs, the latter two intuitions allow us to say something about the anomalousness of a context.

Based on these intuitions, each instance x gets the following anomaly score:

$$a_u(x) = 1 - g\left(\frac{point\_deviation(x) \times cluster\_deviation(x)}{cluster\_size(x)}; \gamma\right) \tag{5.1}$$

where each of its three components are defined next.

The **point deviation** captures the deviation of x from its cluster center, normalized by the maximum deviation from the center over all data points within its cluster:

$$point\_deviation(x) = \begin{cases} \frac{\delta(x,c(x))}{\nu(x)} & \text{if } |C(x)| > 1 \\ 1 & \text{otherwise} \end{cases} \tag{5.2}$$

$$\nu(x) = \max_{x_i \in C(x)} \delta(x_i, c(x)) \tag{5.3}$$

where $C(x)$ represents all the instances in the same cluster as x, δ is the (Euclidean) distance function, and $c(x)$ is the center of the cluster x belongs to. If $C(x)$ contains a single instance, the point deviation is 1. Because $\nu(x) \neq 1$, but rather chosen adaptively, the resulting score is truly local.

The **cluster deviation** measures how much cluster center $c(x)$ differs from the other cluster centers weighted by the maximum inter-cluster distance:

$$cluster\_deviation(x) = \begin{cases} \frac{\min_{1 \leq i \leq k \wedge c(x) \neq c_i} \delta(c(x), c_i)}{\max_{1 \leq i,j \leq k} \delta(c_i, c_j)} & \text{if } k > 1 \\ 1 & \text{otherwise} \end{cases} \tag{5.4}$$

where c_i represents the center of cluster i, and k is the number of clusters. If there is only one cluster, the cluster deviation is 1.

The **cluster size** expresses the size of the cluster x belongs to relative to the largest found cluster:

$$cluster\_size(x) = \frac{|C(x)|}{\max_{1 \leq i \leq k} |C_i|} \tag{5.5}$$

where C_i is the set of instances assigned to cluster i. Again, this score reduces to 1 if there is only one cluster.

The **squashing function** g is used to map the anomaly score into the range between 0 and 1, with a higher value representing a greater degree of anomalousness. We use a squashing function from the exponential family:

$$g(x;\lambda) = 2^{-\frac{x^2}{\lambda^2}}. \tag{5.6}$$

The parameter λ is set such that the percentage of instances that have a score greater than 0.5 after mapping, is equal to the user's requested percentage of instances that the system should flag as anomalous if no labels are available (the contamination factor).

Compared to CBLOF this approach has several advantages: (1) it requires parameters, (2) it can handle label information in the form of constraints, (3) it is a local anomaly detection method, and (4) it reflects that also an instance's context (cluster) can be anomalous.

5.1.2 Updating Anomaly Scores via Label Propagation with Ssdo

Given any unsupervised anomaly score, SSDO updates this score via label propagation to more accurately reflect the labels provided by the user. SSDO necessarily starts from precomputed scores because most of the time the label information in itself is not sufficient to build a performant anomaly detector. First, we only have a limited amount of labels. Furthermore, the active learning strategy does not necessarily query the labels of instances that form a representative subsample of the data. It selects the query instances based on their usefulness for improving the underlying model, resulting in a biased subsample of the dataset.

SSDO can be combined with any unsupervised anomaly detector, provided its anomaly scores are normalized similarly as described in Equation 5.6. The unsupervised model provides an initial model for the full dataset, while the available labels are used to correct the unsupervised model's biases *locally*. In this chapter, we use the constrained-based-clustering model from Section 5.1.1 as unsupervised model.

In order to maximize the value of having labels, we update the initial unsupervised anomaly score by combining it with a score obtained by performing

label propagation. The updated score obtained for each instance x is:

$$a_l(x) = \frac{1}{Z(x)} \left[a_u(x) + \alpha \sum_{x_i \in L_a} g(\delta(x, x_i); \eta) \right] \tag{5.7}$$

$$Z(x) = 1 + \alpha \sum_{x_i \in L} g(\delta(x, x_i); \eta) \tag{5.8}$$

where L_a is the set of labeled anomalous instances and L is the set of all labeled instances. For the squashing function $g(\cdot; \eta)$, η is the harmonic mean of the k-distances of the instances. We use the harmonic mean because it is less sensitive to noise in the data. The goal of g is to make the influence of the label propagation depend on the distance between a labeled and unlabeled instance. That is, a labeled instance will have more effect on the score of close-by instances than on far away instances.

$Z(f)$ is a normalization factor such that the final score is between 0 and 1. Finally, α controls the impact of the user labels versus that of the clustering step. If α is larger than 1, the clustering is overruled immediately whereas if it is smaller, multiple labels are needed to overrule the clustering. Controlling the labels' influence is desirable in practice because the user may incorrectly label instances, resulting in noisy labels [125]. Together, α and the chosen k-distance, determine the speed with which the constraint-clustering-based model can change its mind (see Section 5.4).

5.1.3 Querying for Labels

As previously mentioned, in the water consumption domain there are several types of normal consumption patterns that occur less often than anomalous patterns. It is difficult to distinguish between infrequent normal behavior and anomalies in a purely unsupervised manner. Therefore, acquiring labels offers the potential to improve performance. To do so, we employ the commonly used *uncertainty sampling* active learning framework [82]. In uncertainty sampling, the algorithm selects the instance with the least certain predicted label and asks the user to label that instance. In our setting, this entails selecting the instance whose anomaly score is closest to 0.5.

5.2 Applying Anomaly Detection to Water Consumption Time Series Data

Water Consumption Data. Colruyt is a retail group operating a large number of stores. The provided continuous time series data report the water consumption over the last three years for 20 Colruyt stores across Belgium. In each store, the consumption is measured every five minutes. There are no missing or incorrect measurements. Initially, none of the data was labeled.

Modeling Granularity. We perform anomaly detection on a store-by-store basis, and within each store we further subdivide the data T into non-overlapping segments of one hour (i.e., 00:00-01:00, 02:00-03:00, etc.). Working on the store level is important because each store has a different set of characteristics such as size, location, what services are offered, and opening hours, all of which affect water consumption. Larger stores have more employees and hence use more water. A butchery uses a large amount of water, however, not all stores offer this service. Furthermore, each store has a maintenance system, which is specifically configured for that store.

We divide the data within a store into the non-overlapping one hour segments for three reasons. First, and most importantly, this period was chosen because it is the smallest unit in which enough data is available for domain experts to detect typical patterns and label data. Second, time of day is an important feature that is implicitly captured by this partition. Finally, the anomalies have to be communicated to an expert, who may be monitoring multiple stores, and varying or "non standard" (e.g., 12:13-13:51) partitions reduce interpretability.

Feature Construction. Since describing the data requires heterogeneous descriptors, each one hour segment is transformed into a feature vector that describes the characteristics of the signal during that particular segment. The following classes of features are constructed:

Summary statistics: We compute the following eight statistical features: max, min, mean, median, standard deviation, skewness, kurtosis, and entropy.

Descriptive features: We consider three categorical features: the day of the week, the month of the year, and whether a day is a Belgian bank holiday or not. These categorical features are dealt with using one-hot encoding.

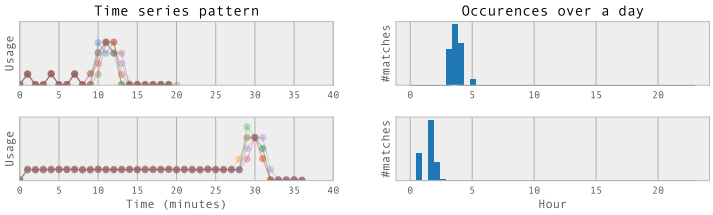


Figure 5.1: Two patterns resulting from time series clustering. Subtly varying sequences are clustered together. A shape feature is the prototype of such a cluster. Both examples shown match typical maintenance operations.

Shape features: The data contains certain operations, such as the maintenance, that have a typical usage pattern. While these operations typically begin around the same time, the number of days between successive operations is not fixed. Recognizing and reasoning about these shapes is important. To automatically identify these interesting shapes, a time series clustering approach is taken.[1] Because the duration of interesting usage patterns is unknown a priori, the data is coarsely discretized and given to the *prediction by partial match* algorithm [28] to efficiently detect coarse patterns that appear at least two times. This algorithm removes all the random sequences and leaves us with candidate patterns. Next, we apply time series clustering to all subsequences in the original data that correspond to a candidate coarse pattern [112]. The distance measure used for clustering, DTW, compensates for subtle variations introduced by the sensor's sampling strategy. Additionally, for each cluster we compute the histogram of occurrences throughout the day. We consider all the shapes that are a prototype of a cluster with at least five elements and where the entropy of the histogram is less than two to prefer patterns that are localized in time. For each of these shapes, there is one feature whose value is the DTW measure from the observed data to the shape. Figure 5.1 shows an example of two patterns from two different stores that were found and typify a maintenance operation.

Applying Ssdo to the Water Consumption Data. The feature-vector representation of the water usage data works as-is with the Ssdo algorithm, given two important design choices. First, within each store, we construct 24 segment groups, one for each one hour interval and detect anomalies separately in each group. The grouping of segments spanning the same hour interval

[1] Implementation available: `https://github.com/wannesm/dtaidistance`

on different days, is motivated by the importance of time of day as a factor governing the water consumption (e.g., the opening hours of a store affect consumption). Second, the active learning aspect functions slightly differently in practice due to the temporal nature of the data and its partition into hour long segments. When querying the user, we display a plot showing the water usage during the entire day (from 00:00 to 24:00) as well as the usage during the previous two days, because the context (e.g., time of day, day of week, etc.) is important to help the user determine the label. We highlight the hour of interest as selected by the active learning strategy. However, the user may select any contiguous block of time to label as either normal or anomalous. Hence the user can provide feedback on multiple segments simultaneously, which means the interface can naturally cope with anomalies spanning several segments. Labeling is always done on the level of a segment, so by construction anomalies are always a multiple of the segment length. This is analogous to multi-instance learning: if any behavior in a segment is anomalous, then the segment is anomalous. Figure 5.5 illustrates this process.

5.3 Related work

This section briefly describes the most related work within the context of this specific chapter. Related work on unsupervised and semi-supervised anomaly detection is discussed in Chapter 2.

Natural Resource Analytics. For water analytics, research has been directed mainly towards detecting anomalies in water distribution, or water quality over time [111]. Fagiani et al. [44] compares several unsupervised methods (*Gaussian mixture models*, *hidden Markov models*, and OC-Svm) for leakage detection in water and natural gas grids. Patabendige et al. [105] applied Knno to find outliers in water usage data from aquatic leisure centers [113]. That work uses a much more simplistic set of statistical features to characterize the time-series signal and can therefore not be applied directly on the case presented in this work. The present chapter contains an extensive empirical evaluation comparing to multiple baselines, including Knno, but with more complex features. Furthermore, the models mentioned in the above related work do not have the ability to take expert feedback into account.

More research has studied anomaly detection for the consumption of other natural resources, specifically (household) electricity consumption. Jakkula et al. [67] compares statistical with unsupervised clustering-based techniques for detecting periods of unexpected consumption. Janetzko et al. [68] employs both

a straightforward load-forecasting technique, and an unsupervised clustering-based technique for outlier detection in electricity consumption data. Chou et al. [27] constructs a hybrid neural net ARIMA model to predict electricity consumption, and flag anomalies that deviate from the prediction using the two-sigma rule. Contrary to all of the above, our approach works in both the unsupervised and semi-supervised setting.

5.4 Experiments

We report the results on the evaluation of SSDO as it is was in use by Colruyt in 2018. The empirical evaluation of SSDO focuses on three questions:

Q1: how does SSDO perform compared to unsupervised and other semi-supervised anomaly detection algorithms?

Q2: how do the label propagation parameters α and k affect SSDO's performance with increasing user feedback?

Q3: how does each feature category contribute to the overall predictive performance?

5.4.1 Experimental Setup

Compared Approaches. We experimentally compare SSDO with five state-of-the-art approaches, divided into three categories:

Real-world baseline. COLRUYTBASE is a set of hand-coded rules encoding expert knowledge Colruyt used to detect anomalous water consumption: if a rule is violated, the system flags an anomaly.[2]

Unsupervised anomaly detectors. We consider three techniques: KNNO, LOF, and CBLOF. The latter because it is cluster-based.

Semi-supervised anomaly detectors. SSAD is a semi-supervised extension of the one-class support vector machine algorithm for anomaly detection. It is shown to outperform both SVDD and SVDD-NEG [55].

[2]While Colruyt employs advanced methods, namely fully supervised regression-based and load-forecasting algorithms, for monitoring other resources, applying these techniques to water consumption was challenging due its characteristics (e.g., infrequent non-anomalous patterns, stochastic behavior during the day).

These baselines were selected based on recent extensive empirical evaluations of outlier detection algorithms (see also Section 5.3). Campos et al. show that, in practice, KNNO and LOF perform exceedingly well [15]. In particular, KNNO has achieved good results for water monitoring [105].

Water Consumption Data. Colruyt Group experts have been labeling segments in 10 of the 20 stores during 2016 and 2017. In each store we group the segments associated with the same hour. For the evaluation in this chapter, we consider only the groups containing ≥ 50 labeled segments. This means there are labels for at least 50 unique days in each segment group. However, which particular days are labeled varies by segment group because the expert does not have to label all hours in a day. Four stores have segment groups that fulfill this requirement with the stores having 24, 24, 23, and 10 groups, for 81 groups in total.

Experimental Setup. The goal of the experiment is to evaluate how the anomaly detection method performs as more data is labeled. We perform the following experiment for each of the four stores separately. First, within each segment group, we split the labeled segments into train set D_{train} and test set D_{test} using a stratified fifty-fifty split. Second, the labels in D_{train} are observed incrementally, which simulates beginning in an unsupervised setting and an expert gradually providing labels via the user interface (see Section 5.5). We initialize $D'_{train} = \varnothing$ and iterate over these three steps: (1) For each segment group, run the anomaly detection algorithm over all instances in the group using the labels in D'_{train}; (2) Evaluate the results on the test set D_{test}; (3) Select ten labeled days in accordance with the active learning strategy and add the labels to D'_{train}. These steps are repeated until D'_{train} contains all the labels in D_{train}. For each method, we pick the hyperparameter setting that maximizes training set AUROC in the final iteration. Note that this means that the unsupervised techniques use all the labeled instances in D_{train} to set their hyperparameters. Finally, every experiment is repeated 20 times, with different, randomized train-test splits, and the results are averaged over the 20 runs.

Evaluation Metrics. In step (2), the anomaly detection algorithm outputs a ranking over the segments in D_{test} from most to least anomalous. We evaluate these rankings using AUROC. Additionally, we define $\text{RE}_{x\%}$ as the effort required by the user to find x percent of the total number of anomalies in the test set, going through the ranking. For instance, if $n = x$ percent of the total number of anomalies in the test set, and the user needs to inspect only the first n consecutive instances in the ranking to find n anomalies, $\text{RE}_{x\%}$ is 1. If he

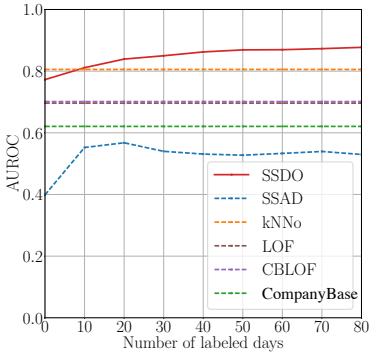
needs to inspect $2n$ instances, $\text{RE}_{x\%}$ is 2, and so on. Lower values of $\text{RE}_{x\%}$ mean that the user needs to inspect fewer instances in the ranking to find $x\%$ of the anomalies.

Hyperparameters. for each method parameter, tuning is performed on the training data using grid search. ColruytBase has no parameters. Lof and Knno each have one parameter k: the number of nearest neighbors, which is optimized over the interval $[1, 30]$. Cblof has three parameters: the number of clusters k which is optimized over the interval $[1, 30]$, and the two parameters required to partition the set of clusters into large and small clusters, which are set to 90% and 5, as in the original paper [62]. Ssad has three parameters: the trade-off parameter κ is set to 1 as in the original paper, and the regularization parameters η_u and η_l are picked from $\{0.01, 0.1, 1, 10, 100\}$. We use the method with a Gaussian kernel. Ssdo has four parameters: the number of clusters k is picked from the set $\{5, 10\}$, k is picked from the set $\{4, 16, 32, 64\}$, α is picked from the set $\{1, 1.5, 2.3, 4\}$, and the contamination factor is set to 5%.

