A data mining perspective on explainable AIOps with applications to software maintenance

Youcef Remil

N° d'ordre NNT : 2023ISAL0072

THÈSE DE DOCTORAT DE L'INSA LYON
MEMBRE DE **L'UNIVERSITÉ DE LYON**

ECOLE DOCTORALE N° 512
MATHÉMATIQUES ET INFORMATIQUE (INFOMATHS)

SPÉCIALITÉ / DISCIPLINE DE DOCTORAT : INFORMATIQUE

Soutenue publiquement le 06/10/2023 par
YOUCEF REMIL

A Data Mining Perspective on Explainable AIOps with Applications to Software Maintenance

Devant le jury composé de :

Romain ROBBES	Directeur de Recherche, CNRS	Rapporteur
Alexandre TERMIER	Professeur, Université de Rennes	Rapporteur
Christel VRAIN	Professeure, Université d'Orléans	Présidente
Arnaud SOULET	Professeur, Université de Tours	Examinateur
Peggy CELLIER	Maître de Conférences HDR, INSA-Rennes	Examinatrice
Jean-François BOULICAUT	Professeur, INSA-Lyon	Directeur de thèse
Mehdi KAYTOUE	Maître de Conférences HDR, INSA-Lyon et Directeur R&I, Infologic	Co-encadrant
Anes BENDIMERAD	Docteur, Data Scientist, Infologic R&D	Co-encadrant

Département FEDORA – INSA Lyon - Ecoles Doctorales

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
ED 206 CHIMIE	**CHIMIE DE LYON** https://www.edchimie-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr	**M. Stéphane DANIELE** C2P2-CPE LYON-UMR 5265 Bâtiment F308, BP 2077 43 Boulevard du 11 novembre 1918 69616 Villeurbanne directeur@edchimie-lyon.fr
ED 341 E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** http://e2m2.universite-lyon.fr Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.e2m2@univ-lyon1.fr	**Mme Sandrine CHARLES** Université Claude Bernard Lyon 1 UFR Biosciences Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69622 Villeurbanne CEDEX e2m2.codir@listes.univ-lyon1.fr
ED 205 EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** http://ediss.universite-lyon.fr Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.ediss@univ-lyon1.fr	**Mme Sylvie RICARD-BLUM** Laboratoire ICBMS - UMR 5246 CNRS - Université Lyon 1 Bâtiment Raulin - 2ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tél : +33(0)4 72 44 82 32 sylvie.ricard-blum@univ-lyon1.fr
ED 34 EDML	**MATÉRIAUX DE LYON** http://ed34.universite-lyon.fr Sec. : Yann DE ORDENANA Tél : 04.72.18.62.44 yann.de-ordenana@ec-lyon.fr	**M. Stéphane BENAYOUN** Ecole Centrale de Lyon Laboratoire LTDS 36 avenue Guy de Collongue 69134 Ecully CEDEX Tél : 04.72.18.64.37 stephane.benayoun@ec-lyon.fr
ED 160 EEA	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** https://edeea.universite-lyon.fr Sec. : Philomène TRECOURT Bâtiment Direction INSA Lyon Tél : 04.72.43.71.70 secretariat.edeea@insa-lyon.fr	**M. Philippe DELACHARTRE** INSA LYON Laboratoire CREATIS Bâtiment Blaise Pascal, 7 avenue Jean Capelle 69621 Villeurbanne CEDEX Tél : 04.72.43.88.63 philippe.delachartre@insa-lyon.fr
ED 512 INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** http://edinfomaths.universite-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	**M. Hamamache KHEDDOUCI** Université Claude Bernard Lyon 1 Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tél : 04.72.44.83.69 direction.infomaths@listes.univ-lyon1.fr
ED 162 MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** http://edmega.universite-lyon.fr Sec. : Philomène TRECOURT Tél : 04.72.43.71.70 Bâtiment Direction INSA Lyon mega@insa-lyon.fr	**M. Jocelyn BONJOUR** INSA Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69621 Villeurbanne CEDEX jocelyn.bonjour@insa-lyon.fr
ED 483 ScSo	**ScSo**[1] https://edsciencessociales.universite-lyon.fr Sec. : Mélina FAVETON Tél : 04.78.69.77.79 melina.faveton@univ-lyon2.fr	**M. Bruno MILLY** (INSA : J.Y. TOUSSAINT) Univ. Lyon 2 Campus Berges du Rhône 18, quai Claude Bernard 69365 LYON CEDEX 07 Bureau BEL 319 bruno.milly@univ-lyon2.fr

[1] ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

List of Publications

French National Conferences:

- **Youcef Remil**, Anes Bendimerad, Marc Plantevit, Céline Robardet and Mehdi Kaytoue. Découverte de Sous-groupes Interprétables pour le Triage d'incidents. In Extraction et Gestion des Connaissances, EGC 2022 [Paper, Code, Presentation, Poster].

International Conferences:

- **Youcef Remil**, Anes Bendimerad, Romain Mathonat, Philippe Chaleat and Mehdi Kaytoue. What makes my queries slow?: Subgroup Discovery for SQL Workload Analysis. In 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021 [Paper, Code, Presentation, Poster].
- **Youcef Remil**, Anes Bendimerad, Marc Plantevit, Céline Robardet and Mehdi Kaytoue. Interpretable Summaries of Black Box Incident Triaging with Subgroup Discovery. In 8th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2021 [Paper, Code, Presentation, Poster].
- **Youcef Remil**. How can Subgroup Discovery help AIOps?. In 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021 [Paper, Presentation, Poster].
- Anes Bendimerad, **Youcef Remil**, Romain Mathonat and Mehdi Kaytoue. On-premise Infrastructure for AIOps in a Software Editor SME: An Experience Report. In 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023. (Accepted) [Paper, Code]
- **Youcef Remil**, Anes Bendimerad, Romain Mathonat, Mathieu Chambard, Marc Plantevit and Mehdi Kaytoue. Subjectively Interesting Subgroups with Hierarchical Targets: Application to Java Memory Analysis. In IEEE International Conference on Data Mining Workshops, ICDMW 2023. (Accepted) [Paper, Code]
- **Youcef Remil**, Anes Bendimerad, Romain Mathonat, Chedy Raissi and Mehdi Kaytoue. DeepLSH: Deep Locality-Sensitive Hash Learning for Fast and Efficient Near-Duplicate Crash Report Detection. In 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024. (Accepted) [Paper, Code]

International journals:

- **Youcef Remil**, Anes Bendimerad, Romain Mathonat and Mehdi Kaytoue. AIOps solutions for Incident Management: Technical Guidelines and A Comprehensive Literature Review. In ACM Transactions on Software Engineering and Methodology, TOSEM 2023. (Under submission)

Remerciements

Au-delà de l'aspect formel de toute thèse de doctorat, qui d'ordinaire met en avant un nom en particulier sur la page de couverture, il s'agit en vrai de reconnaître que ce travail est le fruit d'un sacré effort collectif. Cette contribution ne se limite pas aux personnes dont les noms trônent en première page, mais embrasse des collaborations intellectuelles, des échanges fructueux, des inspirations partagées, et des conseils précieux. Elle s'étend également à l'incroyable soutien humain qui m'a accompagné aux moments de doute tout au long de cette aventure exceptionnelle. Cette thèse, bien plus qu'un simple ajout académique à mon CV, est avant tout, une opportunité extraordinaire qui m'est offerte de croissance et de maturation à plusieurs égards, tant sur le plan professionnel que personnel. Elle m'a permis d'élargir mes horizons et perspectives, et m'a appris la valeur inestimable des échecs, qui, tout comme les réussites, sont des étapes essentielles de notre parcours. Elle m'a incité à accueillir les conseils et les critiques comme des étincelles pour ma progression et mon développement. Enfin, elle m'a inspiré à persévérer, à saisir chaque opportunité et à partir sur le moindre écart pour avancer. Ainsi, en préambule à ce mémoire de thèse, je tiens à exprimer ma profonde gratitude envers toutes les personnes qui m'ont apporté leur aide et leur soutien au cours de ces trois années. Votre contribution significative a été plus qu'un support pour l'élaboration de ce mémoire, et je vous en suis infiniment reconnaissant.

Pour commencer, je tiens à exprimer ma sincère gratitude envers Romain Robbes, directeur de recherche CNRS, et Alexandre Termier, professeur à l'université de Rennes, d'avoir accepté de consacrer leur temps et leurs efforts considérables en tant que rapporteurs de cette thèse. Je saisis également cette opportunité pour exprimer ma profonde gratitude envers Christel Vrain, Professeure à l'université d'Orléans, Peggy Cellier, Maître de conférences à l'INSA-Rennes, et Arnaud Soulet, Maître de conférences à l'université de Tours, d'avoir accepté de faire partie du jury de ma thèse en tant qu'examinateurs.

Je souhaite exprimer ma profonde gratitude envers l'entreprise Infologic, en particulier envers André Chabert, directeur et fondateur de l'entreprise, ainsi qu'envers Maxime Chabert, pour m'avoir offert cette opportunité exceptionnelle et rare de mener une thèse CIFRE. Leur engagement à mettre à ma disposition les moyens et les ressources nécessaires pour aborder des problématiques réelles de l'industrie en collaboration avec la recherche, tout en encourageant la créativité, est admirable. Je suis également reconnaissant de leur détermination à fournir des solutions concrètes et de haute qualité à leurs clients. Je suis donc extrêmement enthousiaste à l'idée de poursuivre cette collaboration au sein de la famille Infologic. Je tiens également à exprimer mes sincères remerciements à tous les membres de l'agence de Lyon pour les expériences formidables que j'ai vécues en leur compagnie.

En ce qui concerne mon encadrement scientifique, ainsi que mon expérience au sein de l'entreprise, il est difficile de trouver suffisamment de mots pour exprimer ma gratitude. Mehdi Kaytoue, je tiens à te remercier d'avoir toujours cru en moi. Ta confiance absolue pendant cette période de thèse, ton soutien indéfectible envers mes idées, tes conseils avisés, et tes remarques toujours pertinentes et constructives ont été d'une valeur inestimable. Je suis reconnaissant pour la chance que tu m'as donnée d'élargir mon réseau professionnel et de me lancer sur de nombreuses opportunités. Je suis plus qu'enthousiaste à l'idée de poursuivre notre collaboration. Un immense merci également à toi, Anes Bendimerad, qui es en grande

partie responsable de ma présence au sein d'Infologic. Ta contribution à la concrétisation de ce travail a été immense, et j'ai énormément bénéficié de tes compétences remarquables en recherche et en développement. Tu as été bien plus qu'un encadrant, tu es devenu un véritable binôme qui connaît chaque détail de mes travaux de thèse. Merci d'être toujours à l'écoute et d'incarner l'encadrant idéal que tout doctorant rêve d'avoir. Je tiens également à exprimer ma gratitude envers Jean-François Boulicaut pour ses remarques pertinentes et son engagement à assurer le meilleur déroulement possible de cette thèse. Enfin, un grand merci à Romain Mathonat pour sa contribution précieuse à ce processus et de m'avoir aidé à avancer sur de nombreux aspects techniques.

Finalement, je souhaite exprimer ma sincère gratitude envers mes parents, mes frères et sœurs, ainsi que mes grands-mères, d'avoir été un véritable pilier de soutien, non seulement pendant cette épreuve, mais tout au long de mon parcours. Merci de m'avoir accompagné à chaque étape de ma vie, de veiller à ce que je sois toujours dans les meilleures conditions pour que je puisse me donner à fond et me concentrer uniquement sur mes objectifs et mes rêves. Sans vous, ce travail n'aurait jamais pu être accompli. Je tiens également à exprimer ma reconnaissance envers tous mes nombreux amis pour les moments de joie et les belles discussions que nous avons partagés ensemble. C'est un immense plaisir d'avoir chacun de vous dans ma vie.

Abstract

The genuine supervision of modern IT systems presents new challenges in terms of scalability, reliability, and efficiency. Traditional operations and maintenance systems that rely on manual tasks and individual troubleshooting are inefficient. Rule-based inference engines, although useful for detecting anomalies and automating resolution, are limited in handling the large number of alerts generated by IT systems. Artificial Intelligence for Operating Systems (AIOps) proposes the use of advanced analytics and machine learning to improve and automate supervision systems. However, there are several challenges in this field. Firstly, the lack of unified terminology makes it difficult to compare contributions from different disciplines. The requirements and metrics for constructing effective AIOps models are not well-defined. Secondly, AIOps has primarily focused on predictive models for anomaly detection and failure prediction, neglecting descriptive models that can handle data quality and complexity concerns. Thirdly, the reliance on opaque black box models limits their adoption by industry practitioners who need a clear understanding of the decision-making process of maintenance models. Lastly, existing AIOps solutions often overlook performance evaluation and scalability issues when developing and evaluating incident management models.