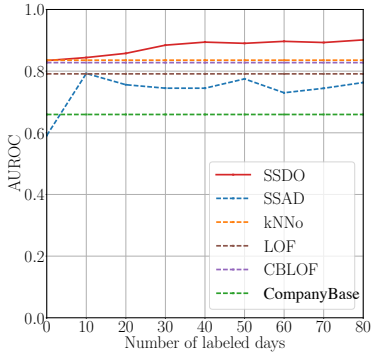
5.4.2 Results Q1: Ssdo versus State-of-the-art

Figure 5.2 shows how the test set AUROC, averaged over all stores and segment groups, varies as a function of the number of labeled days for each method. Both Ssdo and Ssad benefit from adding labels. However, Ssdo achieves a much higher AUROC than Ssad in the unsupervised setting and Ssad is never able to approach Ssdo's performance. On average, Ssdo's AUROC improves by 1.8 percent after adding 10 labeled days, 5.6 percent after 40 labeled days, and 6.6 percent after 80 labeled days. In aggregate, Ssdo outperforms Ssad, Lof, and ColruytBase, regardless of the number of labels provided. Ssdo outperforms Knno on three stores and achieves equivalent performance on the fourth. In fact, on three stores Ssdo achieves equivalent or better performance than Knno with very few (i.e., 10 labeled days) or even no labels. Ssdo also outperforms Cblof by a wide margin on three stores and a small margin on the fourth.

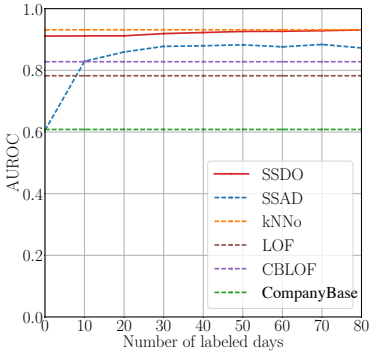
Table 5.1 shows the average AUROC ranks over all 81 segment groups for each method. We apply the hypothesis tests suggested by Demsar [35]. The Friedman test on the AUROC ranks rejects the null-hypothesis that all methods perform the same for both 0 and 80 labeled days (p-values < 0.001). With $p = 0.05$, the Nemenyi post-hoc test on the ranks finds that Ssdo is significantly better than Ssad, Lof, and ColruytBase with no labels. After 80 labeled days, Ssdo is also significantly better than Cblof. Only Knno remains competitive, albeit



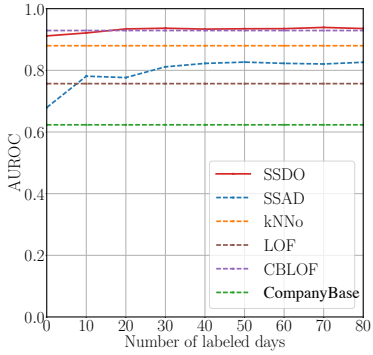
(a) Average AUROC store 1.



(b) Average AUROC store 2.



(c) Average AUROC store 3.



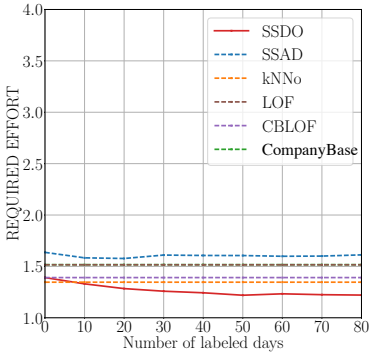
(d) Average AUROC store 4.

Figure 5.2: Each store's average test set AUROC as a function of the number of days labeled by the user for each method. SSDO consistently performs better than SSAD. SSDO also outperforms the unsupervised baselines in 3/4 stores, and ties with KNNO for first place in the last.

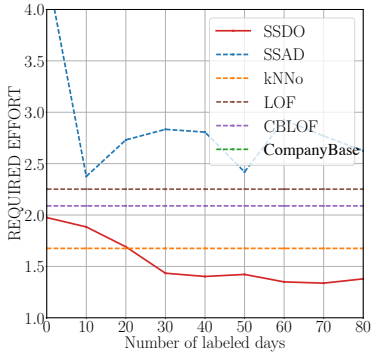
narrowly, as the difference in average ranks of KNNO and SSDO is slightly below the critical difference for the Nemenyi test.[3]

Figure 5.3 shows how the test set $RE_{95\%}$, averaged over all stores and segment groups, varies as a function of the number of labeled days for each method. SSDO consistently outperforms SSAD, both with and without labels. Again,

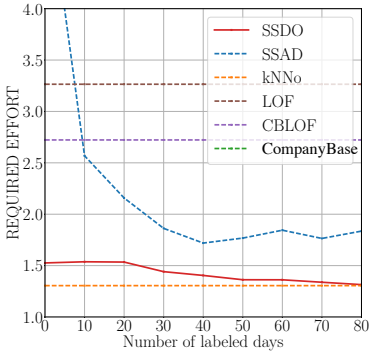
[3]The critical difference for the Nemenyi test at $p = 0.05$, 81 datasets, and six methods is 0.838.



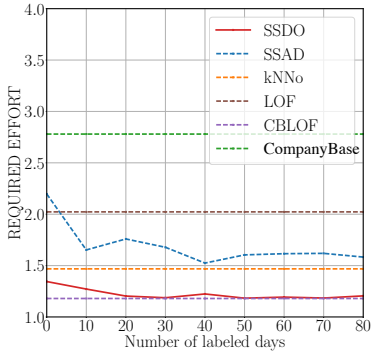
(a) Average $RE_{95\%}$ store 1.



(b) Average $RE_{95\%}$ store 2.



(c) Average $RE_{95\%}$ store 3.



(d) Average $RE_{95\%}$ store 4.

Figure 5.3: Each store's average test set $RE_{95\%}$ as a function of the number of days labeled by the user for each method. In plots (b) and (c), $RE_{95\%}$ for ColruytBase is > 4. Ssdo outperforms Ssad, Lof, and ColruytBase in each store. In each store, adding user feedback leads to a reduction in $RE_{95\%}$ for Ssdo.

Ssdo is better than Knno in three stores and ties for first place in the last. Ssdo's required effort is also substantially lower than Cblof's on three stores and slightly better on one. On average, Ssdo benefits from user feedback as having 10 labeled days reduces the required effort by 3.4 percent and adding 40 labeled days results in a 14 percent reduction.

Table 5.2 shows the average $RE_{95\%}$ ranks over all 81 segment groups for each

Method	0 labeled days	80 labeled days
SSDO	**2.290 ± 1.165**	**1.889 ± 0.903**
KNNO	2.481 ± 1.123	2.630 ± 1.012
CBLOF	2.907 ± 1.412	3.123 ± 1.594
LOF	3.654 ± 1.344	3.988 ± 1.356
SSAD	5.043 ± 1.052	4.580 ± 0.954
COLRUYTBASE	4.623 ± 1.832	4.790 ± 1.945

Table 5.1: Average AUROC rank ± SD for each method across all 81 segment groups. SSDO ranks better than the other baselines, both without labels and after 80 labeled days.

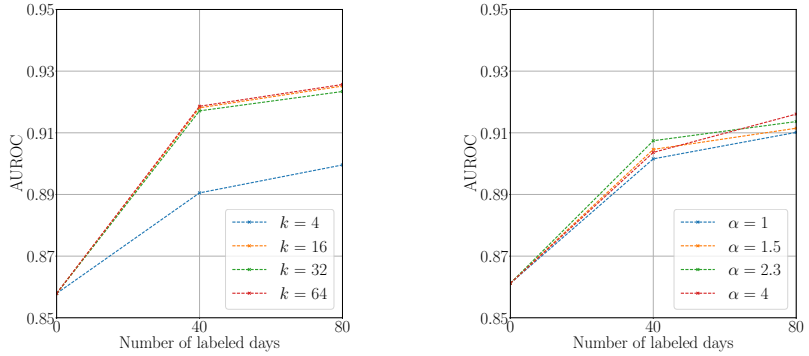
Method	0 labeled days	80 labeled days
SSDO	**2.309 ± 1.244**	**1.802 ± 0.852**
KNNO	2.642 ± 1.240	2.852 ± 1.278
CBLOF	2.741 ± 1.438	3.037 ± 1.644
LOF	3.815 ± 1.325	4.185 ± 1.177
SSAD	4.815 ± 1.156	4.284 ± 1.021
COLRUYTBASE	4.679 ± 1.818	4.840 ± 1.902

Table 5.2: Average $RE_{95\%}$ rank ± SD for each method across all 81 segment groups. SSDO ranks better than the other baselines, both without labels and after 80 labeled days.

method. The Friedman test on $RE_{95\%}$ ranks indicates significantly different performances between the methods, both when 0 or 80 days are labeled (p-values < 0.001). At $p = 0.05$ and 80 labeled days, the Nemenyi test finds that SSDO is significantly better than all baselines, including KNNO. This is an important result, because Colruyt experts need to inspect a large number of stores on a daily basis (Section 5.5).

5.4.3 Results Q2: Impact of the Label Propagation Parameters

SSDO's parameter k controls how many instances are updated by propagating the label of a single labeled instance. To assess SSDO's sensitivity to this parameter, we vary k while keeping all the other parameter fixed. Figure 5.4a shows SSDO's average test set AUROC for each value of k as function of the number of labeled days. Setting $k = 4$ results in too little propagation, and



(a) Impact of k.

(b) Impact of α.

Figure 5.4: The impact of the two label propagation parameters on the AUROC, averaged over all segment groups: (a) shows the impact of k; (b) shows the impact of α.

this failure to fully exploit the labeled instances degrades SSDO's performance. However, all other values of k result in nearly identical performance, regardless of the number of labeled days.

SSDO's parameter α controls the weight given to the unsupervised and label propagation components of an instance's anomaly score. To assess α's impact on SSDO's performance, we vary it while keeping all the other parameter fixed. Figure 5.4a shows SSDO's average test set AUROC for each value of α as function of the number of labeled days. The results indicate that α's value only has a very limited influence on SSDO's overall performance. Note that we have not introduced noise in the expert-provided supervision.

5.4.4 Results Q3: Importance of the Features

To assess the value of adding each feature category, we run SSDO with different combinations of feature categories and report $\text{RE}_{100\%}$, averaged over all segment groups. SSDO with only summary statistics yields the worst average $\text{RE}_{100\%}$ of 2.84. Combining statistics with descriptive features reduces required effort to 2.44 on average. Combining statistics with shape features reduces effort to 2.35. SSDO with all features yields the lowest $\text{RE}_{100\%}$ of 2.30, a 19 percent reduction versus using only summary statistics.

5.5 Deployment in the Real World

The described system was initially deployed to monitor water usage in 20 stores operated by the Colruyt Group in late 2016. It is used on a daily basis to do a post-mortem analysis of the previous day's water usage in each of these stores. The system runs as web-based service that returns a ranking of potentially anomalous periods of usage. Next, we describe the system's day-to-day use, outline its architecture, provide illustrative examples of its deployed performance, and discuss lessons learned.

5.5.1 Day-to-Day System Operation

Each day at 8am, Colruyt engineers generate a CSV file that contains the previous day's water consumption, in five-minute increments, for each of the 20 supermarkets. These files are manually uploaded via a web-based interface and then SSDO is applied to the data. The algorithm returns a global anomaly ranking over all stores for each hour of the previous day. We additionally report excess water consumption as the surplus consumption for a specific hour and store over the expected normal usage for that hour and store. The technical department, which is responsible for maintenance of the supermarkets, reviews this information each morning and plans an action according to the cost of the leakage. For a large leakage, there is a direct intervention. For a small leakage, it is added as task for the technician's next scheduled visit. Finally, the technical department gives feedback via the web-interface as to whether the detected anomalies are indeed anomalies, in order to improve the performance of SSDO.

5.5.2 System Architecture

The system consists of two parts: a web-interface which is connected to a database running on a server. The end-user can access the information in the database through the web-interface, which is separated into two main views: the *predict* view and the *label* view. The *predict* view allows the user to inspect the outcome of the anomaly detection algorithm by selecting a store (or all stores) and a date range. A ranking of the days within the selected range, ordered from most to least anomalous, is returned. The user can easily give feedback by interactively selecting a part of the signal, and indicating the appropriate label, which is then automatically stored in the database. The *label* view presents the segments selected by the algorithm for feedback. The user can provide feedback on the requested segments, or select a range interactively by click-and-drag on

(a) The system queries the user for feedback on a single segment.

(b) By click-and-drag, the user labels multiple hours simultaneously.

Figure 5.5: Screenshots of the web-interface's *label* view. Top: The system queries the user for feedback on one particular segment. Bottom: The user decides to label a large part of the day as normal behavior in one go.

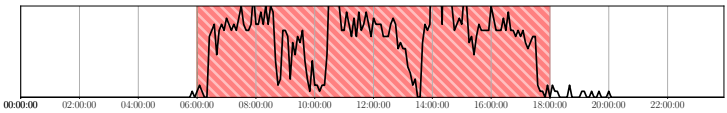
the time series (see Figure 5.5). Finally, the interface also allows for uploading new data.

The database stores the time series data, user feedback, and predictions. When adding new data, we segment the data hourly, and convert the resulting segments into the feature-vector representation, stored in the database. For each store that currently has available data in the database, we run our anomaly detection algorithm on each of the 24 segment groups, and store the predictions.

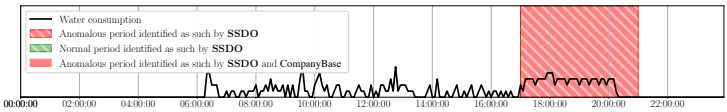
5.5.3 Value of Deployment

SSDO is able to detect anomalies that the value-based alarms (COLRUYTBASE) cannot. Hence, deploying SSDO has both reduced costs for Colruyt and provided an environmental benefit through minimizing water losses. Figure 5.6 illustrates three examples that contrast the behavior of the two systems.

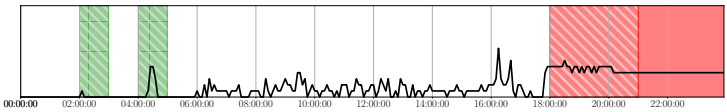
The advantages of SSDO over COLRUYTBASE are particularly evident during the day where the water consumption fluctuates in a stochastic manner. Figure 5.6a shows an example of an anomaly detected by our algorithm during normal opening hours that was missed by the value-based alarms. Figure 5.6b shows



(a) A Tuesday with anomalous consumption during the day, when the store is open.



(b) A Wednesday with a small leak during the last 4 hours of the work day.



(c) A Thursday with a normal maintenance pattern (i.e., water softener) between 2am and 4am, and a leak starting at 6pm.

Figure 5.6: Three days from different stores when SSDO correctly identified periods of anomalous water consumption (the hours shaded red) not picked up by COLRUYTBASE. The day in plot (c) starts with an infrequent maintenance pattern between 2am and 4am (hours shaded green). SSDO correctly identifies this period as normal behavior.

an example of a small leakage during operating hours detected by SSDO but not COLRUYTBASE. Small leaks are difficult to detect, particularly near the end of the day, when the activities of the in-store butchery (e.g., cleaning) increase consumption for a few hours.

Figure 5.6c shows a day that starts with an infrequent, automatic maintenance pattern that results in a small amount of water usage when there is typically no consumption. SSDO correctly identifies this as normal behavior by learning from the user feedback whereas experts must hand code this domain knowledge into the COLRUYTBASE system. At the end of the day, there is a leak that continues into the following day. SSDO identifies the leak three hours earlier than the COLRUYTBASE does.

5.5.4 Lessons Learned: Challenges and Solutions

We summarize four take-away messages. First, feature fusion is important. The behavior of the water consumption depends on various external influences, some can be inferred (e.g., opening hours) but others cannot be measured

directly in a cost efficient manner (e.g., the number of toilet visits, what meat needs to be prepared, etc.). This often results in stochastic behavior. Therefore, in Section 5.2, we use heterogeneous descriptors to characterize the water consumption time series. Second, feature construction is important. Measurements are only recorded at certain time intervals by pulse-based water flow meters. The minimum recorded water consumption is equal to a single pulse. This is a common strategy to reduce the data to transfer, but introduces quantization noise. As a result two identical usage scenarios might show different measurements because they are triggered at different times. We tackled this in Section 5.2 by computing summary statistics and applying fuzzy clustering of time series to find typical shapes. Third, both anomalous and normal patterns can be recurring (e.g., certain types of leaks and maintenance patterns). On the other hand, various distinct patterns constitute normal behavior, some of which may occur less frequently than the anomalous patterns (e.g., specific maintenance operations). In Section 5.1.2, we use user feedback to correct the assumptions of unsupervised anomaly detection. Fourth, it is impossible to obtain fully labeled data (e.g. time constraints). At most, a small amount of normal and anomalous behavior will be labeled. In Section 5.1.3, we employ an active learning strategy to guide the user to provide feedback that is helpful to the detection algorithm, and allow the user to provide labels in a simple and efficient manner.

5.6 Conclusion

In this chapter, we tackled the problem of semi-supervised anomaly detection on water consumption data. The user provides feedback in the form of binary labels guided by an active learning strategy. We designed SSDO, an algorithm for augmenting the anomaly scores of any unsupervised anomaly detector with a limited amount of label information. First, it squashes the anomaly scores of the unsupervised detector. Then, it propagates the known labels through the data, updating the unsupervised, squashed scores. We also designed an algorithm for deriving a constrained-clustering-based anomaly score for each instance in the dataset. Finally, we deployed these algorithms in the real world and evaluated their performance over the course of several months.

Chapter 6

Transfer Learning for Anomaly Detection

Constructing performant anomaly detectors on real-world data generally requires some labeled data. Labeled data, however, are almost always costly to obtain for anomaly detection applications. The problem becomes even worse when one considers that in real-world applications, multiple, related anomaly detection tasks are often considered simultaneously. A company, for example, is usually not interested in monitoring only one of its machines, but wants to run anomaly detection algorithms on all of its machines simultaneously.

Unsupervised anomaly detectors exploit the assumption that anomalies occur infrequently to identity them, which means they fall in low-density regions of the input space, or they are far away from normal instances. However, real-world data regularly violate this assumption, degrading the unsupervised approaches' performance (e.g., system maintenance can occur infrequently and irregularly, but is not anomalous). Labeled data offers the possibility to correct the mistakes made by unsupervised detectors (see Chapter 5). Unfortunately, a fully supervised approach to anomaly detection is infeasible due to the fact that collecting examples of real-world anomalies is expensive (e.g., a machine breaking), meaning that is not a viable strategy to permit anomalous behavior for the sake of data generation. This has spurred interest in semi-supervised approaches to anomaly detection, usually in conjunction with active learning to efficiently collect the labels.

Real-world anomaly detection tasks often involve monitoring numerous assets, each of which is only slightly different. Such a situation may arise when

monitoring machines in a factory, resource usage in a chain of retail stores, wind turbine farms, *etc.* These use cases entail monitoring a large number of assets as a big retail chain could have 100s if not 1000s of stores. Therefore, even when using a strategy like active learning, it may be impossible to collect labels for each individual asset.

The fact that one has multiple, related anomaly detection tasks also leads to an opportunity: iit may be possible to employ *transfer learning* to transfer labeled instances from a related anomaly detection task to the problem at hand. Intuitively, the same behavior observed in different datasets, should only be labeled once. This could then alleviate the need to collect labels for each task separately. Despite the promise of applying the transfer learning paradigm to anomaly detection, this idea has not been studied yet in the Literature. This chapter formally introduces the problem of transfer learning for anomaly detection and designs a number of tested algorithms for tackling this problem.

Contributions of this Chapter

The work outlined in this chapter, makes the following four contributions:

The **first** contribution consists of two different algorithms for selecting labeled source instances that can be transferred to the target dataset: CBIT (Cluster-B-ased Instance Transfer), and ITRADE (Instance-TRansfer for Anomaly DEtection).

The **second** contribution is LOCIT (LOcalized Instance Transfer): a novel instance-based transfer learning algorithm tailored towards anomaly detection. LOCIT selects a subset of the labeled source instances to transfer to the target dataset by picking those instances that have similar localized data distributions in both the source and target domain.

The **third** contribution is SSKNNO (Semi-Supervised K-Nearest Neighbour Outlier detection): a semi-supervised nearest-neighbor algorithm that considers both the transferred, labeled source instances and the unlabeled target instances to assign an anomaly score to each target instance.

The **fourth** contribution is the development of a benchmark for transfer learning for anomaly detection as well as an extensive experimental evaluation of LOCIT both on this benchmark and a real-world dataset.

The **final** contribution is an implementation of LOCIT available at: `https://github.com/Vincent-Vercruyssen/LocIT`

Scientific Publications

The content of this chapter is based on the following publications [143, 146]:

V. Vercruyssen, W. Meert, and J. Davis (2020). Transfer Learning for Anomaly Detection through Localized and Unsupervised Instance Selection. In *Proceedings of the* 34^{th} *AAAI Conference on Artificial Intelligence*, New York City (US), vol. 34(4), pp. 6054-6061.

V. Vercruyssen, W. Meert, and J. Davis (2017). Transfer Learning for Time Series Anomaly Detection. In *Proceedings of the Workshop and Tutorial on Interactive Adaptive Learning at ECML PKDD*, Skopje (Macedonia), pp. 27-37.

Problem Statement

The problem addressed in this chapter is formalized as follows:

Given: A (partially) labeled source dataset D_S and an unlabeled target dataset D_T from the same input-label space;

Do: Assign an anomaly score to each instance in D_T using the information available in D_T and D_S.

6.1 Methodology

Outline of the Methods. Several flavors of transfer learning distinguish themselves in the way knowledge is transferred between source and target domain. In this chapter, we limit ourselves to *instance-based transfer learning*. The idea is to transfer labeled instances from the source dataset to the target dataset that are *helpful* in solving the target task.

The difficulty of transfer learning arises from the fact that the source and target domain could be characterized by concept and covariate shifts. In other words, a given instance might have a different *meaning* in each domain. For example, if the source and target data consist of power output readings gathered from a small and large wind turbine respectively, a high output reading might indicate anomalous behavior for the small wind turbine but normal behavior for the

large turbine. Thus, we need to avoid negative transfer by transferring source instances that degrade the performance of the target classifier.

The popular solution is to define a weight for each source instance based on its similarity to the target data. The weight is computed by comparing the probabilities of observing the instance in each domain. The probabilities themselves are derived from either the marginal, conditional or joint source and target probability distributions. However, since we do not have access to the target labels, we cannot compute the conditional or joint target probability distribution. Thus, unless we introduce some assumptions on the distribution of the target labels, we are restricted to comparing marginal probabilities. While this is not ideal, we can still develop methods for instance-transfer that work well.

In the following sections, variations of this core idea are proposed, tailored towards anomaly detection. Direct estimation and comparison of the marginal probability distributions is costly, difficult when the dimensionality of the data increases, and not particularly suited for anomaly detection.

6.1.1 Cbit Instance Transfer

Cbit is based on the idea that labeled source normals are *helpful* in constructing the target classifier if they fall within a large data cluster in the target domain. The implicit assumption here is that, when clustering the target data, the large clusters will capture the frequently occurring normal target instances. Thus, if a transferred source instance falls within such a cluster, it *likely* corresponds to a normal behavior in the target domain. Conversely, labeled source anomalies are only transferred if they fall outside the large data clusters in the target domain.

Cbit consists of three steps. First, the target data are clustered with the k-means algorithm. Second, the resulting set of clusters $\mathcal{C}$ is divided into a set of *large* clusters $\mathcal{C}_l$, and a set of *small* clusters $\mathcal{C}_s$ according to the following definition [73]:[1]

Definition 13. *Given a set of ordered clusters $\mathcal{C} = \{C_1, ..., C_k\}$ such that $|C_1| \geq |C_2| \geq ... \geq |C_k|$, and two numeric parameters α and β, the boundary b between large and small clusters is defined such that either of the following*

[1] This is the same approach as the Cblof algorithm.

conditions holds:

$$\sum_{i=1}^{b} |C_i| \geq n_T \times \alpha \tag{6.1}$$

$$\frac{|C_b|}{|C_{b+1}|} \geq \beta \tag{6.2}$$

$\mathcal{C}_l = \{C_i | i \leq b\}$ and $\mathcal{C}_s = \{C_i | i > b\}$ are respectively the set of large and small clusters, and $\mathcal{C}_l \cup \mathcal{C}_s = \mathcal{C}$. n_T is the number of target instances.

We define the radius of a cluster as $r_i = \max_{x_j \in C_i} \|x_j - c_i\|^2$ with c_i the cluster center of cluster C_i. According to the assumption, the large clusters capture *mostly* normal behavior and the small clusters capture *mostly* anomalous behavior. Transferred labeled instances from the source domain should adhere to the same intuitions. Each source instance x_s is assigned to a target cluster $C(x_s) \in \mathcal{C}$ such that $\|x_s - c(x_s)\|^2$ is minimized. A source instance is only transferred in two cases. One, if the instance has label `normal` and is assigned to a cluster $C(x_s)$ such that $C(x_s) \in \mathcal{C}_l$ and the distance of the instance to the cluster center is less than or equal to the radius of the cluster. Two, if the instance has label `anomaly` and fulfills either of two conditions: the instance is assigned to a cluster $C(x_s)$ such that $C(x_s) \notin \mathcal{C}_l$, or it is assigned to a cluster $C(x_s)$ such that $C(x_s) \in \mathcal{C}_l$ and the distance of the instance to the cluster center is larger than the radius of the cluster. In all other cases there is no transfer.

6.1.2 iTrade Instance Transfer

iTrade stems from the same core idea as Cbit, but generalizes the latter in two important ways. First, Cbit makes an explicit assumption about the target conditional probability distribution: large data clusters capture normal behavior and small data clusters capture abnormal behavior! While intuitive at first glance, this is troublesome for two reasons. One, the goal of transferring labeled source instances is to learn something about the conditional distribution in the target domain that we could otherwise not have known. By forcing the transferred source instances to comply with an assumption on the target conditional probability distribution, we reduce the possibility of transferring *new* knowledge. We would be better off just modeling our assumption directly when constructing the target classifier and not doing any transfer at all. Therefore, iTrade drops any assumption about the target conditional probability distribution.

Second, Cbit makes the implicit assumption that the source and target domains are not too different from each other. It has no way of discerning when the

source and target distributions are so different that any form of transfer becomes meaningless. iTrade takes a step towards making this assumption explicit by attempting to align the domains globally before locally selecting which instances to transfer.

iTrade works as follows. It starts by aligning the first-order and second-order statistics of each domain using a robust version of the Coral algorithm. Then, it selects a subset of the source instances to transfer based on how similarly the source and target domains are distributed in the local instance space. Algorithm 3 details the full iTrade algorithm.

Robust Coral. The goal of domain adaptation is to align the source and target domain in some manner that accounts for their differences while not obscuring the important structure in the data. One such method is the *correlation alignment* (Coral) algorithm [131]. It works on numerical data and it matches the source and target distributions by minimizing the difference between their first-order (mean) and second-order (covariance) statistics. After feature normalization, the means of each feature in the source and target domain are aligned and equal to zero. Coral subsequently finds a linear transformation that aligns the source and target covariance matrices. In practice, Coral reduces to a classical whitening and coloring transformation, that is: whitening the source data and recoloring it with the target covariance.

Because the estimates of the mean and covariance are highly sensitive to the presence of outliers in the data, the standard Coral approach would yield a transformation that is skewed towards these outliers. Using statistically robust estimates of the mean and covariance would help ensure that outliers do not result in a transformation where source normals are mapped to target anomalies. Therefore, we adapt the Coral algorithm to use the Fast-Mcd algorithm [119] to compute robust estimates of the source and target covariance matrices as well as the source and target means. Applying the resulting transformation results in a transformed source dataset D_S^*.

Instance Selection. Robust Coral aligns the first- and second-order statistics of the source and target data. Summary statistics, however, describe the central tendency of a dataset while ignoring small *local* changes in the data distribution. Thus, even though the source and target data are now globally aligned, there could still be large distributional differences locally. Given that we want to transfer source instances that have the same *meaning* in both domains, we only transfer a source instance if the source and target distributions are similar locally around the instance. Formally, a source instance $x_s^* \in D_S^*$ is transferred to the target dataset if it satisfies the following two conditions:

Algorithm 3: iTRADE(D_S, D_T, k, t)

Input : source data D_S, target data D_T, number of neighbors k, threshold t
Result: Instances of D_T selected for transfer.

```
// Phase (i): transformation phase
```
$D_S^* = \text{ROBUST\_CORAL}(D_S)$
```
// Phase (ii): instance-selection phase
```
$D_{trans}^* = \varnothing$
for $x_s^* \in D_S^*$ **do**
  $x_k \leftarrow k\text{-NN}(x_s^*; D_T)$
  $h(x_s^*) = \frac{\sum_{x_i \in N_k(x_s^*; D_S)} \omega(x_i; x_s^*)}{\sum_{x_i \in N_k(x_s^*; D_S) \cup N_k(x_s^*; D_T)} \omega(x_i; x_s^*)}$
  `// satisfy condition 1 and 2`
  if $|h(x_s^*) - 0.5| \leq t$ **and** $\delta(x_s^*, x_k) \leq \text{k-dist}(x_k; D_T)$ **then**
    $D_{trans}^* = D_{trans}^* \cup \{x_s^*\}$

Condition 1: First, each transferred instance should be embedded in the target data. That is, each transferred instance should be "close to" some target instances. Therefore, we only transfer an instance x_s^* if:

$$\delta(x_s^*, x_k) < \text{k-dist}(x_k, D_T) \tag{6.3}$$

where $x_k = k\text{-NN}(x_s^*; D_T)$. Intuitively, this check helps avoid introducing new types of normal or anomalous instances into the target domain that otherwise would not be present.

Condition 2: Second, we want to transfer an instance if the data density of the source and target domains are similar in the instance's neighborhood. To check this, we find an instance's k-neighborhood in both the source and target domain. If it is hard to distinguish whether the instance is closer to the source or target nearest neighbors, this gives an indication that the local densities match. Formally, we check this as follows:

$$|h(x_s^*) - 0.5| \leq t \tag{6.4}$$

where t is a user-defined threshold and $h(x_s^*)$ is:

$$h(x_s^*) = \frac{\sum_{x_i \in N_k(x_s^*; D_S)} \omega(x_i, x_s^*)}{\sum_{x_i \in N_k(x_s^*; D_S) \cup N_k(x_s^*; D_T)} \omega(x_i, x_s^*)} \tag{6.5}$$

where ω is defined as the standard weighting function:

$$\omega(x_i, x_j) = \frac{1}{\delta(x_i, x_j)^2}. \tag{6.6}$$

If $h(x_s^*)$ is close to 0.5, the instance's KNNO score will be similar in both domains. Hence, x_s^* is likely to belong to the same class (normal or anomaly) in both domains.

The final set of labeled instances transferred from the source to target domain is denoted by D_{trans}^*. Algorithm 3 details the full ITRADE algorithm.

6.1.3 LocIT Instance Transfer

The final technical contribution of this chapter is LOCIT, which improves upon ITRADE in an important way. Both ITRADE and LOCIT make the assumption that if the marginal distributions of the source and target domain are similar in given, small region of the input space, the instances sampled from the distributions in that region should have the same meaning (label). However, while ITRADE imposes this assumption via two disjoint conditions that require setting a threshold, LOCIT takes a learning approach to the problem. This has the advantage that the appropriate threshold can be learned automatically from the data. Furthermore, the learning method is more flexible and easier to interpret.

While the instance selection step in ITRADE is preceded by aligning the domains globally, this alignment step is not included as a part of LOCIT in the remainder of this chapter. Whether to include this step or not strongly depends on the application one is considering. It could very well be that applying a transformation to the source data changes the meaning of the transformed instances in an unintended and unwanted way.

Outline of LocIT. LOCIT decides in a label-agnostic way whether to transfer each source instance to the target domain by checking if the instance's local data distribution is similar in both the source and target domains. LOCIT takes an unsupervised transfer approach because the target labels are not available and the value of the source label should not influence the transfer decision. Algorithm 4 details the overall control flow of LOCIT and the following paragraphs describe each step in more detail.