As part of this Ph.D. thesis, we propose several contributions to tackle these challenges more effectively. Firstly, we offer a systematic approach to AIOps that organizes the extensive knowledge surrounding it. By categorizing data-driven approaches from various research areas and disciplines according to industry standards and requirements, we provide a cohesive framework. Secondly, we explore the application of Subgroup Discovery and its generalization Exceptional Model Mining, a promising data mining technique, in the context of AIOps. This well-defined framework allows for the extraction of valuable hypotheses from large and diverse datasets. It enables users to understand, interact with, and interpret the underlying processes behind predictive models. Our contributions in this area include a practical application focused on identifying suspicious query fragments in large SQL workloads to pinpoint performance degradation issues. Additionally, we develop an interpretation mechanism for incident triage models, providing contextualized explanations for the model's decisions. Furthermore, we address the challenging problem of memory Java analysis using huge and complex datasets that incorporate hierarchical data. Lastly, we address the issue of scalability by studying incident deduplication, a well-known problem in the industry. Our goal is to efficiently retrieve the most similar crash reports by combining locality-sensitive hashing and learning-to-hash techniques within a unified framework. To ensure the relevance and practicality of our propositions, this project involves collaboration between data mining researchers and practitioners from Infologic, a French software editor.

Keywords: AIOps, Incident Management Procedure, Data Mining, Subgroup Discovery, Explainable AI, Locality-Sensitive Hashing.

Résumé

La supervision des systèmes informatiques modernes présente de nouveaux défis en termes de scalabilité, de fiabilité et d'efficacité. Les méthodes traditionnelles de maintenance basées sur l'exécution de tâches manuelles ont prouvé leur inefficacité. De manière similaire, les systèmes experts à base de règles sont limités dans leur capacité actuelle à gérer et anticiper le grand nombre d'alertes générées par les systèmes informatiques. AIOps for Operating Systems (AIOps) propose d'utiliser des techniques avancées d'apprentissage automatique centrées sur la donnée pour améliorer et automatiser les systèmes de supervision. Cependant, il existe plusieurs défis à relever pour concrétiser cette vision, qui sont partagés à la fois par la communauté scientifique et les ingénieurs sur le terrain. Tout d'abord, le manque d'une terminologie claire et unifiée dans le domaine de l'AIOps rend difficile la progression, l'implémentation et la comparaison des contributions provenant de différentes disciplines. De plus, les exigences et les métriques nécessaires à la construction de modèles AIOps, alignés avec les contraintes industrielles, ne sont pas suffisamment élaborées. Deuxièmement, les contributions théoriques en matière d'AIOps se sont principalement concentrées sur les modèles prédictifs pour la détection et prédictions des incidents, en négligeant souvent la capacité des modèles descriptifs à gérer et résoudre les défis liés à la qualité, à la complexité, au volume et à la diversité des données. Troisièmement, la dépendance excessive aux modèles boîte noire opaques limite leur adoption par les praticiens de l'industrie. Enfin, les solutions AIOps existantes ne prêtent pas toujours suffisamment d'importance à l'évaluation des performances des modèles et aux problèmes de scalabilité lors du développement et de l'évaluation des modèles.

Nous proposons d'abord une approche systématique de l'AIOps qui organise les connaissances dans ce nouveau domaine de recherche en fournissant une catégorisation en accord avec les normes et les exigences de l'industrie. Deuxièmement, nous explorons l'application de la découverte de sous-groupes qui est une technique prometteuse de fouille de données qui permet l'extraction d'hypothèses intéressantes à partir de vastes ensembles de données diversifiées. Ainsi, les utilisateurs sont en mesure de comprendre, d'interagir avec et d'interpréter les processus sous-jacents aux modèles. Nos contributions dans ce domaine comprennent une application pratique axée sur l'identification de fragments de requêtes SQL suspects permettant de localiser les problèmes de dégradation de performances. De plus, nous développons un mécanisme d'interprétation pour les modèles de triage des incidents, offrant des explications contextualisées pour les décisions prises par le modèle. Enfin, nous abordons le problème de l'analyse des problématiques de saturation de la mémoire Java, caractérisé par un ensemble de données volumineux et complexes intégrant des données hiérarchiques. Nous traitons également de la scalabilité en étudiant un problème connu de l'industrie qui est la détection de la déduplication des incidents. Notre objectif est de rechercher de manière efficace et scalable les rapports de plantage les plus similaires en combinant des techniques de hachage sensibles à la localité (LSH) et des techniques d'apprentissage de hachage dans un cadre unifié.

Mots clés: AIOps, Procédure de gestion des incidents, Fouilles de données, Découverte de sous-groupes, IA Explicable, Hachage sensible à la localité.

Table of contents

Chapter 1

Introduction

1.1 Context and Motivation

In today's digital era, Information Technology (IT) has become an indispensable element for the automation of business processes across a multitude of industries. Specialized information systems and Enterprise Resource Planning (ERP) solutions play a crucial role in the ongoing transformation, as they are extensively employed to oversee internal operations, customer and supplier relations, as well as Industry 4.0 factories that heavily rely on real-time monitoring and observability. Infologic company is one of France's leading providers of ERP solutions for the agri-food, health nutrition, and cosmetic sectors and has established a remarkable reputation over its 40 years in the market with a plethora of provided services and modules. This comprises commercial oversight, financial and accounting management, decision-making processes, inventory management, quality assurance, and traceability [131]. Beyond offering these services, Infologic also provides its customers with continuous consultation and proactive maintenance support. Having experienced significant growth of over 10% per year, Infologic is now confronted with the challenge of enhancing operational efficiency, all while ensuring optimal customer satisfaction levels are upheld. In fact, Infologic ERP system is currently deployed by hundreds of food industries in France, each of which has at least one server and any failure can be costly. Therefore, Infologic is committed to providing comprehensive maintenance support of its ERP system across its entire server fleet, namely Copilote. This ensures the uninterrupted and reliable provision of services while also mitigating any potential system faults and security threats. To achieve this goal, Infologic employs thorough maintenance protocols that extend to all machines, databases, Copilote instances, and workstations belonging to its clients. These protocols encompass various levels of maintenance, from physical to business-level maintenance.

Prior to 2019, Infologic maintenance protocol was predominantly reliant on reactive maintenance (also known as breakdown maintenance). This approach involves repairing system malfunctions only after they had occurred. Often referred to as the *run to failure* approach, reactive maintenance aims to address immediate issues in a timely manner, without considering the long-term performance of the equipment or service. This shortsighted approach can lead to costly downtime, lost productivity, and unpredictable future breakdowns. Furthermore, the maintenance department lacked complete visibility into the performance of the monitored instances. The absence of a centralized infrastructure for collecting, storing,

querying, and analyzing both historical and real-time data on all client instance performance metrics e.g., Key Performance Indicators (KPIs) deprived Infologic of full observability into the health of the COPILOTE system and limited its ability to measure the availability of their services. Despite having already embedded probes that monitor certain performance metrics in each instance separately, these probes only respond after critical predefined thresholds are exceeded, providing a simple solution that fails to allow for real-time tracking or predictive analysis of potential failures. Moreover, the federated visibility makes it challenging to address identical alarming symptoms across different clients simultaneously and thus hindering the possibility of batch data analysis.

Since 2020, Infologic has pivoted towards a data-centric approach, expanding its maintenance scoop to cover both proactive and reactive maintenance. This policy involves collecting and storing a broad range of data describing the health status of the supervised COPILOTE instances, along with behavioral and usage trace data. However, this strategy is confronted by a number of internal constraints. Firstly, there is a need for data control, since centralized collection and storage of non-individually sensitive data raises concerns over security and data ownership. To this aim, Infologic proceeds in centralizing and securing the data within its own architecture, opting for a Datalake instead of relying on external cloud-based services. Secondly, the non-disruptiveness of supervised instances is also paramount as these systems are typically in production and must not be disturbed during regular data collection intervals (e.g. every 10 seconds for certain time measurements). Thirdly, given the heterogeneous nature of the underlying infrastructure (machines, databases, COPILOTE instances running on different versions and hardware, etc), a more streamlined data management procedure should be instituted. Scalability and flexibility are also fundamental necessities.

The new strategy, while aimed at optimizing the maintenance workflow, comes with several pain points and obstacles. These challenges primarily stem from conventional engineering practices that are no longer adequate to meet the emerging requirements. The previous approach focused on manually performing laborious tasks and resolving anomalies in a repetitive manner, which fails to address the vast amounts of data that need to be monitored. Moreover, it does not provide the necessary predictive analysis to detect outages sufficiently early. This has led to shifting towards the development of a data-centered, intelligent, and automated platform rooted in the so-called concept of AIOps (AI for Operating Systems) [242, 76, 40, 232, 268, 256] to enhance the incident management process, assisting developers and maintenance engineers from the reporting of the incident to its effective resolution. This platform boosts the supervision system to automatically and effectively detect and triage incidents, assess their priority and severity, determine root causes and apply suitable healing actions, whilst also optimizing the Time To Report (TTR), Time to Engage (TTE), Time To Diagnose (TTD), and Time To Mitigate (TTM). However, in order to realize the full potential of an integrally intelligent solution, it is necessary to conduct a thorough evaluation of the current landscape, identify areas for improvement, and anticipate any potential roadblocks that may arise during the implementation of this solution. This process must also adhere to a set of established efficiency criteria, known as desiderata [177, 202], such as interpretability, maintainability, scalability, and security. In the following, we first outline our current maintenance workflow and process, followed by an examination of a modernization initiative to cope with the identified shortcomings.

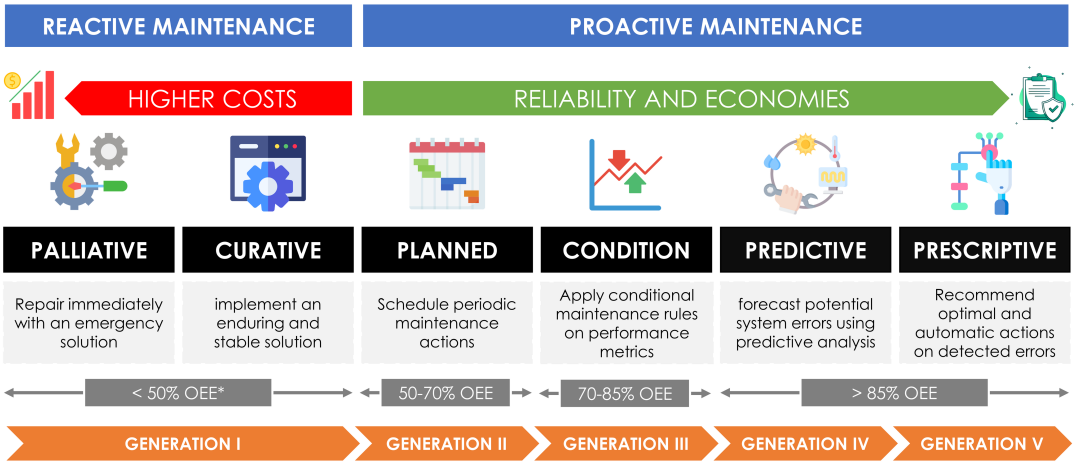


Figure 1.1: Evolution of essential maintenance management strategies. OEE stands for Overall Equipment Effectiveness based on three key factors, availability, performance, and quality.

1.1.1 Exploring Maintenance Practices at Infologic

Infologic offers a unified and dynamic maintenance strategy that encompasses the complete COPILOTE fleet. This includes maintenance for all machines, databases, and application servers. As shown in Figure 1.1, we distinguish between two main types of maintenance, corrective and preventive. Corrective or reactive maintenance is performed when an incident is detected, either by Infologic maintenance staff or through a complaint received from a customer. On the other hand, preventive maintenance is aimed at preventing potential problems from occurring and proactively intervening to rectify them. In both cases, a characterized maintenance call ticket serves as the initial point of primary investigation of the reported problem. This maintenance ticket can be created either manually by humans (often on-call engineers) or generated automatically through user maintenance interfaces.