An instance-transfer function $f(x_s; D_S, D_T) \mapsto \{0, 1\}$ decides whether to transfer each source instance $x_s \in D_S$ to the target domain (1) or not (0). Ideally, each transferred source instance has the same meaning (i.e., would be labeled similarly) in both domains. It only makes sense to transfer a source anomaly (normal) to the target domain if it is similar to a target anomaly (normal). However, LOCIT must make this assessment without access to any

Algorithm 4: LocIT(D_S, D_T, ψ, k)

Input : source data D_S, target data D_T,
neighborhood sizes ψ and k
Result: Anomaly score for the instances in D_T

```
// Phase (i): localized instance-based transfer
```
$F_{pos}, F_{neg} = \varnothing$
for $x_t \in D_T$ **do**
  x_n is the nearest neighbor of x_t in $D_T \setminus \{x_t\}$
  x_f is the farthest neighbor of x_t in $D_T \setminus \{x_t\}$
  $N1 = N_\psi(x_t; D_T)$ and $N2 = N_\psi(x_n; D_T)$
  $F_{pos} = F_{pos} \cup \{[\delta_1(N1, N2), \delta_2(N1, N2)]\}$
  $N2 = N_\psi(x_f; D_T)$
  $F_{neg} = F_{neg} \cup \{[\delta_1(N1, N2), \delta_2(N1, N2)]\}$
SVM = FIT_SVM(F_{pos}, F_{neg})
$D_{trans} = \varnothing$
for $x_s \in D_S$ **do**
  $N1 = N_\psi(x_s; D_S)$ and $N2 = N_\psi(x_s; D_T)$
  $f_s = [\delta_1(N1, N2), \delta_2(N1, N2)]$
  if SVM_PREDICT$(f_s) = 1$ **then**
    $D_{trans} = D_{trans} \cup \{x_s\}$
```
// Phase (ii): prediction in the target domain
```
$D^* = D_T \cup D_{trans}$
for $x_t \in D_T$ **do**
  Predict x_t's anomaly score using Equation 6.10, D^* and k

labels for the target instances. Consequently, LocIT makes the intuitive *assumption* that an instance has a similar meaning in both the source and target domain if the *local structure* of the source and target marginal distributions around the instance are similar, where the structure of the distributions is characterized by first and second order statistics.

Characterizing an Instance's Local Structure. For a given source instance x_s, LocIT defines the *localized source distribution* using the set $N_\psi(x_s; D_S)$ of x_s's ψ nearest neighbors in the source data. Similarly, the *localized target distribution* is based on the set $N_\psi(x_s; D_T)$ of x_s's ψ nearest neighbors in the target data. The assumption is that if the data distribution over $N_\psi(x_s; D_S)$ is sufficiently similar to the distribution over $N_\psi(x_s; D_T)$, x_s can be transferred, where the similarity is measured by comparing the following first and second order statistics of $N_\psi(x_s; D_S)$ and $N_\psi(x_s; D_T)$:

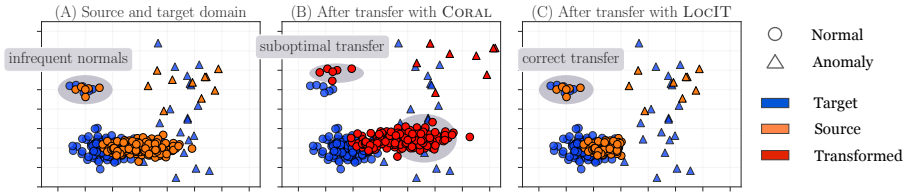


Figure 6.1: Both the source and target domain (panel A) contain a small (infrequent) cluster of normal instances, which many unsupervised algorithms would incorrectly flag as anomalous. Hence, transferring labeled instances from the source to the target could help. CORAL (panel B) transforms the source data to align the global statistics between the domains. However, outliers skew the correction, resulting in a suboptimal mapping. Here, source normals are mapped to target anomalies and the clusters of infrequent normal instances do not match. LOCIT (panel C), on the other hand, only transfers the subset of source instances for which the *localized* source and target distributions match. Hence, it correctly transfers labeled source instances to the small cluster of infrequent normals, while avoiding incorrect transfer on the lower right.

Location distance: This is the $l2$-norm of the difference between the centers (i.e., arithmetic mean) of two neighborhood sets $N1$ and $N2$ of size ψ:

$$\delta_1(N1, N2) = \left\| \frac{1}{\psi} \left(\sum_{x_i \in N1} x_i - \sum_{x_j \in N2} x_j \right) \right\|_2 . \tag{6.7}$$

Here, LOCIT computes $\delta_1(N_\psi(x_s; D_S), N_\psi(x_s; D_T))$. Intuitively, large values of δ_1 indicate less overlap between the regions covered by the two sets, which correspondingly decreases the chance of meaningful transfer.

Correlation distance: This is the relative distance between the covariance matrices of two neighborhood sets:

$$\delta_2(N1, N2) = \frac{\|\Sigma_{N1} - \Sigma_{N2}\|_F}{\|\Sigma_{N1}\|_F} \tag{6.8}$$

where $\|\cdot\|_F$ is the Frobenius norm and Σ is the covariance matrix. Again, LOCIT computes $\delta_2(N_\psi(x_s; D_S), N_\psi(x_s; D_T))$. If δ_2 is large, this signals that the underlying localized source and target distributions are distinct in shape and/or orientation, which again reduces the chance of meaningful transfer.

The size of the neighborhood set ψ is the only hyperparameter in the transfer step of LocIT. Intuitively, ψ controls the strictness of the instance transfer. If ψ is maximal (the minimum of the number of source or target instances), LocIT ignores local distributional differences and considers the full global structure of the source and target domains to determine transfer. If ψ is 1, LocIT only transfers a source point when the distance to its nearest neighbor in the source and target domain is similar to the average distance between any two neighboring points in the target domain.

Learning the Instance-Transfer Function. The instance transfer function f needs to decide whether to transfer x_s by combining the information provided by δ_1 and δ_2. LocIT learns an Svm classifier on the target distribution to serve as f. The classifier predicts if a source instance x_s belongs to the target domain by looking at the correlation and location distance between the neighborhood sets of the instance in the source and target data.

To train the classifier, LocIT generates training data by only considering the target data. It does so by leveraging the smoothness assumption that neighboring target instances should have similar localized distributions while far-away instances should not. Thus, one positive training instance is generated for each instance $x_t \in D_T$ by finding its *nearest neighbor* $x_n \in D_T \setminus \{x_t\}$ and computing $\delta_1(N_\psi(x_t; D_T), N_\psi(x_n; D_T))$ and $\delta_2(N_\psi(x_t; D_T), N_\psi(x_n; D_T))$. Similarly, the negative training instances are generated by computing for each instance $x_t \in D_T$ a feature vector consisting of the distances between the neighborhood sets of x_t and its *farthest neighbor* $x_f \in D_T \setminus \{x_t\}$.

LocIT tunes the Svm's hyperparameters using three-fold cross-validation on the generated training data. It selects either a linear or Gaussian kernel and sets $\eta_c \in [0.01, 0.1, 0.5, 1, 10, 100]$ for both kernels and $\sigma \in [0.01, 0.1, 0.5, 1, 10, 100]$ for the Gaussian kernel.

Figure 6.1 compares LocIT with a popular global domain alignment strategy, Coral [131], on a small synthetic 2D source and target dataset.

6.1.4 Using the Transferred Instances in the Target Domain

The next step is constructing the anomaly detector in the target domain. After transferring the source instances (using Cbit, iTrade, or LocIT), there are two challenges with making predictions for the target instances. First, the target domain now contains a mix of labeled and unlabeled instances. Second, because this is an anomaly detection problem, the known labels in a target instance's neighborhood are not necessarily informative of its label.

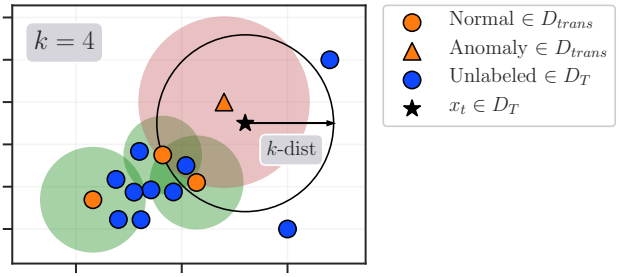


Figure 6.2: After transfer, SSKNNO computes an anomaly score for each target instance as a weighted combination of an *unsupervised* score and a *supervised* score. In this example using $k = 4$, x_t's nearest neighbors are two source normals, a source anomaly, and an unlabeled target instance. Because x_t does not belong to the neighborhood sets of the two source normals (green circles), they are excluded when computing the weight for the supervised component of the score. Thus, the weights of the unsupervised and supervised components are respectively $\frac{3}{4}$ and $\frac{1}{4}$.

The third contribution of this chapter is a semi-supervised anomaly detection algorithm, SSKNNO, that combines the unlabeled and (transferred) labeled instances to compute an anomaly score for each target instance. On the one hand, it considers the local distribution of the unlabeled target instances when computing the score. On the other hand, it weights this score by comparing the neighborhoods of the (transferred) labeled instances and the unlabeled instances in the target data.

Let $D^* = D_T \cup D_{trans}$ be the set of instances that includes both the target instances and the transferred source instances. For a given target instance x_t, we find the set of neighbors $N_k(x_t; D^*)$ and denote its subset of labeled neighbors as $L_k(x_t; D^*)$. The weight w_l of the labeled instances in the neighborhood of x_t is now:

$$w_l(x_t) = \frac{|\{x_i | x_i \in L_k(x_t; D^*) \wedge x_t \in N_k(x_i; D^*)\}|}{k}. \tag{6.9}$$

Intuitively, when assigning an anomaly score to x_t, we only want to consider the label of a transferred source instance if the source instance is similar to x_t. For example, if x_t is an isolated instance, its k-nearest neighbors will be far-away. Even if labeled, such instances will not be very predictive of x_t's label. This is reflected in $w_l(x_t)$, which only considers instances in x_t's neighborhood that also include x_t in their neighborhood.

This weight can now be used to compute the anomaly score $a(x_t)$ for instance x_t as a weighted combination between an unsupervised component a_u, that considers the local data distribution, and a supervised component a_l, that considers nearby labeled instances:

$$a(x_t) = (1 - w_l(x_t))\ a_u(x_t) + w_l(x_t)\ a_l(x_t). \tag{6.10}$$

The supervised component of the score a_l is the distance-weighted average of the labels of the instances in the neighborhood of x_t:

$$a_l(x_t) = \frac{\sum_{x_i \in L_k} \mathbb{1}_{y_i = \texttt{anomaly}}(x_i)\ \omega(x_i; x_t)}{\sum_{x_i \in L_k} \omega(x_i; x_t)} \tag{6.11}$$

where ω is defined as in Equation 6.6 and $\mathbb{1}_{y_i=\texttt{anomaly}}(x_i)$ is 1 if the instance x_i is labeled as anomalous, 0 otherwise. The unsupervised component of the score a_u uses k-dist based on the KNNO algorithm but bounds the distance to $[0, 1]$ using a squashing function from the exponential family:

$$a_u(x_t) = 1 - \exp\left(-\frac{\text{k-dist}(x_t; D^*)^2}{2\gamma^2}\right) \tag{6.12}$$

where γ is the contamination factor, which can be estimated from the source data. This exponential quashing is a monotone function and higher values still represent more anomalous instances. See Figure 6.2 for a graphical explanation of SSKNNO.

In extreme cases the weight $w_l(x_t)$ can be zero or one. If none of x_t's neighbors are labeled, L_k is empty and $w_l(x_t)$ becomes zero, reducing Equation 6.10 to the scaled KNNO score. This can happen if LOCIT's transfer function selects no instances to be transferred. Conversely, if all of x_t's neighbors are labeled and x_t belongs to the neighborhood set of each of its neighbors, the final anomaly score is the standard, weighted KNN classifier.

6.2 Related Work

Instance-Based. In this chapter, we focus on instance transfer where weighted source domain instances are used to construct a decision function in the target domain [97], such as *two-stage weighting framework for multi-source domain adaptation* (2SW-MDA) and *conditional-probability-based multi-source domain adaptation* (CP-MDA) [23]. Unlike our problem setting, however, CP-MDA requires labeled target data to work, while 2SW-MDA does not work when only instances of one class are labeled in the source domain.

Feature-based. These techniques transform both the source and target data into a new, latent feature space that minimizes the distributional differences, and learn a classifier in this new space. To find the latent space, one class of methods, such as *transfer component analysis* (TCA) [101], *transfer joint matching* (TJM) [89], and *geodesic flow kernel* (GFK) [54], corrects only for covariate shifts. A second class of methods, such as *joint distribution adaptation* (JDA) [88] and *joint geometrical and statistical alignment* (JGSA) [155], attempts to correct both a covariate shift and concept shift by computing pseudo-labels for the unlabeled source and target data. Finally, methods such as CORAL [131] align the source and target domains in the original feature space. All these methods use the 1-nearest neighbor classifier (1NN) for the target data, with the transformed source data as the training set.

LOCIT differs from these approaches in three key ways. First, it implicitly corrects for conditional distribution differences between the source and target domains by observing the densities of the data distributions. Second, it uses a semi-supervised nearest-neighbors style classifier in the target domain, which we will show empirically leads to better performance. Third, LOCIT's target domain nearest-neighbors classifier works even if instances from only one class are transferred because it interpolates between a supervised and unsupervised score. In contrast, the other approaches require that labeled instances from all classes are transferred.

Transfer Learning for Anomaly Detection. Only a handful of papers explore the use of transfer learning for anomaly detection, such as [3] that tries to reuse learned image representations across different image datasets, and [148] that designed a robust one-class transfer learning method. These approaches require labeled target instances to construct the classifier. The lack of label information in the anomaly detection task that motivated our approach prohibits using these approaches.

6.3 Benchmark Experiments

The empirical evaluation of LOCIT focuses on five questions:

Q1: How does LOCIT perform compared to state-of-the-art transfer learning and anomaly detection algorithms?

Q2: How does the percentage of labeled source instances affect the transfer learning algorithms' performance?

Q3: How does the transfer difficulty affect the performance of the transfer learning algorithms?

Q4: How does our nearest-neighbor's approach for classifying target instances affect performance?

Q5: How do the values of hyperparameters ψ and k affect the performance of LocIT?

The code, benchmark data, elaborated explanations, full parameter settings, and further experiments are available in an online appendix: `https://github.com/Vincent-Vercruyssen/LocIT`

6.3.1 Experimental Setup

Compared Approaches. We experimentally compare LocIT with 11 state-of-the-art approaches, divided into two categories:

Baseline anomaly detection algorithms. Four standard unsupervised anomaly detection techniques that only consider the target domain data: Knno, Lof, iForest, and Hbos (a histogram-based anomaly detection technique [52]).

Baseline transfer learning approaches. Eight transfer learning approaches: TransferAll (a naive baseline that transfers all source instances as is to the target domain), Coral [131], Tca [101], Gfk [54], Jda [88], Tjm [89], Jgsa [155], and Cbit [143]. After transfer, these methods use a Knn classifier to classify the target instances.

The iTrade algorithm was not separately published so it is not included in the experimental study. It also evolved into the LocIT algorithm. Cbit is included because it was originally published in [143] before the iTrade and LocIT algorithms were developed.

Benchmark Construction. We construct our own benchmark because the current transfer learning benchmarks (e.g., [88, 131, 101]) do not contain anomalies, while the current anomaly detection benchmarks (e.g., [15, 53]) do not contain source-target pairs for each problem. An appropriate benchmark for transfer learning for anomaly detection should exhibit three characteristics. First, it should contain a sufficient number of datasets, each dataset consisting of a source-target domain pair. The source and target domains contain instances

Dataset	# Features	# Instances	# Classes
Abalone	8	4177	$3^{(1)}$
Covertypes	54	581012	7
Gas sensor array drift	128	13910	6
Gesture phase segmentation	32	9900	5
Recognition of handwritten digits	64	5620	9
Statlog landsat satellite	36	6435	6
Letter recognition	16	20000	26
Pen-based recognition of digits	16	10992	10
Image segmentation	19	2310	7
Shuttle	9	58000	7
Waveform	21	5000	3
Poker	11	1025010	$7^{(2)}$

Table 6.1: Characteristics of the 12 master datasets used to construct the transfer learning for anomaly detection benchmark. The datasets are obtained from the UCI Machine Learning repository [39]. For each dataset, only the numeric features are used. Age$^{(1)}$ is divided into three classes. The three$^{(2)}$ smallest classes are removed.

that are sampled from the same d-dimensional input space. Second, each source and target domain should contain normal and anomalous instances, a number of which are nontrivial to classify. The importance of this condition in general anomaly detection benchmarks is stated in [42]. Third, the benchmark datasets should portray varying degrees of differences between the joint data distributions of the source and target domain.