To elaborate further, Figure 1.2 depicts the distinct patterns of maintenance strategies examined in our analysis. In corrective maintenance, there is generally a time constraint, which can prompt palliative maintenance that resolves the issue partly or totally, allowing the underlying activity to be performed. For example, this can be accomplished through a technical provisional configuration. Nevertheless, curative maintenance must be carried out in order to implement a stable solution to forestall the issue from occurring again with the same or other customers. Preventive maintenance, on the other hand, is a set of proactive measures, frequently utilizing scheduled maintenance routines and conditional maintenance protocols that enable the assessment of system functionality, allowing for the identification of anomalies that could potentially result in performance degradation, malfunctions, or errors. Furthermore, the implementation of Proactive Maintenance via predictive analytics is underway. This approach lies heavily on AIOps and leverages the use of advanced machine learning algorithms and big data mining to forecast potential system malfunctions by tracking and analyzing historical data patterns. Prescriptive maintenance goes beyond predictive maintenance by employing a proactive approach to intelligently schedule and plan asset maintenance. Unlike predictive maintenance, which relies solely on historical data, prescriptive

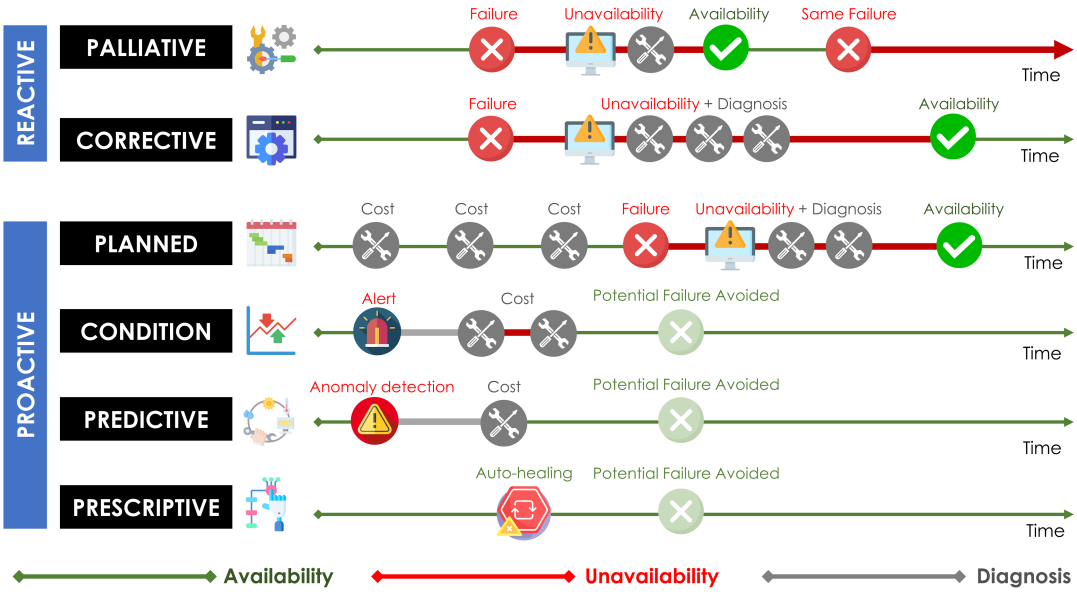


Figure 1.2: Behavioral scheme of the different maintenance protocols. Adapted and improved from [101].

maintenance also uses current equipment conditions to establish precise instructions for repairs or replacements. Moreover, prescriptive maintenance has the capability to recommend optimal automatic palliative or curative healing actions.

The approach to preventive maintenance and corrective diagnosis involves a multi-tiered examination of its components. Firstly, (1) the technical or physical layer includes the machines and their various components, such as the application server machine, the database, and elements such as RAM, SWAP, processor and disk usage, network identities, and connectors. Secondly, (2) the application layer focuses on the key components of the Copilote application server, the database server, and client workstations, including heap, cache code, resource consumption, launch configuration controls, and dump files. (3) The functional layer on the other side, evaluates the processes executed by the Copilote application to ensure efficient electronic data network exchanges, optimal latency, accurate statistics execution, and more. Finally, (4) the business layer assesses the control and comparison of critical business parameters. To address issues in each layer, dedicated teams may be assigned to provide specialized support. The layered maintenance schema is illustrated in Figure 1.3.

Infologic adopts end-to-end client-server architecture to offer comprehensive maintenance support and assistance to its clients. Figure 1.4 highlights the key actors and tools involved in the execution of both corrective and preventive maintenance procedures. An internal organizational platform assumes responsibility for handling maintenance tickets, primarily concerning corrective maintenance operations. On the other hand, a supervisor and monitoring system oversees preventive maintenance actions and automatically generates based on performance metric data preventive maintenance tickets in the former platform. The two approaches exhibit some variations. In the ensuing discussion, we will examine each of the maintenance support services provided, mentioning the underlying motivations, contextual factors, and operational methodology.

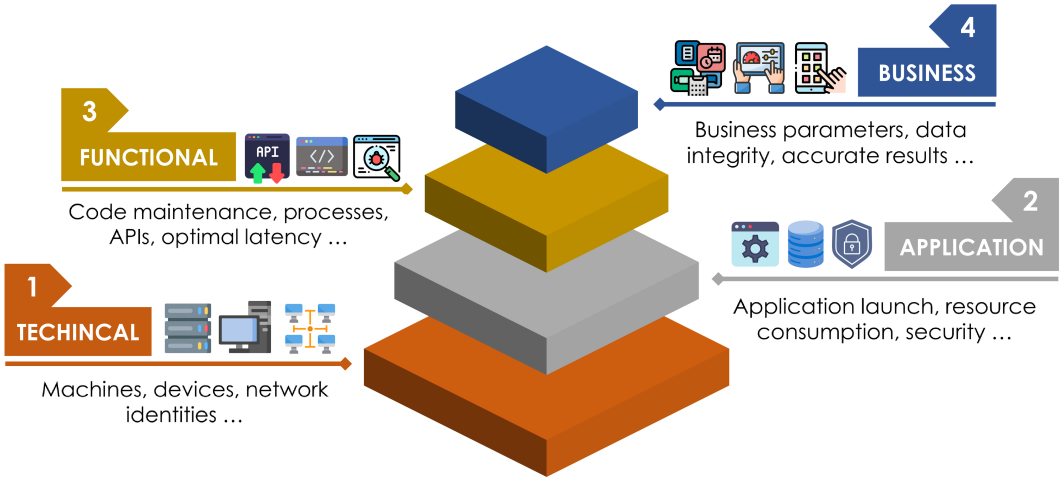


Figure 1.3: Maintenance layers or stratas of the ERP Copilote.

Corrective Maintenance. When a customer encounters a blocking issue requiring maintenance support, such as bug correction or assistance with an urgent task, they may initiate an incident report. In order to optimize the incident reporting procedure and facilitate the creation of maintenance tickets, Infologic equips its customers with various tools and methods of communication to convey their problems. One such tool is (1) the telephone exchange when customers are able to dial a dedicated Infologic number and ask for support requests, which are then attended to by maintenance on-call engineers who create the maintenance ticket manually. Infologic customers have access to (2) Dint, an integrated system within their instances that allows for the submission of detailed Assistance Requests [ASS] or Anomaly Reports [ANO] from their client stations via an interactive interface. This interface provides description fields to categorize the issue, as well as an attachment space to load related captures and files. (3) Certain privileged customers, as stipulated in their contract, have the option to submit their requests via direct email to the designated support address. The maintenance team then assesses these emails and creates the necessary maintenance tickets.

Triage and Diagnosis. Maintenance tickets include fields for important information, such as the on-call engineer who creates the ticket, creation date, customer details, software version, and, most importantly, the designated service team (referred to as a channel). Furthermore, the maintenance ticket tracks the progress of the diagnosis and any transfers of ticket responsibility between service teams. However, the crucial aspect is the description section, which contains textual content detailing the issue, as well as possible image captures, pasted logs, SQL queries, stack traces, and other relevant information. The description section also includes a comment section reserved for internal Infologic personnel to discuss the problem. This setup offers as well the advantage of associating similar maintenance tickets from previous occurrences, albeit through manual identification. Figure 1.5 depicts the maintenance actors involved in the process of incident management from reporting to resolution. At level 0, the primary focus consists in the initiation of a maintenance call ticket considering each of the available maintenance tools. Level 1 represents an intermediary stage, capable of addressing various support issues effectively (e.g., issues related to inventory management or

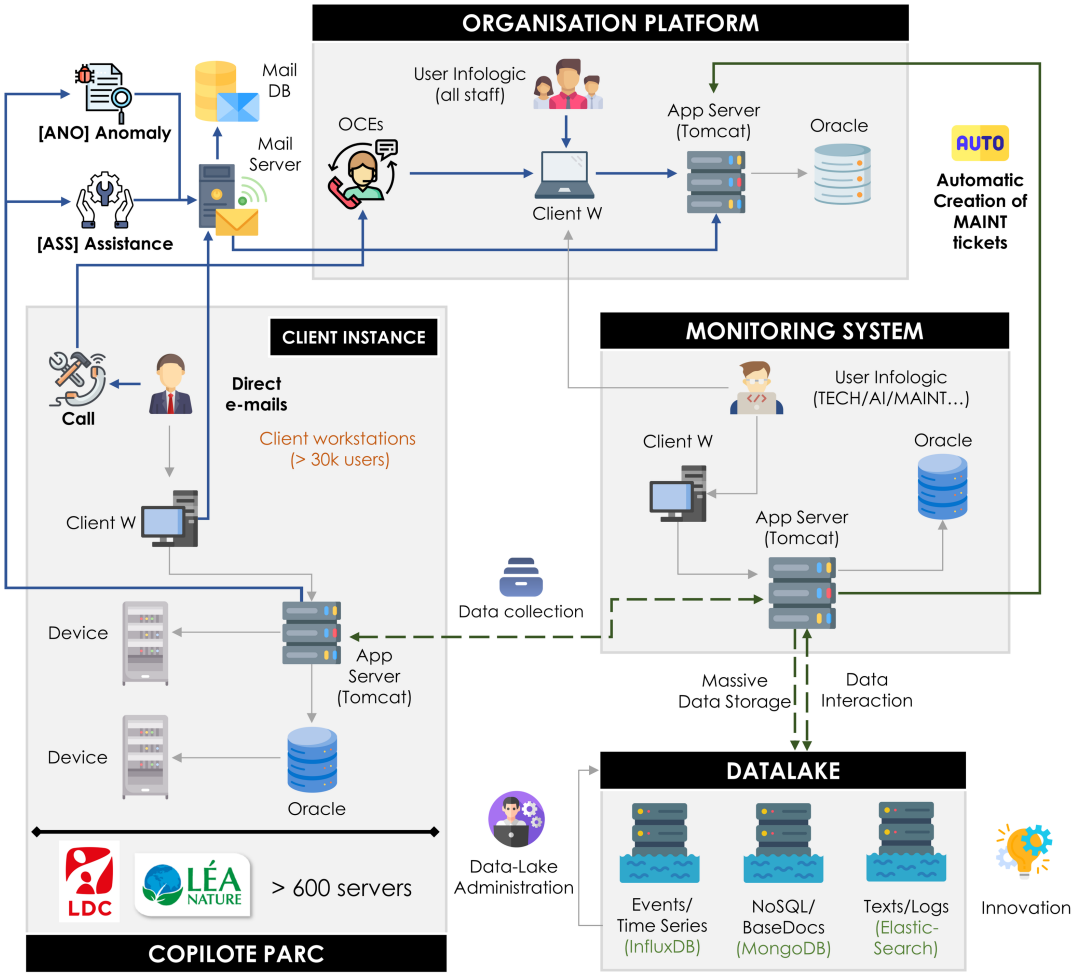


Figure 1.4: Simplified maintenance process at Infologic including key actors and involved tools.

commercial operations). The final level is designated to tackle low-level concerns, such as bugs, infrastructure complications, and system crashes, which require the expertise of specialists. This tri-level classification is based on the complexity and specificity of the problem at hand. When multiple team members from the same group have the capability to address a request, the one with the least number of pending requests is generally selected. This assignment of maintenance calls is, however, limited by two key factors: (1) the complexity of the call, and (2) the choice of the responsible member, taking into account several parameters. In general, blocking and urgent calls have the highest priority, followed by older calls of normal criticality that have not yet been addressed. Nevertheless, this procedure is not standardized, and the complexity of the call cannot be accurately quantified. Furthermore, it is advisable to assign similar calls to the same person.

Preventive Maintenance. Preventive maintenance is managed almost exclusively through a functionally-rich platform that offers secure access to instances, with the ability to configure custom programs to execute control activities and orchestrate data collections. The

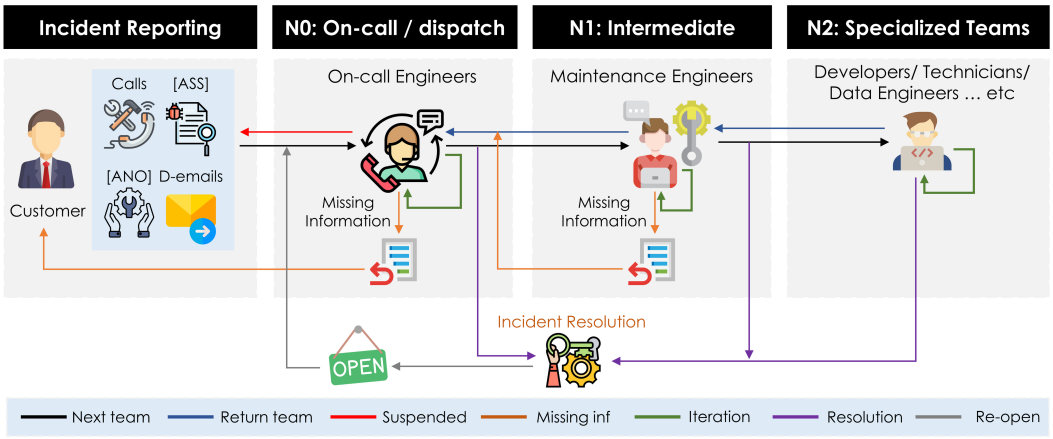


Figure 1.5: Different actors involved in creation, diagnosis, and resolution of maintenance tickets at Infologic.

supervised data is recorded and preserved in two distinct forms to accommodate both short-term operational requirements and long-term data retention needs. The data is stored as (i) aggregated information with a brief retention period on the Oracle instance, serving as the source of operational data, and (ii) its raw form with a prolonged retention period on a datalake. The metric data, meanwhile, is conveniently stored in a time-series database, InfluxDB, enabling efficient querying and analysis (with a perspective of entirely migrating to ClickHouse). Additional data, such as the details of parameterization and logging data, is stored on an object-based database, while the raw text is indexed in ElasticSearch to facilitate text search and retrieval.

Preventive maintenance is carried out through the implementation of alerts that are triggered by expert rules (i.e., conditional maintenance). The alerting system constantly monitors the health of the instances by tracking key performance indicators (KPIs) and other metrics associated with the ERP (physical, application and functional level). Alerts are generated whenever anomalies are detected in the metrics, or when expert rules are violated (e.g., an SQL table size exceeding a pre-determined threshold). Alerts generated by the preventive maintenance management system range from simple alerts concerning a single signal to more complex alerts based on multiple signals or other alerts. These alerts can also be either instantaneous, representing a momentary event, or constantly accumulating, indicating an ongoing issue such as an increase in disk space utilization. Upon the detection of incidents, maintenance call tickets are automatically created and dispatched to the N2 maintenance specialists. For example, the identification of an anomaly in the Oracle console results in the automatic generation of a maintenance call, directed to the Database team. These calls are distinctly marked with the label "Preventive Maintenance" to differentiate them from corrective maintenance calls.

1.1.2 Pain Points and Limitations

The existing maintenance workflow, as explained above, offers a diverse spectrum of services to satisfy the customers' requirements pertaining to their functional, business, and technical

issues. However, it heavily depends on repetitive tasks, and iterative assignments, and is deprived of automated processing. As a result, there is a substantial loss of time and a lack of proficiency in the management of maintenance costs. Herein, we have compiled a list of the principal challenges that we have identified, which are either associated with (i) human and organizational aspects, (ii) traditional practices in maintenance, and (iii) data normalization (particularly, problems of velocity, volume, and variety of data).

Lack of standardized and automated maintenance routines. Standardizing maintenance processes is essential to categorize the process in a phased manner from reporting through diagnosis and mitigation. Optimizing the process in terms of time, resources, and cost-effectiveness, and potentially automating it, provides a unified approach to handling customer support and maintenance requests. At Infologic, existing reporting tools can make the reporting procedure very costly. Furthermore, customers may not always provide all the necessary information for troubleshooting, or may not precisely describe the issue, resulting in a lengthy exchange in order to create the maintenance ticket. Therefore, optimizing these processes is more than essential to ensure a smooth and efficient maintenance process. Moreover, established engineering practices often prioritize adaptive measures and meticulous analysis of individual cases through the examination of bug reproduction steps and exhaustive logs. This approach, however, prove to be ineffective or even unfeasible when dealing with large-scale service scenarios, e.g., in the case of Infologic.

Inefficient incident triage and classification. Incidents must be quickly qualified and assigned to responsible teams and classified into categories in order to determine their priority and allocate the appropriate resources. If this process is not well-defined, incidents may be delayed, leading to a prolonged resolution time. At Infologic, there is a significant demand for an automatic mechanism for routing maintenance tickets to the appropriate team based on the information provided in the maintenance ticket. Additionally, there is no established process for prioritizing calls based on specified criteria. Our analysis of historical maintenance calls also revealed the presence of recurring similar issues that could be grouped together into buckets and then addressed quickly relying on the historical data as a reference. The ideal routing process should factor in the individual expertise for efficient resolution, thus reducing the time to diagnose/mitigate.

Ineffective root cause analysis and incident correlation. If the root cause analysis process is not performed effectively, the same problems may reoccur. On occasion, in response to urgent maintenance requests, the maintenance team needs to provide temporary remedies and promptly close the associated ticket, without affording sufficient time to deliberate on the root cause and implement a lasting resolution. Consequently, the issue is liable to reappear, either with the same or with other customers. Furthermore, it is important to recognize that some issues can lead to further complications if there are interdependencies between them. As such, we should not be confined to addressing solely the reported problem, but it is also important to identify any correlated issues that may be interconnected or caused by the primary problem.

Inconsistent incident documentation In general, inconsistent documentation can lead to duplication of efforts, particularly when incidents are handed off between multiple teams across many levels. Infologic is faced with this problem on a frequent basis, which can be attributed to the lack of proper resolution tracking and ineffective communication practices between different teams. Our analysis of the maintenance call lifecycle has exposed the

prevalence of returns to previous support levels and frequent chain switching, which while may be required in some cases, often results in redundant efforts.

Dealing with diversity and volume challenges in maintenance data. Incident data may come from various internal and external sources, including phone calls, emails, as well as log records and performance metrics. The data structures across these sources may have different structures. These structural variances and large volumes among the diverse sources of incident data present a significant challenge in unifying them into a coherent and robust data model. Moreover, current alert-based systems are stressed when dealing with high volume and velocity of data. Such systems are plagued with several limitations, including alert flooding, which can result in critical alerts being overlooked.

1.1.3 Towards A Seamless Automated Solution: An AIOps Framework

As stated in 1.1.2, the challenges encountered by Infologic have sparked our interest in replacing multiple and manual conventional maintenance routines with a unified, intelligent, and standardized approach to conduct maintenance incidents from creation through diagnosis to resolution. The objective is to automate as many maintenance tasks as possible, with a view to optimize the reporting time, diagnosis and triage time, and resolution time. Such a protocol should not only facilitate the resolution of the concerned incident, but also serve to document the maintenance context within Infologic. This approach is in line with the concept of AIOps, which stands for AI for Operating Systems. AIOps was first coined in 2017 by Gartner [242] to address the challenges faced by DevOps by incorporating AI. AIOps is an extension of the earlier concept of ITOA (IT Operations Analytics) and has been redefined by Gartner as Artificial Intelligence for Operations systems based on public opinion and the growing popularity of AI in various fields [268]. AIOps leverages big data, machine learning, and analytics technologies to intelligently automate a wide range of IT and maintenance operations and to accelerate the identification and resolution of IT issues and outages [242, 76]. By ingesting and analyzing massive volumes of data generated by IT systems, AIOps learns to autonomously detect, diagnose, and even remediate IT service issues in real-time, which can improve service quality, customer satisfaction, engineering productivity, and reduce operational costs [242, 64, 76]. According to [268, 242], a prototypical AIOps system involves six fundamental abilities that engender diverse tasks, duly reflected in the associated capabilities, as demonstrated in Figure 1.6.

1. **Perception.** This capability centers on the ability to gather heterogeneous data types, including logs, metrics, network traffic data, and more, from a multitude of sources, such as networks, infrastructure, and applications. It is essential that the ingestion process accommodates both real-time streaming and historical data analysis. Additionally, powerful data visualization, querying, and indexing mechanisms are also necessary elements.

2. **Prevention.** This process entails proactive identification of potential failures and forecasting high-severity outages in the system using data-driven approaches. Additionally, accurately predicting the remaining useful lifetime of hardware and application components is a vital component of this process. Prevention is paramount to maintaining a healthy and robust system, which is why implementing an automated system capable

Figure 1.6: Fundamental abilities of an AIOps platform. Figure adapted from [268, 242]

of continuous system health monitoring and timely alerting administrators of potential issues is crucial.

3. **Detection.** If errors occur, it is imperative for the system to detect the associated anomalies or symptomes. This is achieved by analyzing vast amounts of perceptual and historical data to identify abnormal content in either the time domain or spatial domain, or both. This process includes discovering abnormal patterns in data and detecting flexible abnormal conditions that exceed static thresholds, while minimizing noise in data, such as false alarms or redundant events.

4. **Location.** The objective of this process is to identify and analyze potential root causes responsible for the underlying incidents by conducting a causality and correlation study. This study must be contextualized within a unified topology to ensure its accuracy and efficacy. Without the context and constraint of topology, the patterns detected, while valid, may be unhelpful and distracting. By deriving patterns from data within a topology, the number of recurrent and redundant patterns can be reduced, exceptionalities in data can be highlighted, and hidden dependencies can be identified.

5. **Action.** This includes conducting reactive triage on problems and prioritizing incidents once detected or predicted, as well as implementing a series of corrective actions based on the current scenario and past solutions that have already been provided. However, it is important to note that automatic healing actions are executed safely and effectively.

6. **Interaction.** It is referred to as human-computer intelligent interaction. This involves bidirectional interactive analysis between the intelligent models and the expertise of users. For instance, the system can integrate human expertise to enhance its models or similarly leverage model insights to enrich and update a practitioner's background knowledge. Furthermore, this includes facilitating communication and collaboration between different maintenance teams and with customers, promoting efficient information sharing and effective issue escalation.

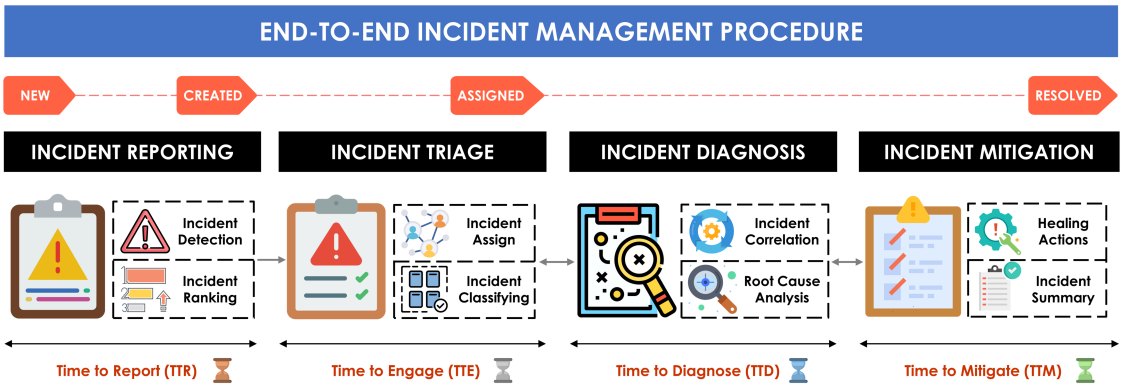


Figure 1.7: Standardized end-to-end procedure proposed for Incident Management.