Our benchmark is constructed to reflect these three characteristics. We start with 12 publicly available, multi-class *master datasets* that are listed in Table 6.1. Each master dataset is converted into three (or five) unique source-target domain pairs, depending on whether the master dataset contains three (or more) classes. Because each source-target domain pair is constructed from the same master dataset, the source and target instances live in the same d-dimensional input space. This results in a total of 56 unique benchmark datasets.

For any source-target domain pair, the target domain is constructed by sampling the normal (anomalous) instances from the largest (second largest) class in the master dataset. We ensure that the target domain contains instances that are nontrivial to classify by varying the *point difficulty* of the sampled instances, as outlined in [42]. The point difficulty simply quantifies how difficult it is to separate the instances of the two classes in a binary classification problem with full access to the label information. Meaningful source normals and anomalies are sampled from the chosen classes using the same procedure as in the target domain. By varying the master dataset classes from which

the source normals and anomalies are sampled, we can introduce meaningful distributional differences between the source and target domain. If the source normals (anomalies) are sampled from a different class than the target normals (anomalies), the marginal distributions of both domains will differ. If the source anomalies (normals) and the target normals (anomalies) are drawn from the same class, conditional distribution differences arise. Both actions decrease the similarity between the source and target domains and make transfer more difficult. Finally, each source or target domain has 500 normal instances and 50 anomalies. Because the sampling is non-deterministic, we repeat it 10 times for source-target domain pair, and report the averaged results.

Experimental Setup. We employ the standard transfer learning experimental setup used in previous work [88, 155]. For each of the 56 source-target pairs in the benchmark, the following two steps are performed. First, each method transforms the source data and/or selects the source instances to transfer to the target domain. Second, a final classifier is learned using the transferred, labeled source data (and the unlabeled target data) and a prediction is made for each target instance. The anomaly detection algorithms are simply run on the target data (i.e., they do not use any source data). The final classifier's performance is evaluated using the AUROC, as is standard in anomaly detection [42].

Hyperparameters. Because there are no labels in the target domain and the distribution of the source data is different, hyperparameter tuning using cross-validation is impossible [88, 54, 101]. Hence, we resort to setting the hyperparameters of the baselines in accordance with the recommendations made in the original papers or subsequent comparison studies. TransferAll and Coral have no parameters to set. For Tca, Jda, Gfk, Tjm, Cbit, and Jgsa, we use the parameter settings from the original papers. After transfer, the above methods use the Knn classifier with $k = 10$ to make predictions in the target domain. iForest is trained with 100 members in the ensemble. Knno and Lof both have a single parameter k, the number of neighbors. We set $k = 10$ for Knno based a recent study that tested Knno on approximately 1000 datasets with different values for k [15]. Similarly, we set $k = 10$ for Lof. For Hbos, we set the number of bins to 10. LocIT has two hyperparameters. We set the neighborhood size $\psi = 20$ and $k = 10$ in SSknno. **Q5** analyzes the impact of ψ and k on LocIT's performance.

Transfer method	Final classifier	Average AUROC rank ± SD of each method	Average AUROC ± SD of each method	# times LocIT			Average % change in AUROC using LocIT over all datasets
				wins	draws	loses	
LocIT	SSknno	**3.741 ± 3.318**	**0.762 ± 0.182**	-	-	-	-
-	Knno	5.321 ± 3.663	0.705 ± 0.177	**44**	4	8	+9.63% ± 18.31%
Coral	Knn	5.848 ± 3.311	0.666 ± 0.188	**36**	1	19	+20.65% ± 40.60%
-	Lof	6.018 ± 4.413	0.677 ± 0.113	**37**	0	19	+14.53% ± 31.53%
TransferAll	Knn	6.732 ± 3.646	0.649 ± 0.185	**37**	0	19	+27.34% ± 62.98%
-	iForest	6.795 ± 3.898	0.690 ± 0.169	**50**	1	5	+11.32% ± 13.53%
Gfk	Knn	7.125 ± 3.312	0.642 ± 0.186	**38**	1	17	+29.10% ± 67.03%
Tca	Knn	7.348 ± 2.356	0.656 ± 0.154	**44**	1	11	+18.20% ± 27.64%
-	Hbos	7.920 ± 4.015	0.646 ± 0.216	**48**	2	6	+29.16% ± 53.65%
Tjm	Knn	8.000 ± 2.793	0.627 ± 0.175	**44**	2	10	+27.22% ± 39.12%
Cbit	Knn	8.223 ± 2.345	0.623 ± 0.150	**46**	0	10	+24.57% ± 28.54%
Jda	Knn	8.509 ± 2.910	0.617 ± 0.170	**44**	0	12	+30.51% ± 48.76%
Jgsa	Knn	9.420 ± 3.845	0.576 ± 0.096	**46**	0	10	+33.99% ± 33.27%

Table 6.2: Comparison of LocIT to the state-of-the-art baselines when the source domains are fully labeled. The table shows over the full benchmark: the average AUROC rank ± standard deviation (SD) of each method; the average AUROC ± SD of each method; the number of times LocIT wins (higher AUROC), draws (absolute difference in AUROC < 0.005), and loses (lower AUROC) vs. each method; and the average percentage change in AUROC ± SD of using LocIT over each method.

6.3.2 Results Q1: LocIT versus State-of-the-art

Table 6.2 compares LocIT to all baselines when the source domain is fully labeled. LocIT outperforms all baselines on the full benchmark, having the lowest average AUROC rank (lower rank is better) and achieving the highest average AUROC. Furthermore, LocIT yields an average increase in AUROC of at least $> 9\%$ compared to performing unsupervised anomaly detection. This provides evidence that LocIT's approach to transfer can help in anomaly detection tasks. Finally, LocIT achieves average AUROC gains of $>18\%$ over all the transfer learning baselines, indicating that it substantially outperforms the state-of-the-art transfer approaches.

6.3.3 Results Q2: Effect of Varying the Percentage of Source Labels

To explore how the amount of labeled source data affects performance in the target domain, we vary the proportion of labeled sources instances. This is relevant because often it is difficult to obtain fully labeled data for anomaly detection problems. We randomly sampled $10\% - 100\%$ with increments of 10% of the source instances and only considered the labels of these instances when performing transfer. We repeated this procedure five times and averaged results.

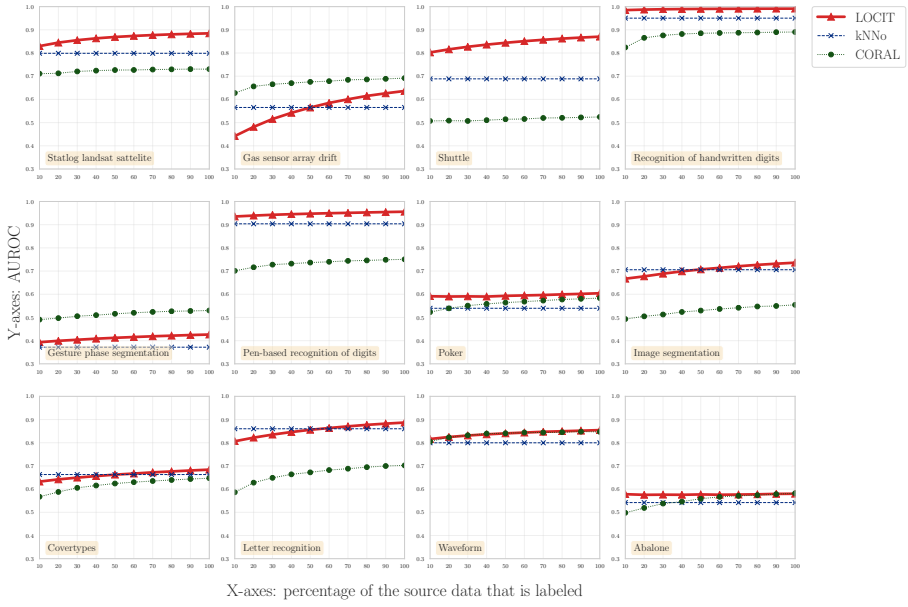


Figure 6.3: The AUROC averaged over all source-target pairs for each master dataset, as a function of the percentage of labeled source data for LocIT, Knno, and Coral. LocIT's and Coral's performances improve as labels are added to the source domain. With 50% of the source labels, LocIT always performs better or equivalently to Knno.

Figure 6.3 shows how the average AUROC varies as a function of the proportion of labeled source instances. For readability, the plots show the results for LocIT as well as Coral and Knno, which are the best transfer and anomaly detection baselines respectively as determined by Table 6.2. Knno's curves are straight lines because it only considers the target data. Coral improves as more source labels become available. Regardless of the proportion of source labels, LocIT outperforms Coral on nine of the 12 master datasets. Furthermore, with only 10% of the source labels, LocIT performs better than or similar to Knno on eight of the 12 master datasets. As more source labels become available, LocIT's performance improves and with 50% of the source labels, LocIT always performs better or equivalently to Knno. As source labels continue to be added, the performance gap with Knno widens.

Transfer method	Final classifier	Average AUROC rank ± SD of each method	Average AUROC ± SD of each method	# times LocIT			Average % change in AUROC using LocIT over all datasets
				wins	draws	loses	
LocIT	SSknno	**2.071 ± 1.431**	**0.762 ± 0.182**	-	-	-	-
Coral	SSknno	2.741 ± 0.973	0.735 ± 0.171	**37**	5	14	+3.88% ± 9.54%
TransferAll	SSknno	3.420 ± 1.047	0.727 ± 0.173	**41**	4	11	+5.17% ± 10.29%
Gfk	SSknno	3.848 ± 1.526	0.705 ± 0.172	**44**	5	7	+8.95% ± 13.28%
Cbit	SSknno	4.214 ± 1.021	0.713 ± 0.171	**50**	1	5	+7.28% ± 10.64%
Tca	SSknno	4.705 ± 2.063	0.533 ± 0.177	**42**	0	14	+74.81% ± 116.18%

Table 6.3: Comparison of LocIT to the best transfer baselines of Table 6.2 coupled with LocIT's nearest neighbor approach for making predictions in the target domain. The source domains are fully labeled. The table shows over the full benchmark: the average AUROC rank ± SD of each method; the average AUROC ± SD of each method; the number of times LocIT wins (higher AUROC), draws (absolute difference in AUROC < 0.005), and loses (lower AUROC) vs. each method; and the average percentage change in AUROC ± SD of using LocIT over each method.

6.3.4 Results Q3: Impact of Transfer Difficulty on LocIT

Figure 6.4 shows the average AUROC values for LocIT, Coral (the best transfer method from Table 6.2) and Knno (the best anomaly detection method from Table 6.2) as a function of the transfer difficulty for the 12 master datasets. The transfer difficulty is reflective of the similarity between the source and target domains: the more similar the domains, the lower the transfer difficulty. Because Knno does not use source instances, the difficulty of the transfer task does not affect its performance. However, it is useful to show its performance to assess under what condition transfer is helpful. For each dataset, LocIT's AUROC is higher than that of Knno when the difficulty of the transfer task is low. This illustrates an important empirical insight: the gains from transfer increase when the source and target domain are more similar. As the domains become less similar, the performance of the two transfer methods degrades. Still, on nine of the 12 datasets LocIT performs similarly or better than Knno on the most difficult transfer setting. Additionally, on nine of the 12 master datasets, LocIT consistently achieves higher AUROC values than Coral when the transfer difficulty is high. LocIT's localized instance-selection step helps deal with dissimilarities between the source and target domains, sometimes at the cost of slightly worse performance when domains are similar.

6.3.5 Results Q4: Impact of the SSknno Approach

To assess how much of LocIT's gains come from using our novel SSknno approach, we use SSknno instead of Knn as the target domain classifier for the best competing transfer methods from Table 6.2. Table 6.3 shows the results for

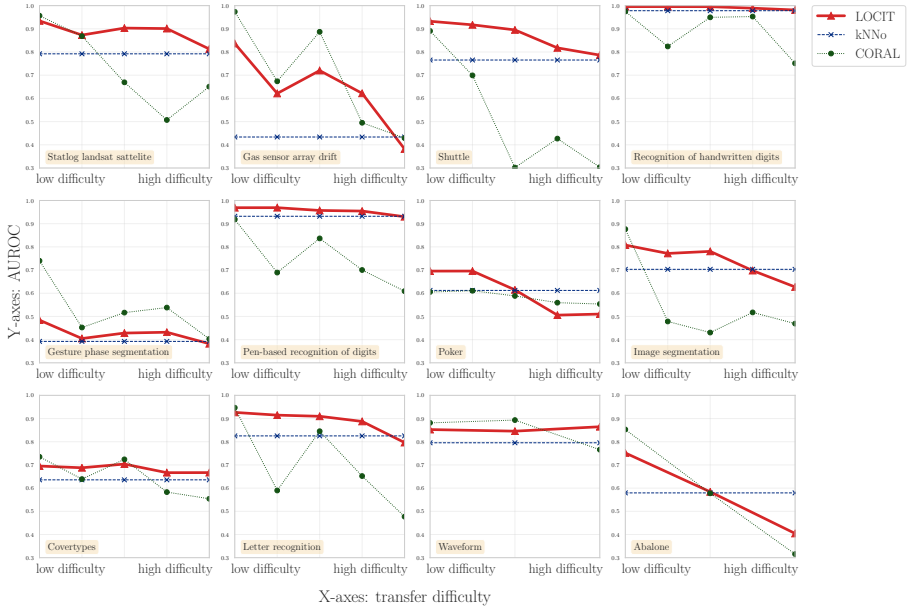


Figure 6.4: The average AUROC for each master dataset as a function of the difficulty of the transfer task. The plot shows results for LocIT (red line) as well as Knno (blue line), and Coral (green line) which are the best anomaly detection and transfer learning baselines from Table 6.2. The plots show that when the source and target domains are more similar (lower transfer difficulty), transfer yields larger gains. As the source and target domains become more dissimilar, the usefulness of transfer decreases.

this experiment. For Coral, Gfk, TransferAll, and Cbit, using SSknno as the classification approach results in improved performance versus the Knn classifier. The exception is Tca. Even when they are coupled with the SSknno classifier, LocIT outperforms all the baselines. It performs better than or similar to (difference in AUROC < 0.005) all competitors on at least 37 out of 56 benchmark datasets, and achieves a higher average AUROC than all other methods. This indicates that both LocIT's novel instance selection procedure for transfer and its approach to classification in the target domain contribute to its superior performance.

6.3.6 Results Q5: Impact of LocIT's Hyperparameters.

LocIT has two hyperparameters. First, the hyperparameter ψ controls the strictness of the instance-selection step. Choosing $\psi \geq 20$ generally yields good performance. Lower values degrade performance. Second, hyperparameter k in the SSknno step of LocIT controls the number of neighbors considered when deriving an anomaly score for an instance. A *good* value of k depends on the type of dataset. Lower values of $k < 20$ are an overall good choice across a range of datasets.

6.4 Real-world Experiments

We evaluate LocIT's effectiveness on a real-world transfer task of detecting anomalous water usage in a chain of retail stores (see Chapter 5). The retail company operates hundreds of stores and wants to avoid excess usage due its harmful environment impact and to minimize costs. Detecting anomalous usage is challenging because the data contains infrequent, but *normal* irregularities such as maintenance patterns, after-hour events, and temporary alterations in opening hours. Similarly, certain *abnormal* behaviors such as leaks occur relatively frequently. Hence, the data violate the standard assumptions made in unsupervised anomaly detectors.

Having access to some labeled instances could help improve detection performance and correct the mistakes made by unsupervised detectors. Unfortunately, it is not feasible to label even a small subset of the data for every store due to the time costs associated with labeling. This raises the following the question: "can labeled instances be transferred between different stores to improve anomaly detection performance?" Transfer in this setting is not straightforward because the observed usage, and consequently what constitutes (ab)normal behavior, varies substantially due to contextual differences among stores (e.g., location, size, clientele, opening hours, services offered, etc.).

Data and Methodology. We have three full years of historical water usage time series data for three stores. The data consists of a univariate measurement that is recorded every five minutes. In each store, company experts labeled about *ten percent* of the data, with each of the provided labels indicating whether a block of one hour (e.g., 01:00-02:00) shows normal behavior or not. The rest of the data is unlabeled.