Drawing on the capabilities of AIOps and our commitment to shift our maintenance system towards a more standardized approach, our first contribution in this thesis entailed a redesign and categorization of the maintenance workflow at Infologic. Our new iterative and sequential workflow assists maintenance tickets through four standardized and clearly defined phases, from initial reporting to final mitigation and postmortem. This approach, which we refer to as the end-to-end incident management procedure, was driven by an industrial need with the aim of minimizing costs and optimizing resources. Figure 1.7 provides a clear visual representation of this categorization. It should be noted that a maintenance ticket typically goes through many sub-phases in the maintenance workflow as illustrated above, but there may be instances where certain phases may not be necessary. In such cases, a phase may be skipped if it is deemed redundant or does not add value to the resolution of the reported issue. For instance, if an incident is assigned to a category that has already been investigated in the past, a root cause analysis phase may not be required.

Incident Reporting. The initiation of a maintenance ticket, also referred to as incident reporting, can be performed in various ways, such as direct calls, emails, maintenance support interfaces, and preventive alerts. Nevertheless, this multiplicity of resources can present a drawback since it necessitates a considerable amount of human resources to manage them. Furthermore, the reporting system can be protracted due to customers failing to provide the requisite information, implying a back-and-forth trajectory between the customer and the maintenance staff for proper qualification of the maintenance call. We propose to simplify the process of creating external incidents through a portal server accessible to all our customers. This solution will guide the process of creating and describing maintenance calls, as well as identifying the necessary fields for triaging, diagnosing, and resolving the reported incidents. Some fields must be filled out in order to submit the request, such as the environment in which the incident is being raised (client, server, version, address, client workstation, etc), while other fields can be automatically reported through our background processes. The customer will be able to provide a detailed description of the incident, such as a step-by-step procedure to reproduce the error if it is a bug. By imposing certain constraints on the qualification of a maintenance call, we can significantly reduce the number of exchanges with the customer and prevent any foreseeable interaction. Meanwhile, the monitoring system can detect or predict incidents by periodically monitoring the runtime information of service systems, such

as software logs, performance metrics, and machine/service-level events. When creating a well-qualified maintenance ticket in the maintenance dashboard, one initial need consists in providing an appropriate order for these calls. Chronological ordering may appear to be an intuitive and trivial choice, as an attempt to tackle long-standing calls is typically a priority. Nevertheless, it is also crucial to incorporate the constraint of incident severity. At Infologic we assign each maintenance ticket a priority, denoting its level of urgency (normal, urgent, or blocking). Typically, it is the customer who designates this status, indicating, for example, the unavailability of a crucial service. But when multiple calls share the same priority level, how can one determine which call to address first? What metric should be used to quantify the priority score? These considerations highlight the necessity of a comprehensive, ticket-based approach that considers all pertinent fields, such as ticket date, description, incident reproducibility, functional/business criticality, customer profile, the occurrence of similar problems with other customers, and more, to establish a normalized severity level.

Incident Triage. Once the maintenance call is created and assessed in terms of criticality, it must be resolved. A first intuition relies on ascertaining the nature of the incident, whether it relates to a technical issue such as network problems, security breaches or connectivity problems with servers/printers, a business-related problem, or software outages (e.g., crashes, a functionality with erroneous results, etc). To achieve a comprehensive understanding of the reported incident, it is necessary to precisely and explicitly characterize it in a well-defined context, which involves narrowing the scope of the call by assigning it to a specialized team. This process is known as Incident Triage but more specifically as Incident Assignment [329, 59]. Automatic Incident Triage has garnered notable attention in the software engineering community. This process involves analyzing the information provided in incident reports and relying on advanced natural language processing techniques to effectively direct maintenance tickets to the appropriate maintenance team. It is a supervised classification task in which a model is trained to discern the salient features of an incident, along with any important characteristics that may be available, and map them to a corresponding class that designates a specialized service. One could even take the incident triage process to the next level by automatically recommending the most suitable individual to tackle the incident. This is typically accomplished by leveraging the prior experience gained from comparable calls previously handled by the same person. However, in order to fine-tune the incident triage process even further, it is imperative to address another critical task, namely incident classification or incident similarity search. In accordance with our objective, it is also required to categorize incidents assigned to a particular team based on their respective topics, whenever feasible. For instance, the technical service team at Infologic may be called upon to rectify incidents related to resource saturation (e.g. hard disk capacity, number of active threads), processes that exert a considerable demand on the CPU and SWAP, difficulties with establishing the connection between Copilote and factory equipment, and security vulnerabilities, etc. To efficiently manage these issues, the technical team should appoint knowledgeable staff to each topic. Thus, it would be ideal to establish an incident triage procedure that carries on the process when a call is assigned to the responsible team, with the identification of the appropriate topic serving as the initial step in the procedure.

Given that, an incident is categorized into a topic (e.g., the incident is assigned to the database team, and it is related to an SQL query performance, where significantly high latency is experienced). The team designated to handle such matters is called upon to

mitigate the issue. However, it is highly probable that this type of incident is not an isolated event, and it has been reported on prior occasions. In some instances, the issue may not be limited to a single customer, but could potentially impact multiple customers who have encountered the same problem. Thus, further investigation of the issue would be redundant if a solution has already been implemented. However, in the presence of a substantial volume of historical incidents, searching for identical incidents can be challenging. As a result, (1) similar maintenance incidents known as near-duplicates should be grouped into the same category that refers to a specific topology, and (2) a quick, and precise way to search for historical similar incidents already resolved should be also provided.

Incident Diagnosis. In large-scale systems like COPILOTE, where numerous components and functionalities are intricately connected, a single incident can potentially initiate a chain reaction of other incidents, which may manifest immediately or gradually reveal themselves over time. While unit tests are designed to capture such cases after remedying functional errors or software bugs, detecting incidents that arise during on-the-fly solutions or over time presents a significant challenge. Thus, a causality and correlation study must be performed to localize the origin of the problem and its potential ramifications.

Incident Mitigation. Obviously, the final stage in incident management lies in restoring the affected service to its normal, functioning state. If the triage and diagnostic stage has been performed accurately, then the incident resolution process will be carried out optimally. Nevertheless, certain problems may have obvious solutions that can be automated, such as restarting instances for memory leaks. Lastly, it is important to automatically generate an incident report during postmortem treatment, known as the incident summary, which can be referred to during the incident diagnosis phase for future issues of a similar nature.

1.2 Challenges Addressed in this Thesis

Several companies have started dispensing AIOps tools as commodities within the last few years, while a number of technology giants have adopted an AIOps algorithmic viewpoint to proficiently maintain their on-premises or cloud computing infrastructures and manage incidents [76, 177, 172, 178, 244, 184, 64], thereby inducing the academic field to evolve and deliver more ingenious and innovative solutions. In actuality, the notion of utilizing AI to refine IT and maintenance operations, despite its recent formalization and introduction as a research field, is not entirely novel [232, 40]. Beginning in the mid-1990s, some research work explored software defects in source code by employing statistical models founded on source code metrics [147, 66, 46]. Since the start of the new decade, various techniques have been proposed to tackle online software [327, 241, 333] and hardware [315, 174, 337] failure prediction and anomaly detection [67, 297, 225]. Multiple other domains of AIOps, such as event correlation [306, 199, 187], bug triage [319, 309, 320, 59], and root cause analysis [149, 192, 136, 175], have also witnessed significant contributions over the last two decades. In fact, The reliability, efficiency, and maintainability of hardware and software systems have always been a prominent research focus. However, rather than a steady progression of research contributions, we have recently witnessed an increased interest in this field. This phenomenon is driven by two main factors: firstly, the remarkable advances achieved in the field of machine and deep learning, and secondly, the shift of numerous IT organizations from product delivery to service release, coupled with the transition from traditional to dynamic infrastructures.

Despite the promising benefits that AIOps offers, it nonetheless poses challenges from both technical and non-technical perspectives.

Challenge 1. The field of AIOps lacks a unified terminology, making it challenging to discover and compare contributions from various specialized disciplines. Desiderata and requirements to construct an effective AIOps model, as well as metrics for comparison, must also be outlined and contextualized for real-world scenarios.

Firstly, the field of AIOps remains predominantly federated and unstructured as a research topic. Rather than a homogeneous and well-defined area of study, it encompasses a diverse array of contributions derived from various specialized disciplines. Given its novelty and cross-disciplinary nature, AIOps contributions are widely dispersed, lacking standardized taxonomic conventions for data management, target, and focus areas, implementation details, requirements, and capabilities. As such, discovering and comparing these contributions has proven to be a challenging and infeasible [232]. The lack of a unified terminology results in the absence of guidelines and a clear roadmap for addressing the gaps in the state-of-the-art within AIOps. Although various data-driven approaches may be attributed to the AIOps research area, findings from disparate domains, such as machine learning, may not necessarily apply to software analytics domains like AIOps. For instance, natural language processing models commonly used in machine learning may produce spurious outcomes when applied to software engineering-related data according to Menzies [210] and Ray et al. [245]. Dang et al. [76] highlight several challenges unique to AIOps, which necessitate a deeper comprehension of the overall problem space, including the business value, data, models, as well as system and process integration considerations. Therefore, it is important to determine within the purview of AIOps, the optimal taxonomy that must be entirely driven by an industrial need necessitating domain expertise in both IT operations and AI techniques.

It is also highly important to outline the requirements (desiderata) to construct effective AIOps models, including maintainability, interpretability, and scalability, among others. Additionally, one must also inquire about the metrics that should be employed to compare AIOps methods that belong to the same category, such as anomaly detection or root cause analysis. Metrics based on machine learning, such as contingency metrics, do not suffice or reflect the real accuracy of models when deployed in actual scenarios and hence require contextual adaptation. For instance, El-Sayed et al. [98] have proposed the novel *just-in-time* metric to evaluate their prediction approach on job failures, which utilizes a time window to mark valid predictions. Multiple other factors and peculiarities should be taken into account (e.g., human involvement in the loop.)

Challenge 2. The unique data requirements of AIOps models, coupled with the challenge of obtaining high-quality labeled data, make building accurate AIOps solutions a complex task.

Secondly, AIOps models require a unique set of data that differs from what is typically used for general machine learning models. Despite the fact that major cloud services collect terabytes and even petabytes of telemetry data every day/month, the data quality and quantity available today still do not meet the needs of AIOps solutions. According to [76, 172, 63], data from diverse sources can assume disparate formats and structures, which complicates the

task of normalization and cleaning. This data can be either unstructured or semi-structured, such as logs, execution traces, source code, hierarchical and graph data such as heap memory dumps, and network traffic, which requires distinct analytical techniques. Additionally, AIOps models that heavily depend on supervised machine learning algorithms for anomaly detection or failure prediction necessitate labeled data for model training. However, data can often be contaminated with noise or missing values, and labeled data may not be easily obtainable, making it challenging to build accurate and robust AIOps models.

Challenge 3. AIOps has predominantly focused on developing predictive models for anomaly detection and failure prediction, despite the challenges posed by data quality and complexity. In contrast, descriptive models, which rely on massive data mining techniques to extract informative and exceptional patterns, are less adopted but can be valuable and highly effective in dealing with these challenges.

Thirdly, in numerous AIOps settings, constructing supervised machine learning models for AIOps poses challenges due to the quality of the data. These challenges include the absence of clear ground truth labels or the requirement of manual efforts to obtain high-quality ones, extremely imbalanced datasets, too little amount of data, a high degree of noise, etc. Consequently, unsupervised or semi-supervised machine learning models are the only feasible options. Indeed, it is difficult to obtain enough labels to learn "what is abnormal" about a service, primarily because every service behavior is continually evolving with the change of customer needs and underlying infrastructure changes. The difficulty of creating high-quality unsupervised models also stems from the complexity of dependencies and relationships among components and services. In addition, the need for frequent model updates and online learning presents significant challenges to DevOps/MLOps practices, especially when it comes to intricate feature engineering efforts. Another challenge is to ensure that the behavior of the model during the training phase is consistent with its performance in the testing and production phases. Traditional metrics used to assess models are susceptible to the contamination zone phenomenon [102], which may lead to erroneous assessments. Indeed, Fourure et al. [102], highlight that by parameterizing the proportion of data between training and testing sets, the F1-score of anomaly detection models can be artificially increased.