In each store, the time series data is first divided into non-overlapping one-hour segments (i.e., 00:00-01:00, 01:00-02:00, etc.). Then, each segment is transformed

source	target	KNNO	TRANSFERALL	CORAL	LOCIT
store 1	store 2	0.779 ± 0.126	0.611 ± 0.154	0.646 ± 0.133	$\mathbf{0.800 \pm 0.112}$
store 1	store 3	0.943 ± 0.063	0.758 ± 0.192	0.776 ± 0.182	$\mathbf{0.969 \pm 0.049}$
store 2	store 1	0.754 ± 0.227	0.771 ± 0.207	0.771 ± 0.219	$\mathbf{0.779 \pm 0.227}$
store 2	store 3	0.943 ± 0.063	0.790 ± 0.232	0.794 ± 0.232	$\mathbf{0.969 \pm 0.050}$
store 3	store 1	0.754 ± 0.227	0.704 ± 0.159	0.702 ± 0.159	$\mathbf{0.779 \pm 0.227}$
store 3	store 2	0.779 ± 0.126	0.592 ± 0.164	0.638 ± 0.135	$\mathbf{0.800 \pm 0.112}$

Table 6.4: The average AUROC ± SD on the real-world water usage data by LOCIT, the best non-transfer baseline (KNNO) and the two best transfer baselines (CORAL and TRANSFERALL) from the benchmark. Best results are in bold. LOCIT consistently outperforms its competitors.

into a length 31 feature vector describing the signal's characteristics during that segment [145]. Because of a store's opening hours, the time of day has a significant effect on water usage. Therefore, all segments for the same hour interval (e.g., 11:00-12:00) are grouped, resulting in 24 groups per store. A separate anomaly detector is trained for each group.

In the transfer learning experiment, each segment group for one store is treated as the *unlabeled* target domain, while each of the 24 segment groups from a different store serves as the *partially labeled* source domain (about ten percent of the instances is labeled). Using three stores, we construct six unique source-target store combinations, each of which has 576 source-target pairs.

Experimental Results and Discussion. Table 6.4 reports the AUROC values of this experiment, averaged per source-target store combination. In aggregate, LOCIT is always better than both KNNO and CORAL on all six store-store combinations. On all 3456 transfer tasks (576 tasks per store-store pair × six pairs), LOCIT wins (difference in AUROC > 0.01) 2427 and ties (difference in AUROC < 0.01) 280 times with CORAL. It wins 2512 and ties 272 times versus TRANSFERALL. Finally, it wins 1868 and ties 1441 times versus KNNO. LOCIT outperforms or ties with KNNO on 95% of the real-world transfer tasks, which provides evidence of the effectiveness of label transfer to improve anomaly detection in a real-world setting.

6.5 Conclusion

In this chapter, we tackled the problem of transfer learning for anomaly detection. The user provides flexible supervision in the form of binary labels for a dataset related to the dataset for which we are constructing a model. She does not

provide any feedback as to which information in the related dataset might be useful to transfer, nor how to exploit this information in the target model. We designed three algorithms for tackling this task: CBIT, ITRADE, and LOCIT. The LOCIT algorithm builds upon the insights gleaned from CBIT and ITRADE. It works in two steps. First, it learns which source instances might be useful to transfer to the target domain based on the similarity of the localized source and target distributions around each instance. Then, it derives an anomaly score for each target instance using the unlabeled, target instances and the transferred, labeled source instances. We also designed the SSKNNO algorithm, a semi-supervised anomaly detection algorithm that interpolates between the unsupervised KNNO and supervised KNN algorithm.

Chapter 7

Conclusion

This chapter summarizes the main challenges and contributions encountered in this dissertation, as well as some lesson learned for the field of anomaly detection. Finally, it provides an outlook on some of the open, interesting research questions in the field of anomaly detection.

7.1 Summary

The central hypothesis of this dissertation is that *anomaly detection algorithms would benefit from exploiting flexible forms of supervision.* Flexible supervision refers to all possible forms of domain knowledge an expert can provide to the anomaly detection model, going substantially beyond assigning a simple binary labels. The forms of flexible supervision studied in this dissertation are: knowledge of sporadically occurring patterns, event logs accompanying time series data, binary labels, and the availability of related datasets.

The current literature on anomaly detection does not explore designing algorithms that exploit flexible forms of supervision. In this dissertation, we zoomed in on three concrete gaps in the literature. First, there is no algorithm that can identify the suspicious absence of a sporadically occurring pattern in a time series dataset. The user provides flexible supervision in the form of several annotated occurrences of the pattern. She does not provide information as to what is normal or anomalous or which context is relevant to explain the pattern's occurrence. She simply knows the pattern is relevant.

Second, there is no algorithm that can combine the information contained

in both a continuous time series dataset and its accompanying event log to construct an anomaly detection model. The user provides flexible supervision in the form of an event log which contains discrete events that could explain the patterns observed in the continuous time series data. She does not provide details on which events are normal or anomalous or how they are connected to the patterns in the continuous series.

Third, there is a lack of algorithms that can augment purely data-driven anomaly detectors with binary label information either obtained directly from the expert or derived from a related, yet different, source dataset. The user provides flexible supervision in the form of binary labels for the target dataset itself or for the source dataset. However, she does not indicate how the source dataset could be relevant to solving the target anomaly detection problem nor how the source label information should be exploited.

7.1.1 Exploiting Reoccurring Pattern Information

FZapPa

Some anomalies are characterized by the *absence* of behavior not its *presence*. Non-i.i.d. data, such as time series data, can contain sporadically occurring patterns. For example, a maintenance operation executed on the system that we are monitoring. This results in its characteristic fingerprint being observed in the data. In this case, the absence of this pattern that is expected to occur, is indicative of an anomaly. There is no algorithm to detect absent pattern anomalies in time series as the existing algorithms focus on detecting the presence of anomalous behavior.

We developed FZapPa, an algorithm that works in three steps. First, it learns to identify all occurrences of a specific pattern in the time series data based on a limited amount of user-annotated occurrences. Second, it learns how this pattern is distributed in the time series and automatically identifies the relevant context underlying its distribution. Finally, it identifies suspicious absences of the pattern based on the distribution learned in step two.

Detecting all occurrences of a sporadically occurring pattern in a time series is not trivial. We showed that it is possible to treat this problem as a PU learning problem. The user first provides flexible supervision in the form of a limited set of annotated occurrences of the pattern (the positive labels). Then, the classifier identifies the remaining occurrences in the rest of the time series. The classifier cannot detect suspicious absences based on the set of annotated occurrences alone.

Modeling the occurrence sequence of a sporadically occurring pattern requires the right context descriptors. How the occurrences of a pattern are distributed across a time series dataset and how subsequent occurrences are linked cannot always be explained by just looking at the temporal context. Some applications require other descriptors to explain why a pattern occurred again at a given moment. These descriptors can be derived from the time series data itself, such as cumulative usage. FZapPa is able to learn which context descriptor best lends itself to modeling the pattern occurrences.

A Python implementation of the FZapPa algorithm is available at: `https://github.com/Vincent-Vercruyssen/absent_pattern_detection`

7.1.2 Exploiting Event Log Information

Pbad

Real-world time series data are often accompanied by discrete event logs containing information relevant to detecting anomalies. The event log is supplied by the user as a form of flexible supervision and the information contained in this log can help identify normal and anomalous patterns in the data. We denote such data as *mixed-type time series*. The current research on time series anomaly detection limits itself to either univariate or multivariate, continuous time series data or to discrete event logs, but not the combination thereof.

We developed Pbad, a time series anomaly detection algorithm that works in three steps. First, it leverages frequent pattern mining algorithms to mine frequent patterns from the mixed-type time series. Second, it computes a soft distance between each time series segment and the frequent patterns, resulting in a feature vector representation. We designed an efficient dynamic programming algorithm to compute these distances. Finally, it employs a standard anomaly detection algorithm in the embedding space to detect anomalies.

Frequent pattern mining provides principled methods to extract itemsets and sequential patterns from different types of time series. These patterns capture frequent behavior in the data. By expressing each segment in terms of distances to the frequent patterns, the resulting representation naturally lends itself to anomaly detection. Anomalies are time series segments that do not contain frequent patterns and are characterized by large distances to them.

A Python implementation of the FZapPa algorithm is available at: `https://bitbucket.org/len_feremans/pbad/src/master/`

7.1.3 Exploiting Binary Label Information

Ssdo

Unsupervised anomaly detection algorithms necessarily make assumptions about anomalous behavior, such as anomalies are far away from other instances or lie in low-density regions of the input space. However, real-world data often violate these assumptions. Maintenance operations, for example, can occur less frequently than some anomalous behaviors. On the other hand, some anomalous behaviors (e.g., water leaks) show great regularity. This results in misclassifications by the detector. Even a limited amount of expert feedback in the form of binary labels can correct such mistakes. There is a lack of algorithms that can effectively exploit label information to augment the data-driven anomaly detectors.

We developed Ssdo, an algorithm works in two steps. First, it derives an unsupervised anomaly score for each instance in the data. We designed a constrained-cluster-based algorithm for this purpose which, unlike the Cblof algorithm [61], is a truly local anomaly detector. It can also handle label information during the clustering phase. The use of this algorithm as the first step of Ssdo is optional. One could use any existing unsupervised detection algorithm. Second, it propagates the known labels obtained from the expert throughout the dataset, updating each instance's unsupervised score based on how well it matches a labeled normal or anomaly. The labels are obtained from the expert with the help of an active learning strategy.

A Python implementation of the Ssdo algorithm is available at: `https://github.com/Vincent-Vercruyssen/anomatools`

LocIT

For some applications, we have multiple, related datasets at our disposal. For example, a retailer collects water usage data for all of its stores and wants to build an anomaly detection model for each store. Although each model would greatly benefit from some expert feedback, providing even a limited amount of labels for each store quickly becomes infeasible as the number of stores grows. Nonetheless, it would be possible to provide labels for a subset of the stores. The current semi-supervised anomaly detection algorithms can only exploit the label information contained in one dataset, but not in multiple datasets simultaneously. The label information of a source dataset serves as a form of flexible supervision to improve the anomaly detection model of the target

dataset. However, how should this information be exploited? We propose using transfer learning to address this issue.

We developed LocIT, an transfer learning for anomaly detection algorithm. Given two datasets, a (partially) labeled source dataset and an unlabeled target dataset, LocIT works in two steps. First, it figures out which labeled instances of the source dataset could be useful to improve an anomaly detector in the target domain. It does so by comparing the local instance space around a source instance in both domains. If they are similar enough, the instance is transferred. Second, it uses both the transferred, labeled source instances and the unlabeled target instances to construct a model for anomaly detection in the target domain.

Transfer learning for anomaly detection is challenging due to the nature of the problem. Because anomalies are not the result of a data generating process, there will always be some form of covariate shift and prior shift, albeit minimal, between the source and target datasets. Concept shift is very difficult to correct for, especially given the limited label availability. The only solution is to introduce explicit assumptions upon which the transfer process can be modeled. The possibility of negative transfer is real and transfer anomaly detection algorithms should explicitly try to avoid this. The LocIT algorithm circumvents this issue by assuming that source and target instances have the same meaning if the data distributions localized around the instance in the source and target domain are similar.

A Python implementation of the LocIT algorithm is available at: `https://github.com/Vincent-Vercruyssen/transfertools`

SSknno

Due to the nature of the transfer learning for anomaly detection problem, the transferred source instances might be few, or only part of either the normal distribution or the anomaly distribution, or represent a biased subsample of the target domain. Therefore, the final target anomaly detection model should consists both of transferred source instances and unlabeled target instances. This requires the use of a semi-supervised anomaly detection algorithm.

We developed SSknno, a semi-supervised anomaly detection algorithm that computes an anomaly score in two steps. First, it computes for each instance its unsupervised Knno anomaly score, and a supervised score based on the distance-weighted average of the labels of the instances in that instance's neighborhood. Then, it computes a final score as weighted combination between

the unsupervised and supervised score. SSKNNO has KNNO and KNN as edge cases when respectively none or all instances in the dataset are labeled.

A Python implementation of the SSKNNO algorithm is available at: `https://github.com/Vincent-Vercruyssen/anomatools`

7.2 Challenges and Takeaways

In this dissertation, we designed anomaly detection algorithms that exploit flexible forms of supervision and applied them to a number of real-world use cases. Below, we summarize some of the challenges we encountered and the lesson we learned from them.

Simple is better than complex. Simple models and techniques often outperform the more complex techniques. In many real-world problems, such as detecting anomalous water consumption, simple and intuitive techniques already get you quite far (Chapter 5). Complex models with lots of hyperparameters to tune or choose can be unwieldy. They are difficult to evaluate due to *label scarcity* (see supra) or consist of many parts whose interactions are difficult to predict. State your assumptions and use a simple model that reflects them.

Construct useful features. It is better to spend time constructing informed features than trying out complex models. Well-chosen features are able to describe the important patterns present in the data, allowing the anomaly detection models to identify them (Chapters 5 and 3). Another strategy is to map the data to a new representation that is better suited for anomaly detection (Chapter 4). If the dataset is large (e.g., multivariate time series data), manually constructing features that capture all the interdependencies is challenging. Techniques for automated feature construction could be helpful [122, 33].

Labels are hard to get. In real-world anomaly detection applications, labels are hard to get. We refer to this as *label scarcity*. This has several explanations. First, anomalies are rare so obtaining labels for them entails sifting through a *lot* of data. Second, it is not always feasible to let, for example, a machine break just to generate an example of anomalous behavior. Finally, anomalies are not well-defined, so we do not always know beforehand what exactly we are looking for. The label scarcity has several important implications. First, it is paramount to exploit the information contained in the few labels one might have, as much as

possible. Second, comparing the performance of different algorithms is not always feasible.

Domain knowledge is key. Expert domain knowledge about an applications comes in many forms. Without exploiting this information explicitly, it is often not possible (or at least extremely cumbersome) to solve the task at hand, such as absent pattern detection (Chapter 3). There are already many unsupervised anomaly detection algorithms and surveys that compare them [53, 15, 38, 41, 130]. We should not add new unsupervised detectors to list, but rather find innovative ways to extend the existing algorithms such that they can exploit domain knowledge or handle more complex data representations.

Construct your own benchmark. For many real-world problems, there exist no scientific benchmark datasets that allow the cross-comparison of different algorithms. It is sometimes necessary to construct your own benchmark. Additionally, many of the existing benchmarks are constructed in a synthetic way that does not represent reality. A common practice is to start with a classification dataset and downsample one of the classes to represent the anomalies [15]. However, anomalies are supposed to not follow a specific distribution. These interpretations are clearly conflicting.

7.3 Future Research Directions

Normalizing, Unifying, and Interpreting Anomaly Scores. Different unsupervised anomaly detectors produce different anomaly scores, the only commonality being that higher scores usually indicate a higher degree of anomalousness. How should the user interpret different scores, let alone draw parallels or find distinctions between the scores of different algorithms? There is a need for a principled approach for mapping the anomaly scores to anomaly probabilities. Anomaly probabilities are easier to interpret for the user. Furthermore, they would serve as an important stepping stone to unifying the output of different algorithms which is in itself important for building ensembles.

Explaining Anomaly Predictions. Explainability has received a lot of attention in machine learning research recently. However, most of the focus in anomaly detection research still lies on devising new detectors. It would be worthwhile to develop methods that can explain the predictions of those detectors. For the expert, understanding *why* something is an anomaly - or at least why the model thinks it is an anomaly - is an important step toward preventing future occurrences.

Expressing Uncertainty about Predictions. Another research avenue that has remained largely untouched, is endowing anomaly detectors with the capability of expressing uncertainty about their predictions. It is only natural that certain aspects of real-world data, such as noise or missing values, result in detectors that achieve different levels of performance in different regions of the input space. Some traditional classification methods are capable of providing a degree of certainty with their prediction. One can easily conceive an anomaly detector that indicates "*I don't know*" as a label rather than `normal` or `anomaly`. Such methods would open the door for active learning strategies that focus on minimizing model uncertainty.

Robustness of Anomaly Detection Algorithms. Just like in any other machine learning problem, the datasets collected for anomaly detection applications can be noisy. In fact, the very nature of anomalies, implies that if you were to collect the same dataset multiple times, you would each time end up with a (slightly) different version. It stands to reason that a model trained on a different versions of the same dataset will yield different predictions. This brings about two important questions. First, how can we quantify the resilience or robustness of anomaly detectors against such variations in the data? Second, how can we design algorithms that are robust against these variations.

The same questions can be extended to, for example, spurious features or noise in the provided label information. What if the expert makes a mistake during labeling? Can we quantify the impact this has on the existing semi-supervised anomaly detectors and if so, how to mitigate the effect of noisy labels or spurious features?

Towards Truly Flexible Supervision. This dissertation explored a number of ways in which an expert can supervise anomaly detection algorithms. This is only the tip of the iceberg, however. There are undoubtedly various other ways in which an expert can provide feedback. There are two important questions: (1) which types of supervision can the expert provide given her background knowledge, and (2) which types would be most effective at improving the anomaly detection models? Consider, for example, the use of constraints or rules to express expert knowledge in the form of "anomalous water consumption always has an irregular pattern." These types of feedback should be formalized and algorithms should be designed that can fuse them with existing anomaly detectors. More concretely, multiple-instance learning techniques could be relevant to time series anomaly detection where the expert labels segments rather than individual measurements.

Choosing a Model and its Hyperparameters. A particularly tricky question, especially in the case of unsupervised anomaly detection, is: "which detector and hyperparameter settings should I use for my dataset?" The lack of labels prevents the use of cross-validation or tuning on a separate validation set. Nonetheless, this question comes up time and again. One solution would be to develop internal evaluation measures that do not require labels [93]. Another solution is to take an AutoML-inspired approach; by thoroughly exploring how data set characteristics such as size, contamination factor, or number of features, impact the performance of various anomaly detectors instantiated with different hyperparameter settings, we could map out a concrete guide to which model is useful when. Such an effort will require an extensive, agreed-upon benchmark.

Generating Effective Benchmark Datasets. The current benchmark datasets suffer from three issues. First, there is no agreement in the Literature on which datasets should be included. The result is that different surveys use different datasets, compare different methods, and even use different evaluation metrics [15, 41, 38]. Second, most benchmark datasets are constructed by taking a (binary) classification dataset and downsampling one of the classes to obtain the anomalies. This clearly violates the idea that anomalies have no data generating mechanism. This makes it debatable how relevant such benchmarks actually are? Third, most benchmarks are too small [41]. This prevents us from encoding the characteristics underlying the benchmark datasets as independent variables that impact the performance of an algorithm (the dependent variable). Subsequently, it is not possible to statistically connect the two, e.g., how does the performance of my algorithm change if I double the amount of spurious features?

The issue is even worse for time series anomaly detection. There is a lack of publicly-available time series datasets that contain labeled anomalous behavior.

Improving Efficiency and Scalability. Finally, this dissertation has mainly focused on designing algorithms that are able to exploit flexible supervision, and less on making these algorithms efficient. Improving the efficiency of anomaly detection algorithms is an important line of research [124]. The important questions are: (1) which algorithms should be made more efficient, and (2) which approximation techniques or other algorithmic means can be used to do so?

Bibliography

[1] Abdallah, A., Maarof, M. A., and Zainal, A. Fraud detection system: A survey. *Journal of Network and Computer Applications 68* (2016), 90–113.

[2] Ahmad, S., Lavin, A., Purdy, S., and Agha, Z. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing 262* (2017), 134–147.

[3] Andrews, J. T., Tanay, T., Morton, E. J., and Griffin, L. D. Transfer representation-learning for anomaly detection. In *33rd International Conference on Machine Learning* (2016).

[4] Angiulli, F., and Pizzuti, C. Fast outlier detection in high dimensional spaces. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery* (2002), Springer-Verlag, pp. 15–26.

[5] Bandaragoda, T. R., Ting, K. M., Albrecht, D., Liu, F. T., Zhu, Y., and Wells, J. R. Isolation-based anomaly detection using nearest-neighbor ensembles. *Computational Intelligence 34*, 4 (2018), 968–998.

[6] Barnett, V., and Lewis, T. *Outliers in statistical data*. John Wiley & Sons, 1994.

[7] Baydogan, M. G., Runger, G., and Tuv, E. A bag-of-features framework to classify time series. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 11 (2013), 2796–2802.

[8] Bekker, J., and Davis, J. Learning from positive and unlabeled data: A survey. *Machine Learning 109*, 4 (2020), 719–760.

[9] Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. A theory of learning from different domains. *Machine learning 79*, 1-2 (2010), 151–175.

[10] Berndt, D. J., and Clifford, J. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (1994), pp. 359–370.

[11] Bhatia, A., Gill, U., and Mol, R. Ready or not, AI is coming to IT operations. Boston Consulting Group.

[12] Box, G. E., Jenkins, G. M., and Reinsel, G. C. *Time series analysis: Forecasting and control*. John Wiley & Sons, 2011.

[13] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. LOF: Identifying density-based local outliers. In *Proceedings of the ACM SIGMOD international conference on Management of data* (2000), vol. 29, pp. 93–104.

[14] Bu, Y., Leung, T.-W., Fu, A. W.-C., Keogh, E., Pei, J., and Meshkin, S. WAT: Finding top-k discords in time series database. In *Proceedings of the SIAM International Conference on Data Mining* (2007), pp. 449–454.

[15] Campos, G. O., Zimek, A., Sander, J., Campello, R. J., Micenková, B., Schubert, E., Assent, I., and Houle, M. E. On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery 30*, 4 (2016), 891–927.

[16] Carbonneau, M.-A., Cheplygina, V., Granger, E., and Gagnon, G. Multiple instance learning: A survey of problem characteristics and applications. *Pattern Recognition 77* (2018), 329–353.

[17] Carrera, D., Rossi, B., Fragneto, P., and Boracchi, G. Online anomaly detection for long-term ecg monitoring using wearable devices. *Pattern Recognition 88* (2019), 482–492.

[18] Cavalcante, R. C., Minku, L. L., and Oliveira, A. L. Fedd: Feature extraction for explicit concept drift detection in time series. In *International Joint Conference on Neural Networks* (2016), pp. 740–747.

[19] Chalapathy, R., and Chawla, S. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407* (2019).

[20] Chalapathy, R., Menon, A. K., and Chawla, S. Anomaly detection using one-class neural networks. *arXiv preprint arXiv:1802.06360* (2018).

[21] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM Computing Surveys 41*, 3 (2009), 1–72.

[22] Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering 24*, 5 (2010), 823–839.

[23] Chattopadhyay, R., Sun, Q., Fan, W., Davidson, I., Panchanathan, S., and Ye, J. Multisource domain adaptation and its application to early detection of fatigue. *ACM Transactions on Knowledge Discovery from Data 6*, 4 (2012), 18.

[24] Chen, X.-y., and Zhan, Y.-y. Multi-scale anomaly detection algorithm based on infrequent pattern of time series. *Journal of Computational and Applied Mathematics 214*, 1 (2008), 227–237.

[25] Cheng, H., Yan, X., Han, J., and Hsu, C.-W. Discriminative frequent pattern analysis for effective classification. In *IEEE 23rd International Conference on Data Engineering* (2007), pp. 716–725.

[26] Choi, S. W., Yoo, C. K., and Lee, I.-B. Overall statistical monitoring of static and dynamic patterns. *Industrial & Engineering Chemistry Research 42*, 1 (2003), 108–117.

[27] Chou, J.-S., and Telaga, A. S. Real-time detection of anomalous power consumption. *Renewable and Sustainable Energy Reviews 33* (2014), 400–411.

[28] Cleary, J., and Witten, I. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications 32*, 4 (1984), 396–402.

[29] Cook, A., Misirli, G., and Fan, Z. Anomaly detection for IoT time-series data: A survey. *IEEE Internet of Things Journal* (2019).

[30] Cui, Y., Ahmad, S., and Hawkins, J. Continuous online sequence learning with an unsupervised neural network model. *Neural Computation 28*, 11 (2016), 2474–2504.

[31] Das, S., Wong, W.-K., Dietterich, T., Fern, A., and Emmott, A. Incorporating expert feedback into active anomaly discovery. In *IEEE 16th International Conference on Data Mining* (2016), pp. 853–858.

[32] Davis, J., and Goadrich, M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine learning* (2006), pp. 233–240.

[33] De Brabandere, A., Robberechts, P., De Beéck, T. O., and Davis, J. Automating feature construction for multi-view time series

data. In *ECMLPKDD Workshop on Automating Data Science* (2019), pp. 1–16.

[34] Decroos, T., Schütte, K., De Beéck, T. O., Vanwanseele, B., and Davis, J. AMIE: Automatic monitoring of indoor exercises. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2018), Springer, pp. 424–439.

[35] Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research 7* (2006), 1–30.

[36] Dewitte, T., Meert, W., and Van Wolputte, E. Anomaly detection for CERN beam transfer installations using machine learning. In *Proceedings of the 17th International Conference on Accelerator and Large Experimental Control Systems* (2019).

[37] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. Querying and mining of time series data: Experimental comparison of representations and distance measures. In *Proceedings of the VLDB Endowment* (2008), vol. 1, pp. 1542–1552.

[38] Domingues, R., Filippone, M., Michiardi, P., and Zouaoui, J. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition 74* (2018), 406–421.

[39] Dua, D., and Graff, C. UCI machine learning repository, 2017.

[40] Dunning, T., and Friedman, E. *Practical machine learning: A new look at anomaly detection.* O'Reilly Media Inc., 2014.

[41] Emmott, A., Das, S., Dietterich, T., Fern, A., and Wong, W.-K. A meta-analysis of the anomaly detection problem. *arXiv preprint arXiv:1503.01158* (2015).

[42] Emmott, A. F., Das, S., Dietterich, T., Fern, A., and Wong, W.-K. Systematic construction of anomaly detection benchmarks from real data. In *Proceedings of the ACM SIGKDD workshop on Outlier Detection and Description* (2013), pp. 16–21.

[43] European Environment Agency. Water scarcity, 2008. `http://www.eea.europa.eu/themes/water/featured-articles/water-scarcity`, Site visited 2017-04-01.

[44] Fagiani, M., Squartini, S., Gabrielli, L., Severini, M., and Piazza, F. A statistical framework for automatic leakage detection in smart water and gas grids. *Energies 9*, 9 (2016), 640–665.

[45] Feremans, L., Vercruyssen, V., Cule, B., Meert, W., and Goethals, B. Pattern-based anomaly detection in mixed-type time series. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2019), Springer.

[46] Feremans, L., Vercruyssen, V., Meert, W., Cule, B., and Goethals, B. A framework for pattern mining and anomaly detection in multi-dimensional time series and event logs. In *International Workshop on New Frontiers in Mining Complex Patterns* (2019), Springer, pp. 3–20.

[47] Fernando, B., Habrard, A., Sebban, M., and Tuytelaars, T. Subspace alignment for domain adaptation. *arXiv preprint arXiv:1409.5241* (2014).

[48] Fournier-Viger, P., Lin, J. C.-W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., and Lam, H. T. The SPMF open-source data mining library version 2. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2016), Springer.

[49] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research 17*, 1 (2016), 2096–2030.

[50] Gershenfeld, N., Krikorian, R., and Cohen, D. The internet of things. *Scientific American 291*, 4 (2004), 76–81.

[51] Giannoni, F., Mancini, M., and Marinelli, F. Anomaly detection models for IoT time series data. *arXiv preprint arXiv:1812.00890* (2018).

[52] Goldstein, M., and Dengel, A. Histogram-based outlier score (HBOS): A fast unsupervised anomaly detection algorithm. In *KI-2012: Poster and Demo Track* (2012), Citeseer, pp. 59–63.

[53] Goldstein, M., and Uchida, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS One 11*, 4 (2016).

[54] Gong, B., Shi, Y., Sha, F., and Grauman, K. Geodesic flow kernel for unsupervised domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition* (2012), pp. 2066–2073.

[55] Görnitz, N., Kloft, M., Rieck, K., and Brefeld, U. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research 46* (2013), 235–262.

[56] Gretton, A., Borgwardt, K. M., Rasch, M., Schölkopf, B., and Smola, A. J. A kernel method for the two-sample-problem. In *Advances in Neural Information Processing Systems* (2007), pp. 513–520.

[57] Grubbs, F. E. Procedures for detecting outlying observations in samples. *Technometrics 11*, 1 (1969), 1–21.

[58] Grubbs, F. E., et al. Sample criteria for testing outlying observations. *The Annals of Mathematical Statistics 21*, 1 (1950), 27–58.

[59] Hariri, S., Kind, M. C., and Brunner, R. J. Extended isolation forest. *arXiv preprint arXiv:1811.02141* (2018).

[60] Hawkins, D. M. *Identification of outliers*. Chapman and Hall, 1980.

[61] He, Z., Deng, S., and Xu, X. Outlier detection integrating semantic knowledge. In *International Conference on Web-Age Information Management* (2002), Springer, pp. 126–131.

[62] He, Z., Xu, X., and Deng, S. Discovering cluster-based local outliers. *Pattern Recognition Letters 24*, 9 (2003), 1641–1650.

[63] He, Z., Xu, X., Huang, Z. J., and Deng, S. FP-outlier: Frequent pattern based outlier detection. *Computer Science and Information Systems 2*, 1 (2005), 103–118.

[64] Hemalatha, C. S., Vaidehi, V., and Lakshmi, R. Minimal infrequent pattern based approach for mining outliers in data streams. *Expert Systems with Applications 42*, 4 (2015), 1998–2012.

[65] Hendrickx, K., Meert, W., Mollet, Y., Gyselinck, J., Cornelis, B., Gryllias, K., and Davis, J. A general anomaly detection framework for fleet-based condition monitoring of machines. *Mechanical Systems and Signal Processing 139* (2020).

[66] Henke, N., Bughin, J., Chui, M., Manyika, J., Saleh, T., Wiseman, B., and Sethupathy, G. The age of analytics: Competing in a data-driven world. McKinsey Global Institute.

[67] Jakkula, V., and Cook, D. Outlier detection in smart environment structured power datasets. In *6th International Conference on Intelligent Environments* (2010), IEEE, pp. 29–33.

[68] Janetzko, H., Stoffel, F., Mittelstadt, S., and Keim, D. A. Anomaly detection for visual analytics of power consumption data. *Computers & Graphics 38* (2014), 27–37.

[69] Karlsson, I., Papapetrou, P., and Boström, H. Generalized random shapelet forests. *Data Mining and Knowledge Discovery 30*, 5 (2016), 1053–1085.

[70] Keller, F., Muller, E., and Bohm, K. HiCS: High contrast subspaces for density-based outlier ranking. In *IEEE 28th international Conference on Data Engineering* (2012), pp. 1037–1048.

[71] Keogh, E., and Lin, J. Clustering of time-series subsequences is meaningless: Implications for previous and future research. *Knowledge and Information Systems 8*, 2 (2005), 154–177.

[72] Keogh, E., Lonardi, S., and Ratanamahatana, C. A. Towards parameter-free data mining. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), pp. 206–215.

[73] Kha, N. H., and Anh, D. T. From cluster-based outlier detection to time series discord discovery. In *Revised Selected Papers of the PAKDD 2015 Workshops on Trends and Applications in Knowledge Discovery and Data Mining* (2015), Springer-Verlag, pp. 16–28.

[74] Knorr, E. M., and Ng, R. T. A unified notion of outliers: Properties and computation. In *KDD* (1997), pp. 219–222.

[75] Kouw, W. M., and Loog, M. An introduction to domain adaptation and transfer learning. *arXiv:1812.11806* (2018).

[76] Kouw, W. M., and Loog, M. A review of domain adaptation without target labels. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[77] Kriegel, H.-P., Kröger, P., Schubert, E., and Zimek, A. Outlier detection in axis-parallel subspaces of high dimensional data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2009), Springer, pp. 831–838.

[78] Kriegel, H.-P., Schubert, M., and Zimek, A. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2008), pp. 444–452.

[79] Langone, R., Alzate Perez, C., De Ketelaere, B., Vlasselaer, J., Meert, W., and Suykens, J. LS-SVM based spectral clustering and regression for predicting maintenance of industrial machines. *Engineering Applications of Artificial Intelligence 37* (2015), 268–278.

[80] Laptev, N., Amizadeh, S., and Flint, I. 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* (2015), pp. 1939–1947.

[81] Lei, Y., Li, N., Guo, L., Li, N., Yan, T., and Lin, J. Machinery health prognostics: A systematic review from data acquisition to RUL prediction. *Mechanical Systems and Signal Processing 104* (2018), 799–834.

[82] Lewis, D. D., and Gale, W. A. A sequential algorithm for training text classifiers. In *17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (1994), Springer-Verlag, pp. 3–12.

[83] Lin, J., Keogh, E., Lonardi, S., and Chiu, B. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (2003), pp. 2–11.

[84] Lines, J., and Bagnall, A. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery 29*, 3 (2015), 565–592.