Despite these challenges facing predictive models, AIOps has predominantly focused on the development of predictive models for anomaly detection and failure prediction. However, there is a lesser-known yet highly useful approach, which is the application of descriptive models. These models rely on massive data mining techniques to extract informative patterns from data that can aid in detecting, diagnosing, and resolving issues. Descriptive models are particularly advantageous in dealing with the challenges of data diversity, complexity, and quality, which makes them a valuable asset in cases such as deduplication failures and complex dependencies. Consequently, it is essential to direct focus toward enhancing descriptive models in conjunction with predictive models.

Challenge 4. The use of complex machine learning models, which are often opaque and lack transparency, poses a significant challenge in the adoption of AIOps solutions by industry practitioners who require a full understanding of maintenance processes.

Fourthly, in general, the efficacy of machine learning models is directly proportional to their intricacy, i.e., the most accurate models are opaque, known as black box models as they do not convey any explanation of the internal process of making decisions or predictive capabilities [115, 217]. This lack of transparency significantly hinders their adoption by industry practitioners who require transparency and a full understanding of maintenance processes and tool behavior. While it is highly valuable that we leverage robust models that optimize maintenance costs and automate repetitive tasks, doing so necessitates the sacrifice of transparency and full comprehension of the underlying maintenance processes and the rationale behind key decisions. Several recent studies in the field of AIOps argue that interpretable models, even those with lower performance, are preferred when high-performing models lack interpretability [202]. Rijal et al. [256] emphasize that there is widespread doubt about the efficiency of machine learning models in the industry. This is driven by the fact that AIOps solutions primarily rely on learning from experience to predict the future and identify trends from vast quantities of data. However, IT professionals who have been in the field for some time are questioning the effectiveness of these models, even after recognizing the need for digital transformation [202]. Therefore, it appears that businesses need additional time to build confidence in the soundness and dependability of recommendations from AIOps.

Successfully automating incident management processes is a significant challenge that requires gaining practitioners' trust by providing them with explanations of model decisions. The popularity of black box prediction models, combined with the crucial need for transparency in many decision-making processes, has generated an unprecedented interest in eXplainable Artificial Intelligence (XAI).

Challenge 5. An existing limitation in AIOps includes overlooking the importance of performance evaluation when comparing and evaluating incident management models.

Finally, it is imperative in AIOps environments to not only focus on the efficiency of the model but also on its performance. While optimizing TTx times (reporting, triage, diagnosis, and mitigation) as highlighted in Figure 1.7 is a crucial factor for implementing a successful automated incident management procedure, performance study is often overlooked when comparing different models that tackle the same research field. Many surveys, literature reviews, experience reports, or benchmark studies [232, 122, 260, 78, 96] primarily rely on efficiency metrics such as contingency table metrics (e.g., accuracy, precision, recall, ROC, or AUC) or regression metrics (e.g., MSE, RMSE, MAE, etc) to determine the superiority of one method over the other. However, in practical scenarios, a model that takes less time to execute but with a capability of anomaly detection of 90% of F-Score may be preferred over a model that yields 95% of F-score but takes a longer time to run. Particularly, the authors in [64] introduce a new metric, Time to Broadcast (TTB), which measures the time it takes to broadcast a failure to all impacted services.

1.3 Key Research Areas

In response to the aforementioned challenges, we first structured the body of knowledge around AIOps. Our primary contributions addressed the necessary requirements for building AIOps models, including challenges related to data, modeling, interpretability, and performance optimization. While some of our solutions were designed to address specific use cases,

Figure 1.8: Key research areas addressed in this thesis.

they can be applied to a wide range of use cases. Overall, this thesis stands out for its ability to expose, analyze, internalize, and contribute to several diverse research areas that converge data mining, machine learning, and software engineering. During the course of this thesis as shown in Figure 1.8, we focused our attention on four interlinked domains: (1) Artificial Intelligence for Operating Systems (AIOps), which is the primary focus of this work, (2) Subgroup Discovery (SD) and its generalization Exceptional Model Mining (EMM), which we employed to address data complexity and diversity challenges in AIOps, (3) eXplainable Artificial Intelligence (XAI), which we used to increase the transparency and trustworthiness of our models, and finally (4) Locality Sensitive Hashing (LSH), which we utilized for efficient and fast similarity search to cope with performance challenges. These domains were carefully selected based on their potential to improve our understanding of AIOps and to address the challenges associated with building effective and interpretable models. As such, we believe that this thesis represents a significant step in our academic and professional journey.

1.3.1 Subgroup Discovery and Exceptional Model Mining

We tackled the challenges related to data, model implementations, and interpretability using Subgroup Discovery (SD) [304, 15, 233], a promising data mining technique also known as Supervised Rule Discovery. This approach aims to identify interesting and interpretable patterns in large datasets with respect to a specific target problem. The primary objective of Subgroup Discovery is to retrieve interpretable links between different characteristics (i.e., descriptive variables) and the property of those individuals we are interested in. Referring to Figure 1.9, which illustrates a practical example showcasing the utility of this approach. The dataset used for this example contains information about servers, including their descriptive attributes and whether they experienced an out-of-memory crash during a week. One inter-

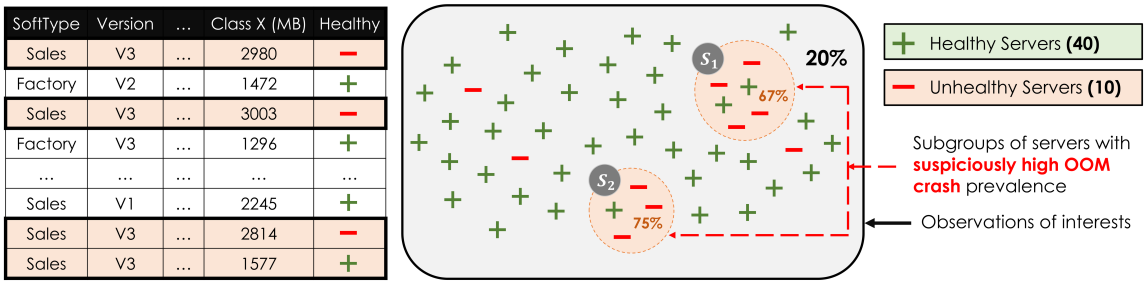


Figure 1.9: Toy example demonstrating the working of Subgroup Discovery.

esting analysis is to identify subgroups of servers whose probability of crashing is significantly higher than the average. The dataset is imbalanced, with only 20% of servers shown to crash. However, two well-characterized subgroups, S_1 and S_2, have a higher prevalence of crashing than other subgroups. For instance, subgroup S_2 is characterized by the rule: `SoftType` = `Sales` and `Version` = `V3`, and the size of a particular class `X` in the heap dump exceeds 2000 MB, which is highly discriminatory. These subgroups are exceptional and can be useful in identifying the root cause of the crashes.

Subgroup Discovery is also highly adept at handling the complexity, diversity, and large volumes of data that arise in the AIOps context, which may contain noise, poor quality, and non-homogeneity. Furthermore, many subgroup discovery approaches have been proposed in the data mining community to handle structured, semi-structured, and unstructured data (e.g., sequential data [112, 206, 124], trees [3] and graphs [86, 288, 16, 144], etc). However, this technique has yet to be explicitly or implicitly applied in the AIOps domain or its derived subdomains. Subgroup Discovery can be employed not only to model input data but also to establish associations with single or many complex target concepts. This methodology can be leveraged, for instance, in SQL queries to discern the context wherein a well-defined set of queries display significant latency time [247]. The input data pattern can be mapped to a formal representation of SQL queries, while the target is expressed numerically in seconds. However, the target can also be complex and include many different attributes. For example, one might be interested in SQL queries that are relatively slow but also return fewer lines. Subgroup Discovery can handle such complex targets, allowing for the discovery of specific patterns in the data that would be difficult to uncover through manual analysis.

Furthermore, we believe that Subgroup Discovery is particularly useful as it expands the scope of common problems in AIOps beyond their typical predictive purview to a new axis focused on identifying exceptional or regular data patterns within a well-defined context. This approach can help solve challenging problems associated with anomaly detection, fault localization, event correlation, and root cause analysis. As noted by Prasad and Rich [242], the identification of relevant and actionable patterns by AIOps relies on contextualizing the data within an appropriate contextual framework. Without this contextual constraint, the patterns identified, while valid, may prove unhelpful and distracting. By contextualizing the data, Subgroup Discovery can reduce the number of patterns, establishes their relevancy, and reveal hidden dependencies.

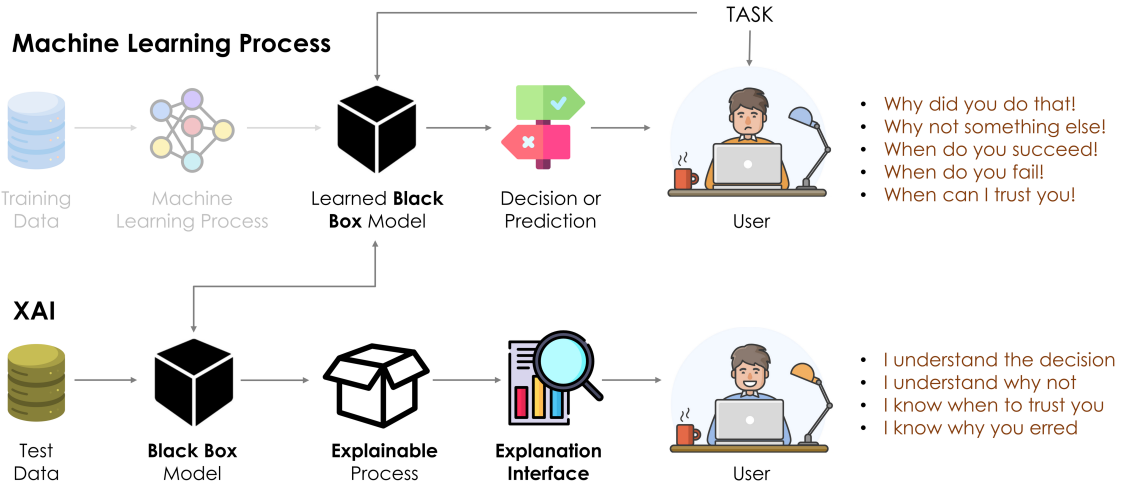


Figure 1.10: Concept of Explainable AI (XAI). New interpretable machine-learning processes try to have the ability to explain the rationale of black box models, characterize their strengths and weaknesses, and convey an understanding of how they will behave in the future.

1.3.2 Explainable Artificial Intelligence

When deploying predictive models in AIOps for incident management, it is important to ensure that these models are not only accurate and effective but also interpretable. Consequently, it is no longer a matter of choice, but rather a necessity to consider interpretability when building AIOps models. To address this, we conducted an extensive analysis of the explainable artificial intelligence (XAI) domain. The primary objective of XAI is to establish a mechanism that offers understandable and transparent explanations for the decisions and actions of black box models. This is achieved through a range of techniques, including rule extraction [255, 85, 20], feature importance [265, 198, 254], and model inspection and visualization [154, 110]. Furthermore, XAI aims to detect any biases that may exist in the training process and identify areas for improvement. Figure 1.10 illustrates a straightforward schema outlining the process of interpreting black box models. Because of their particularities, the interpretation of AIOps models poses unique challenges due to their distinctive characteristics. According to [202], interpretations of these models must meet the criteria of internal and external consistency. Furthermore, interpretation must also follow the principle of time consistency when updating or explaining future predictions.

1.3.3 Locality Sensitive Hashing

As mentioned earlier, one of the significant challenges in adopting AIOps is to optimize performance while implementing multiple tasks associated with the incident management procedure. For instance, the increasing volume and complexity of data generated by these systems have made it difficult to quickly locate and search for recurrent and similar data patterns that often induce anomaly detection and root cause analysis. While it is generally easy to identify and prioritize extreme problems, ranking and assigning some issues can be challenging. In fact, different incidents may imply a single root cause. Finding such data

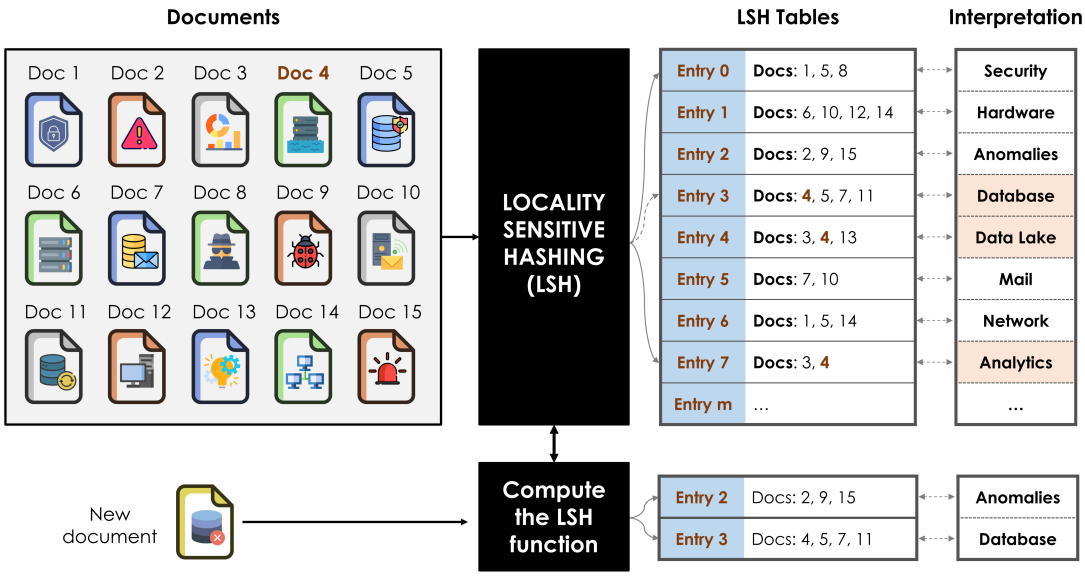


Figure 1.11: Overall view of LSH applied to maintenance documents.

patterns in high-dimensional and large datasets is a challenging task that cannot be effectively tackled with exhaustive search algorithms. We proposed to leverage Locality-Sensitive Hashing (LSH) [292], a highly useful and widely used hashing technique for Approximate Nearest Neighbors (ANN) search that has been employed by Google for indexing and searching similar documents [259]. While this technique has been successfully applied in various fields, including document retrieval [56, 324, 311], security and privacy approaches [22], and biological studies [9], its potential in the context of AIOps, specifically for incident management, has not been fully realized. This technique, however, can provide significant benefits, particularly with respect to near-duplicate detection. In essence, Locality-Sensitive Hashing maps high-dimensional data points to low-dimensional hash codes, such that similar points are mapped to the same or nearby hash codes with a high degree of probability. This enables efficient nearest neighbor search in the low-dimensional hash code space, which can be faster than direct comparison in the high-dimensional data space. Nonetheless, the application of LSH is not straightforward, as it requires designing suitable probabilistic schemes with theoretical guarantees that match the target problem.

The diagram shown in Figure 1.11 provides a high-level overview of how LSH implementation can be applied to maintenance reports. The input to the system is a set of maintenance reports presented as a collection of documents that describe various problems, while the output is a set of tables (only one is shown for simplicity) that map the documents to particular categories. Each entry in the table corresponds to a specific group of related documents, such as reports that pertain to security, database, or hardware issues. The documents in the same entry are expected to have a semantic similarity value of at least a certain threshold value among themselves, whereas documents in different entries are expected to have a lower similarity value.

1.4 Contributions and Prototypes

Following an introduction to the challenges addressed by this thesis and the main research areas that has been explored throughout this journey, we elaborate now on our contributions.

1.4.1 A Comprehensive Literature Review of AIOps-based Solutions for Incident Management Procedure

Our primary contribution [250] lies in providing a systematic approach to the body of knowledge on AIOps, specifically in the context of intelligent incident management procedures. We established a unified and standardized terminology to define the related terms adopted in the state of the art, as well as a comprehensive and objective-based taxonomy that captures the most relevant research in pioneering conferences and journals of machine learning, data mining, and software engineering. Our focus has been on analyzing data-driven methods within specialized disciplines and differentiating them by introducing a comprehensive taxonomy. This taxonomy is designed to closely align with the specific needs of both industry and research, ensuring that researchers and practitioners can derive maximum benefits from our work and facilitate the discovery, implementation, and comparison of these methods.
Prototype. As part of our efforts to make this research accessible, we have created a repository that organizes all reviewed work into various categories, provides available codes and datasets, and publishes them for public access at: `https://github.com/RemilYoucef/AIOps-complete-review`

1.4.2 Identifying Suspicious Query Fragments in Large SQL Workloads using Subgroup Discovery

In our research [247], we have tackled a compelling issue related to SQL workload analysis. This task is essential for database administrators (DBAs) who aim to identify patterns that highlight database schema issues. The problem we address falls under both the incident detection and incident diagnosis phases. The motivation for this issue stems from a genuine industrial need. DBAs can easily pinpoint queries that are repeatedly causing performance issues, but it is a challenge to automatically identify subsets of queries that share common properties and at the same time foster a target measure such as execution time, fewer lines returned, and concurrency issues. While there is no ground truth about suspicious queries causing these issues, we believe that this problem is a Subgroup Discovery instance. We aim to retrieve a well-contextualized subset of queries defined with interpretable patterns that demonstrate exceptional behavior related to performance degradation, alerting signals, or execution issues, among other queries.

Various methods have been proposed for specific tasks on SQL workloads. However, many of these approaches rely on clustering-based methods that do not offer a practical means of identifying subsets of data that specifically discriminate a property of interest. While several major commercial database systems have developed tools to automate this task, they remain specific and non-generic tools, limited to certain features. Hence, we proposed a Subgroup Discovery interactive model that retrieves a well-contextualized subset of queries with common properties that affect specific target concepts according to the user interest, such as execution time and concurrency issues. Our approach involved developing a pattern

language, integrating diverse interestingness measures, and providing exact and heuristic algorithms to identify subgroups of interest. We conducted experiments on a large SQL workload, to pinpoint query properties that are correlated with performance degradation.

Prototype. The data and source code used in this study are publicly available at: `https://github.com/RemilYoucef/sd-4sql`

1.4.3 Generating Understandable Summaries of Black Box Incident Triage Model using Subgroup Discovery

The work presented in [248] tackles a two-fold problem that involves designing an effective and accurate incident triage model aiming to assign incidents to the right maintenance service team within our company to successfully mitigate anomalies quickly and interpreting the decisions of this model to increase trust and have a supportive interpretation. The incident triage process is particularly challenging due to various concerns. Firstly, incidents come in different forms, including predicted incidents and incidents directly reported by the end-user, but often poorly described and contextualized. Secondly, the distribution of incidents across different services is extremely imbalanced, with some services having thousands of incidents while others have rarely been called upon. On the other hand, accurate predictive models proposed for incident triage, often rely on opaque models, which are commonly referred to as Black Boxes. Such models do not provide any explanation of their output, which presents a significant challenge in gaining the trust of practitioners. An essential aspect of successfully automating incident triage is to provide practitioners with an explanation of each model outcome. To address this challenge, there is a need for transparent models that can provide practitioners with clear and interpretable explanations of the incident triage process.

Our primary contribution in this work is the proposal of an efficient LSTM black box model with an attention mechanism for incident assignment, which was trained on a dataset consisting of over 170,000 reported incident reports. This hence motivates our main contribution, consisting in an original approach that summarizes local explanations of black box predictions. Indeed, recent developments in explainable AI help in providing global explanations of the model, but also, and most importantly, with local explanations for each model prediction. Unfortunately, providing a human with an understandable explanation for each outcome is not conceivable when dealing with an important number of daily predictions. To address this problem, our original method rooted in Subgroup Discovery proposes to conceptualize the predicted model outcomes into subgroups according to both (1) a common description and (2) the ability to locally mimic the black box model with a white box one in order to provide an interpretable interface to the user enabling her to understand the decision-making process of the model. The proposed solution allows the user to not only interpret the black box outcome for each subgroup but also to understand the nature of the objects that a subgroup contains. The conducted experiments show that the identified subgroups provide interpretable and meaningful explanations that help practitioners understand why a given incident was assigned to a specific service team.

Prototype. The source code used in this study are publicly available at: `https://github.com/RemilYoucef/split-sd4x`

1.4.4 Mining Java Memory Heap Dumps using Subjective Interesting Subgroups with Hierarchical Targets Concepts

In our work [249], we tackled the issue of effectively analyzing and diagnosing a multitude of incidents related to Java memory heap dumps. One of the most prevalent of these issues is the occurrence of `OutOfMemory` Errors resulting from memory leaks. Typically, engineers utilize tools that provide a detailed memory usage breakdown per class in a Java Virtual Machine to pinpoint the cause of memory saturation. It is important to note that these classes are hierarchically organized into packages. By analyzing this histogram, analysts can identify any classes that consume an unusually high amount of memory, suggesting that they may be the underlying cause of memory overload. Analyzing histograms for memory incidents can be a daunting and time-consuming task due to various reasons. Firstly, understanding the normal consumption size for each class requires significant prior experience. Secondly, certain memory incidents are not restricted to a particular class but may impact multiple classes in a package. Identifying the relevant packages and classes from numerous hierarchy levels can be challenging. Lastly, analysts typically analyze vast datasets containing multiple incidents across different servers and scenarios. In such cases, identifying contexts that consistently coincide with memory errors is crucial for pinpointing root causes. Current methods only diagnose memory issues independently and lack the ability to analyze large sets simultaneously for discovering common patterns. Our goal is to develop an approach that addresses these challenges and extracts valuable insights from such incidents.

Once again, we utilize Subgroup Discovery to uncover interesting subgroups within a hierarchical structure of multiple attributes. To address noisy data, we have developed an adaptable pattern syntax and a subjective interestingness metric that selects informative and non-redundant subgroups and also integrates prior knowledge about the data from the user, resulting in unexpected patterns. Additionally, the interestingness model can be updated iteratively as the mining task progresses. Our methodology includes a straightforward yet efficient algorithm to generate subgroup descriptions and to navigate the complex target concept space.

Prototype. The data and source code used in this study are publicly available at: `https://github.com/RemilYoucef/sca-miner`

1.4.5 Enhancing Near-Duplicate Crash Report Retrieval with Deep Locality Sensitive Hash Learning

In [251], we addressed an interesting problem that lies in the effective automation of bucketing operations for near-identical crash reports. This task entails grouping slightly similar and redundant reports into coherent categories to expedite the triage and diagnosis of associated software incidents. To this end, we required an expeditious means of identifying the nearest neighbors. Hence, we needed a fast and efficient method to identify the closest matches for a given crash ticket. This would enable us to determine whether any remedial or preventive actions had been implemented for a similar crash report.

The problem of crash deduplication has long plagued the software development community, but innovative strategies have emerged to tackle this issue. One promising approach is to design highly accurate similarity measures that leverage information retrieval capabilities and graph matching techniques to compare stack traces. These tailored metrics take into

account specific stack traces characteristics and are typically incorporated into clustering algorithms. However, this approach has drawbacks, including the need for extensive similarity calculations and frequent recalculations to maintain stability for clusters. We approached the problem by treating it as a scalable approximate nearest neighbor search challenge within large datasets. To this end, we propose to use Locality-Sensitive Hashing (LSH), which offers sublinear performance and provides theoretical guarantees on similarity search accuracy. However, it can be challenging to derive hash functions satisfying the locality-sensitive property for advanced crash bucketing metrics. Therefore, our contribution is focused on exploring how to employ LSH for this purpose. To consider the most relevant metrics used in the literature, we introduce DeepLSH, a Siamese DNN architecture with an innovative loss function that accurately approximates the locality-sensitivity property. Through our experimental study, we demonstrate the effectiveness and scalability of our approach in quickly retrieving nearly identical crash reports across a dozen similarity metrics on a large real-world dataset.

Prototype. Data and source code are publicly available at: `https://github.com/RemilYoucef/deep-locality-sensitive-hashing`

1.5 Structure of the Thesis

This chapter provides an overview of the thesis context. Firstly, we introduced the industrial scope, which motivates the study of AIOps research area, and discussed its relevance. We then highlighted the various challenges encountered and addressed in relation to the development of a standardized and automated approach for intelligent incident management. Drawing on this constructed background and the identified limitations, the chapter focuses as well on the main research questions that drive the contributions of this thesis. The remainder of this thesis is organized as follows:

- Chapter **2** is dedicated to presenting the current state of the art in AIOps. This chapter serves as a valuable contribution to the field. Our aim is to establish a standardized terminology and a comprehensive taxonomy that effectively organizes the body of knowledge surrounding AIOps. We propose to categorize federated data-driven approaches from diverse research areas and specialized disciplines in accordance with established industry requirements. To achieve this, we first establish clear definitions of the key aspects that define the taxonomy, including data sources, desiderata required for effective incident management, and evaluation metrics. In concluding this chapter, we motivate our contributions, and we discuss its positioning regarding the proposed taxonomy.