[85] Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. In *IEEE International Conference on Data Mining* (2008), pp. 413–422.

[86] Liu, H., Shah, S., and Jiang, W. On-line outlier detection and data cleaning. *Computers & Chemical Engineering 28*, 9 (2004), 1635–1647.

[87] Liu, Y., Li, Z., Zhou, C., Jiang, Y., Sun, J., Wang, M., and He, X. Generative adversarial active learning for unsupervised outlier detection. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[88] Long, M., Wang, J., Ding, G., Sun, J., and Yu, P. S. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE International Conference on Computer Vision* (2013), pp. 2200–2207.

[89] Long, M., Wang, J., Ding, G., Sun, J., and Yu, P. S. Transfer joint matching for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 1410–1417.

[90] Loureiro, A., Torgo, L., and Soares, C. Outlier detection using clustering methods: A data cleaning application. In *Proceedings of KDNet*

Symposium on Knowledge-based systems for the Public Sector (2004), Springer Bonn.

[91] Ma, H., Hu, Y., and Shi, H. Fault detection and identification based on the neighborhood standardized local outlier factor method. *Industrial & Engineering Chemistry Research 52*, 6 (2013), 2389–2402.

[92] Malhotra, P., Vig, L., Shroff, G., and Agarwal, P. Long short term memory networks for anomaly detection in time series. In *23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (2015).

[93] Marques, H. O., Campello, R. J., Sander, J., and Zimek, A. Internal evaluation of unsupervised outlier detection. *ACM Transactions on Knowledge Discovery from Data 14*, 4 (2020), 1–42.

[94] Marteau, P.-F., Soheily-Khah, S., and Béchet, N. Hybrid isolation forest-application to intrusion detection. *arXiv preprint arXiv:1705.03800* (2017).

[95] Mei, H., and Eisner, J. M. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems* (2017), pp. 6754–6764.

[96] Mei, H., Qin, G., and Eisner, J. Imputing missing events in continuous-time event streams. *arXiv preprint arXiv:1905.05570* (2019).

[97] Mignone, P., Pio, G., D'Elia, D., and Ceci, M. Exploiting transfer learning for the reconstruction of the human gene regulatory network. *Bioinformatics* (2019).

[98] Mordelet, F., and Vert, J.-P. A bagging SVM to learn from positive and unlabeled examples. *Pattern Recognition Letters 37* (2014), 201–209.

[99] Mudholkar, G. S., and Srivastava, D. K. Exponentiated Weibull family for analyzing bathtub failure-rate data. *IEEE Transactions on Reliability 42*, 2 (1993), 299–302.

[100] Mueen, A., Keogh, E., Zhu, Q., Cash, S., and Westover, B. Exact discovery of time series motifs. In *Proceedings of the 2009 SIAM International Conference on Data Mining* (2009), pp. 473–484.

[101] Pan, S. J., Tsang, I. W., Kwok, J. T., and Yang, Q. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks 22*, 2 (2011), 199–210.

[102] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 10 (2010), 1345–1359.

[103] Papadimitriou, S., Kitagawa, H., Gibbons, P. B., and Faloutsos, C. Loci: Fast outlier detection using the local correlation integral. In *19th IEEE International Conference on Data Engineering* (2003), pp. 315–326.

[104] Parzen, E. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics 33*, 3 (1962), 1065–1076.

[105] Patabendige, S., Cardell-Oliver, R., Wang, R., and Liu, W. Detection and interpretation of anomalous water use for non-residential customers. *Environmental Modelling & Software 100* (2018), 291–301.

[106] Paulheim, H., and Meusel, R. A decomposition of the outlier detection problem into a set of supervised learning problems. *Machine Learning 100*, 2-3 (2015), 509–531.

[107] Perini, L., Galvin, C., and Vercruyssen, V. A ranking stability measure for quantifying the robustness of anomaly detection methods. In *2nd Workshop on Evaluation and Experimental Design in Data Mining and Machine Learning at ECML PKDD* (2020).

[108] Perini, L., Vercruyssen, V., and Davis, J. Class prior estimation in active positive and unlabeled learning. In *29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence* (2020), pp. 2915–2921.

[109] Perini, L., Vercruyssen, V., and Davis, J. Quantifying the confidence of anomaly detectors in their example-wise predictions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2020), Springer Verlag.

[110] Pevnỳ, T. Loda: Lightweight on-line detector of anomalies. *Machine Learning 102*, 2 (2016), 275–304.

[111] Raciti, M., Cucurull, J., and Nadjm-Tehrani, S. Anomaly detection in water management systems. In *Critical Infrastructure Protection*. Springer, 2012, pp. 98–119.

[112] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. Searching and mining trillions of time series subsequences under dynamic time warping. In *18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2012), pp. 262–270.

[113] Ramaswamy, S., Rastogi, R., and Shim, K. Efficient algorithms for mining outliers from large data sets. *ACM SIGMOD Record 29*, 2 (2000), 427–438.

[114] Randall, R. B. *Vibration-based condition monitoring: Industrial, aerospace and automotive applications*. John Wiley & Sons, 2011.

[115] Redko, I., Habrard, A., and Sebban, M. On the analysis of adaptability in multi-source domain adaptation. *Machine Learning 108*, 8-9 (2019), 1635–1652.

[116] Ren, X., Zhu, K., Cai, T., and Li, S. Fault detection and diagnosis for nonlinear and non-gaussian processes based on copula subspace division. *Industrial & Engineering Chemistry Research 56*, 40 (2017), 11545–11564.

[117] Robberechts, P., Bosteels, M., Davis, J., and Meert, W. Query log analysis: Detecting anomalies in DNS traffic at a TLD resolver. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2018), Springer, pp. 55–67.

[118] Rousseeuw, P., and Leroy, A. *Robust Regression and Outlier Detection*. John Wiley & Sons, 1987.

[119] Rousseeuw, P. J., and Driessen, K. V. A fast algorithm for the minimum covariance determinant estimator. *Technometrics 41*, 3 (1999), 212–223.

[120] Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., and Kloft, M. Deep one-class classification. In *International Conference on Machine Learning* (2018), pp. 4393–4402.

[121] Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., and Platt, J. C. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems* (2000), pp. 582–588.

[122] Schouterden, J., Davis, J., and Blockeel, H. Multi-directional rule set learning. In *Discovery Science* (2020), Springer, pp. 517–532.

[123] Schubert, E., Zimek, A., and Kriegel, H.-P. Generalized outlier detection with flexible kernel density estimates. In *Proceedings of the 2014 SIAM International Conference on Data Mining* (2014), pp. 542–550.

[124] Schubert, E., Zimek, A., and Kriegel, H.-P. Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection. *Data Mining and Knowledge Discovery 28*, 1 (2014), 190–237.

[125] Sheng, V. S., Provost, F., and Ipeirotis, P. G. Get another label? Improving data quality and data mining using multiple, noisy labelers. In *14th ACM SIGKDD international conference on Knowledge Discovery and Data Mining* (2008), pp. 614–622.

[126] Siddiqui, M. A., Fern, A., Dietterich, T. G., Wright, R., Theriault, A., and Archer, D. W. Feedback-guided anomaly discovery via online optimization. In *24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2018), pp. 2200–2209.

[127] Sikorska, J., Hodkiewicz, M., and Ma, L. Prognostic modelling options for remaining useful life estimation by industry. *Mechanical Systems and Signal Processing 25*, 5 (2011), 1803–1836.

[128] Sommer, R., and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy* (2010), pp. 305–316.

[129] Spinoni, J., Vogt, J. V., Naumann, G., Barbosa, P., and Dosio, A. Will drought events become more frequent and severe in europe? *International Journal of Climatology* (2017).

[130] Steinbuss, G., and Böhm, K. Benchmarking unsupervised outlier detection with realistic synthetic data. *arXiv preprint arXiv:2004.06947* (2020).

[131] Sun, B., Feng, J., and Saenko, K. Return of frustratingly easy domain adaptation. In *30th AAAI Conference on Artificial Intelligence* (2016), p. 2058–2065.

[132] Swersky, L., Marques, H. O., Sander, J., Campello, R. J., and Zimek, A. On the evaluation of outlier detection and one-class classification methods. In *IEEE International Conference on Data Science and Advanced Analytics* (2016), pp. 1–10.

[133] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., and Liu, C. A survey on deep transfer learning. In *International Conference on Artificial Neural Networks* (2018), Springer, pp. 270–279.

[134] Tang, J., Chen, Z., Fu, A. W.-C., and Cheung, D. W. Enhancing effectiveness of outlier detections for low density patterns. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2002), Springer, pp. 535–548.

[135] Tax, D. M., and Duin, R. P. Support vector data description. *Machine Learning 54*, 1 (2004), 45–66.

[136] Ting, K. M., Liu, F. T., and Zhou, Z. Isolation Forest. In *8th IEEE International Conference on Data Mining* (2008), pp. 413–422.

[137] Trittenbach, H., Englhardt, A., and Böhm, K. An overview and a benchmark of active learning for one-class classification. *arXiv preprint 1808.04759* (2018).

[138] Van Craenendonck, T., Meert, W., Dumančić, S., and Blockeel, H. COBRAS TS: A new approach to semi-supervised clustering of time series. In *Discovery Science* (2018), Springer, pp. 179–193.

[139] Van Erven, T., and Harremos, P. Rényi divergence and Kullback-Leibler divergence. *IEEE Transactions on Information Theory 60*, 7 (2014), 3797–3820.

[140] Van Stein, B., Van Leeuwen, M., and Bäck, T. Local subspace-based outlier detection using global neighbourhoods. In *2016 IEEE International Conference on Big Data* (2016), pp. 1136–1142.

[141] Venkatasubramanian, V., Rengaswamy, R., Yin, K., and Kavuri, S. N. A review of process fault detection and diagnosis. *Computers & Chemical Engineering 27* (2003), 293–346.

[142] Vercruyssen, V., De Raedt, L., and Davis, J. Qualitative spatial reasoning for soccer pass prediction. In *Machine Learning and Sports Analytics Workshop at ECML PKDD* (2016).

[143] Vercruyssen, V., Meert, W., and Davis, J. Transfer learning for time series anomaly detection. In *Interactive Adaptive Learning Workshop at ECML PKDD* (2017), pp. 27–37.

[144] Vercruyssen, V., Meert, W., and Davis, J. "Now you see it, now you don't!" Detecting suspicious pattern absences in continuous time series. In *SIAM International Conference on Data Mining* (2020), pp. 127–135.

[145] Vercruyssen, V., Meert, W., Verbruggen, G., Maes, K., Bäumer, R., and Davis, J. Semi-supervised anomaly detection with an application to water analytics. In *IEEE International Conference on Data Mining* (2018).

[146] Vincent, V., Wannes, M., and Jesse, D. Transfer learning for anomaly detection through localized and unsupervised instance selection. In *34th AAAI Conference on Artificial Intelligence* (2020), pp. 6054–6061.

[147] Wagstaff, K., Cardie, C., Rogers, S., and Schrödl, S. Constrained k-means clustering with background knowledge. In *International Conference on Machine Learning* (2001), pp. 577–584.

[148] Xiao, Y., Liu, B., Philip, S. Y., and Hao, Z. A robust one-class transfer learning method with uncertain data. *Knowledge and Information Systems 44*, 2 (2015), 407–438.

[149] Xiong, H., Pandey, G., Steinbach, M., and Kumar, V. Enhancing data analysis with noise removal. *IEEE Transactions on Knowledge and Data Engineering 18*, 3 (2006), 304–319.

[150] Yang, Z., Bozchalooi, I. S., and Darve, E. Anomaly detection with domain adaptation. *arXiv preprint arXiv:2006.03689* (2020).

[151] Ye, L., and Keogh, E. Time series shapelets: A new primitive for data mining. In *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2009), pp. 947–956.

[152] Yeh, C.-C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D. F., Mueen, A., and Keogh, E. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *IEEE International Conference on Data Mining* (2016), pp. 1317–1322.

[153] Zaki, M. J., and Meira, W. *Data mining and analysis: Fundamental concepts and algorithms*. Cambridge University Press, 2014.

[154] Zhang, G. P., and Qi, M. Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research 160*, 2 (2005), 501–514.

[155] Zhang, J., Li, W., and Ogunbona, P. Joint geometrical and statistical alignment for visual domain adaptation. *arXiv preprint arXiv:1705.05498* (2017).

[156] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. A comprehensive survey on transfer learning. *arXiv preprint arXiv:1911.02685* (2019).

[157] Zimek, A., and Filzmoser, P. There and back again: Outlier detection between statistical reasoning and data mining algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8*, 6 (2018).

Curriculum Vitae

Vincent Vercruyssen was born in Borgerhout, Anwerpen on August 29 1991. He obtained a bachelors (2012) and masters (2014) diploma in Commercial Engineering from the University of Antwerp. His master specialized in finance and innovation management and his thesis was on quantifying the system dynamics of distributed energy sources. Vincent obtained a second, specialized masters (2015) diploma in Artificial Intelligence, option Engineering and Computer Science, from the KU Leuven with a thesis on predicting hand movement from EEG and ECoG signals. Between his first and second masters, Vincent interned at the Applied Data Science group of the University of Antwerp under supervision of Prof. dr. David Martens, focusing on predicting customer churn.

In September 2015, Vincent started his doctoral studies under supervision of Prof. dr. Jesse Davis in the Machine Learning group within the DTAI lab (Declarative Languages and Artificial Intelligence) at the KU Leuven. Initially, his PhD focused on sports analytics. In 2016 he changed focus to the field of anomaly detection. During his PhD he extensively collaborated with the Colruyt Group and the Adrem Data Lab of the University of Antwerp.

List of Publications

Conference Publications in Chronological Order

1. L. Perini, V. Vercruyssen, and J. Davis (2020). Quantifying the Confidence of Anomaly Detectors in their Example-Wise Predictions. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, Ghent (Belgium).

2. L. Perini, V. Vercruyssen, and J. Davis (2020). Class Prior Estimation in Active Positive and Unlabeled Learning. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, Yokohama (Japan), pp. 2915-2921.

3. V. Vercruyssen, W. Meert, and J. Davis (2020). "Now you see it, now you don't" Detecting Suspicious Pattern Absences in Continuous Time Series. In *Proceedings of the 2020 SIAM International Conference on Data Mining (SDM)*, Cincinnati (US), pp.127-135.

4. V. Vercruyssen, W. Meert, and J. Davis (2020). Transfer Learning for Anomaly Detection through Localized and Unsupervised Instance Selection. In *Proceedings of the* 34^{th} *AAAI Conference on Artificial Intelligence*, New York City (US), vol. 34(4), pp. 6054-6061.

5. L. Feremans*, V. Vercruyssen*, B. Cule, W. Meert, and B. Goethals (2019). Pattern-Based Anomaly Detection in Mixed-type Time Series. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, Würzburg (Germany), pp. 240-256.

6. V. Vercruyssen, G. Verbruggen, K. Maes, R. Bäumer, W. Meert, and J. Davis (2018). Semi-supervised Anomaly Detection with an Application to Water Analytics. In *IEEE International Conference on Data Mining (ICDM)*, Singapore, pp. 527-536.

Workshop Publications in Chronological Order

1. L. Perini, C. Galvin, and V. Vercruyssen (2020). A Ranking Stability Measure for Quantifying the Robustness of Anomaly Detection Methods. Presented at the 2^{nd} *Workshop on Evaluation and Experimental Design in Data Mining and Machine Learning at ECML PKDD*, Ghent (Belgium).

2. F. Feremans, V. Vercruyssen, W. Meert, B. Cule, B. Goethals (2019). A Framework for Pattern Mining and Anomaly Detection in Multi-Dimensional Time Series and Event Logs. In *New Frontiers in Mining Complex Patterns*, Würzburg (Germany), pp. 3-20.

3. V. Vercruyssen, W. Meert, and J. Davis (2017). Transfer Learning for Time Series Anomaly Detection. In *Proceedings of the Workshop and Tutorial on Interactive Adaptive Learning at ECML PKDD*, Skopje (Macedonia), pp. 27-37.

4. V. Vercruyssen, L. De Raedt, and J. Davis (2016). Qualitative Spatial Reasoning for Soccer Pass Prediction. Presented at the *Machine Learning and Data Mining for Sports Analytics Workshop at ECML PKDD*, Riva del Garda (Italy), paper nr. 8.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DTAI
Celestijnenlaan 200A box 2402
B-3001 Leuven
vincent.vercruyssen@kuleuven.be
https://dtai.cs.kuleuven.be/