- Chapter **3** focuses on **contribution 2**. It begins by introducing the theoretical boundaries of Subgroup Discovery (SD). We highlight the essential building blocks of this framework, including pattern language, interestingness measures, target concepts, and enumeration algorithms. These building blocks are essential for defining and solving the underlying mining tasks. We then explore a practical application by studying a real-world problem of identifying suspicious query fragments in large SQL workloads, with the aim of pinpointing issues related to performance degradation. To this end, we present the raw data, pre-processing strategy, and an informal description of the problem, as well as how we adapt SD to tackle this problem. We evaluate the effectiveness of our approach through

an empirical study that combines quantitative and qualitative analysis, as well as an interactive visualization tool.

- Chapter **4** details **contribution 3**. We begin by presenting our incident assignment model, which is implemented using a range of techniques, from rough data to black box model evaluations. Next, we introduce and formalize a novel problem of summarizing black box outcome explanations using Subgroup Discovery. To address this challenge, we identify interpretable subgroup descriptions along with their outcome explanations, consisting of feature importance. We present an empirical study that demonstrates the effectiveness of our approach in identifying interpretable subgroups with meaningful explanations in the context of incident assignment.

- Chapter **5** is devoted to detail **contribution 4**, starting with an introduction of the raw data consisting in Java memory heap dumps that are organized hierarchically along with their descriptive attributes. We propose a data model that can be used to generically unify this data, and provide an informal description of the problem. Next, we define the problem of retrieving subjectively interesting patterns from this data using Subgroup Discovery. We introduce a pattern language, subjective interestingness measures, an iterative updating procedure of the model, and a mining algorithm to retrieve actionable patterns. Finally, we present the results of an empirical study that evaluates the effectiveness of our approach.

- Chapter **6** is intended to present **contribution 5**. Initially, we introduce the underlying problem of retrieving near-duplicate crash reports by relying on approximate nearest neighbors. Next, we provide an overview of the theoretical boundaries of hashing techniques for ANN search and particularly focus on locality-sensitive hashing. In order to overcome the problem of deriving hash functions for stack trace similarity measures, we define our approach consisting of a deep Siamese neural network architecture with an original objective loss function to learn and provide a family of binary hash functions that perfectly approximate the locality-sensitive property. Finally, we demonstrate through our experimental study the effectiveness and scalability of our approach in yielding near-duplicate crash report under several stack trace-based similarity metrics.

- Chapter **7** serves as the conclusion of this thesis, where the contributions are summarized, and potential avenues for future research are discussed.

Chapter 2

AIOps-based Data-Driven Approaches for Incident Management

In this chapter, we delve into the various aspects of AIOps and analyze its current state. Our primary objective is to establish a foundational knowledge base for AIOps that can be leveraged in future research. This knowledge serves also as a guiding framework for our thesis, wherein we propose our significant contributions. In order to establish a standardized terminology that includes all related concepts, we start by providing clear definitions of these terms associated with AIOps for incident management. We conduct a thorough review of previous research and related efforts within the field of AIOps, which share similar objectives and contribute to the introduction, definition, or exploration of AIOps. Moreover, we present a list of crucial requirements that should be considered when developing AIOps solutions. Furthermore, we propose an efficient taxonomy that categorizes federated data-driven approaches from various research areas and specialized disciplines. This taxonomy is designed based on established incident management phases, data sources, model properties, and requirements.

2.1 Introduction

This chapter serves as an introductory point to the domain of AIOps for incident management procedures. In addition to presenting our contribution, it provides a comprehensive investigation of existing approaches aimed at addressing various tasks within incident management. Additionally, we review other notable contributions that seek to define, structure, or organize the body of knowledge surrounding AIOps, including the works of [76, 242, 64], and more. While this chapter effectively familiarizes the reader with the AIOps domains and its subdomains, offering an overview of diverse contributions in this research area, it is important to note that it is optional and not necessary for understanding the subsequent chapters, which delve into the details of our own contributions. A more extensive version of this chapter can be found in our Survey. However, it is important to mention that in the discussion sections of the contribution chapters, we will align our work with the proposed taxonomy presented in this chapter. This alignment allows us to position our contributions within the current state of the art, reflecting our vision and advancements in the field.

Roadmap. The remainder of this chapter is organized as follows. In Section 2.2, we provide definitions and clear terminology for the most commonly used terms in the context of AIOps and incident management. This section also presents a framework that unifies these terms with clear examples, helping to clarify their meaning and interconnections. In Section 2.3, we present an exhaustive list of desiderata to be taken into account by both industry and academia when designing and implementing AIOps solutions. Then, Section 2.4 introduces our taxonomy, which is used to classify the existing contributions alongside our own contributions. This taxonomy allows for a detailed exploration of various aspects. Aiming to delve deeper into some aspects of this taxonomy, Section 2.5 discusses the different data sources, while Section 2.6 presents the metrics that should be used to assess AIOps solutions for different tasks, including detection, prediction, and diagnosis. In Section 2.7, we review the most relevant methods in the state of the art for all the defined tasks. Finally, we conclude with a discussion of this chapter, which also serves as an understanding of how our contributions, to be detailed in the upcoming chapters, are located with respect to this research field.

2.2 Definitions and Terminology

In the following, we aim to provide an easy-to-understand explanation of various terms and concepts that are commonly used in incident management and AIOps. We will define these terms from our perspective, offering a formal definition that helps clarify their meaning and relevance in the field.

2.2.1 Faults, Errors, Anomalies, Failures, and Outages

Various terms related to incidents, such as fault, error, bug, failure, outage, and anomaly, have been widely used in the field, often without a thorough examination of their precise meanings. For instance, the most commonly used term in the literature is *failure* to indicate system downtime, hardware and software crashes, service disruptions, network traffic disturbances, etc. [327, 241, 174, 337]. On the other hand, some other methods utilize the term *outage* to refer to the most severe cases that can significantly reduce system availability and impact user

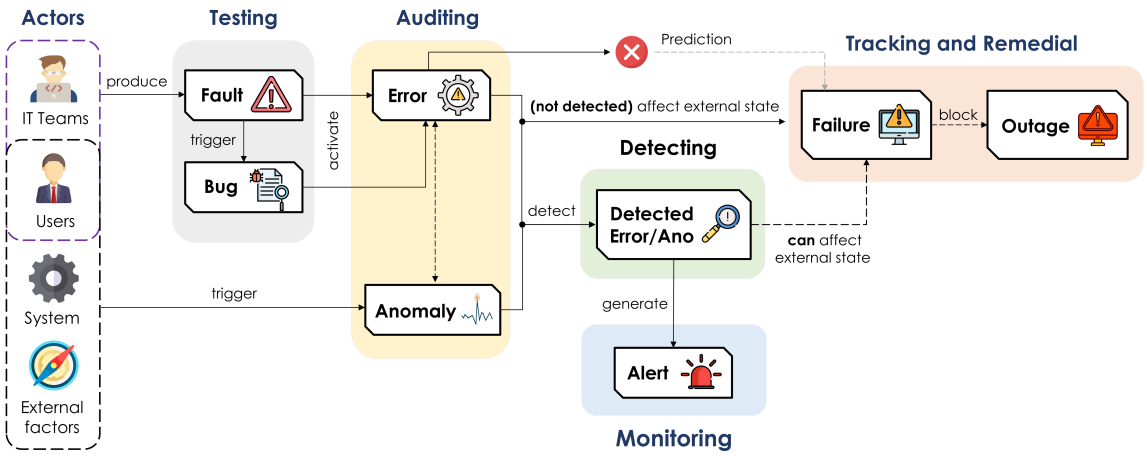


Figure 2.1: Comprehensive chronological schema highlighting the distinctions and key connections among Faults, Bugs, Errors, Anomalies, Failures, and Outages.

experience [62, 333]. These two terms are generally associated with the prediction process. However, there are also studies that focus on *anomaly* detection, which deals with identifying abnormal behaviors that are already present in the data, often in system metrics [67, 297, 225]. Regarding the analysis of root causes, the majority of research falls under the category of *fault* localization, which generally identifies faulty components in the source code [307, 253, 2]. It may also extend to faulty actions [180], but it is agreed that this can be the initial point that ultimately leads to failures. The distinctions between these terms based on faulty behavior, underlying causes, and resulting consequences have not received sufficient attention. Although some attempts have been proposed to categorize and differentiate these terms, such as the work of [260, 21], they do not provide a comprehensive framework. In order to establish a unified and precise terminology that clarifies the meaning of each interrelated term, we propose a coherent lexicon that covers a broader range of concepts compared to the framework proposed by [260]. Moreover, we present a chronological schema in Figure 2.1 illustrating the key relationships among these terms. We also identify the actors involved in initiating faulty behaviors within a system. Subsequently, we provide formal definitions for each term from our perspective.

Failures. A Failure refers to an event that occurs when a system, a component, or a service is unable to fulfill its intended primary function and deviates from it. Failures are often evident and can be observed by either the user or the system, typically stemming from errors or anomalies that give rise to observable issues or malfunctions. It is noteworthy that within systems, various issues may arise, but unless they result in an undesired output, they do not qualify as failures. Users typically report failures, which can prompt the need for troubleshooting or repairs to prevent outages.

Outages. An outage refers to a period in which a system, a service, or a network becomes entirely inaccessible or unavailable to users. Outages can stem from failures such as hardware malfunctions or software glitches indicating complete interruptions or unavailability of the service. These situations often necessitate immediate corrective measures to restore normal operations before delving into the underlying problem for curative maintenance.

Errors. Errors signify instances where the system deviates from its correct and normal state, indicating the presence of an actual issue. These errors may not always be immediately apparent to the user and can remain implicit until they manifest as failures, especially if they are not accurately detected at the opportune moment. Alternatively, errors can be identified through the use of specialized tools or specific algorithms designed for detection purposes.

Anomalies. Anomalies are defined as unexpected or abnormal behaviors of patterns that deviate from the expected state. They represent irregularities or unusual occurrences that may or may not indicate an error. Unlike errors, anomalies can serve as early indications of underlying issues, but they can also be harmless or temporary deviations that do not directly result in failures. Various factors, such as unusual data patterns or external influences like cyber attacks, can contribute to the emergence of anomalies. These anomalies can be identified and detected through the monitoring and analysis of system metrics.

Faults and Bugs. A fault pertains to an abnormality or defect discovered in a hardware or software component that exhibits incorrect behavior, potentially leading to errors and failures if not promptly detected. These faults generally arise from inherent problems or weaknesses within the system's components. They can be caused by various factors, including human interventions by end-users or administrators, design flaws, system settings, or improper handling. In software development, faults manifest as bugs, which stem from coding mistakes or oversights during the development phase. Identifying faults that lead to bugs often takes place during the testing phase. Conversely, in the case of hardware or setup issues, faults directly result in errors during system operation.

Alerts. In addition to leading to failures, both undetected and detected errors and anomalies can cause system parameters to deviate from normal behavior as a side effect. This condition is commonly referred to as symptoms [113]. These symptoms typically manifest as alerting reports, indicating a specific event or condition that demands attention or action. Alerts are usually triggered based on predefined rules or thresholds associated with the symptoms, particularly in the case of anomalies.

Figure 2.1 illustrates the progression of a fault or anomaly, stemming from internal or external factors, towards a failure and potentially an outage. To illustrate this, let's consider a scenario of a fault-tolerant system with a memory leak problem. The fault in this system is a missing `free` statement in the source code, which prevents the proper deallocation of memory. As long as the specific part of the software responsible for memory deallocation is never executed, the fault remains dormant and does not affect the system's operation. However, when the piece of code that should free memory is executed, the software enters an incorrect state, turning into an error. In this case, memory is consumed but never freed, even though it is no longer needed. Initially, if the amount of unnecessarily allocated memory is small, the system may still deliver its intended service without any observable failures from the outside. As the code with the memory leak is executed repeatedly, the amount of free memory gradually decreases over time. This out-of-norm behavior, where the system's parameter `free-memory` deviates from the expected state, can be considered a symptom of the error. It serves as an indication that something is amiss within the system. At some point, when the memory leak has consumed a significant amount of available memory, there may not be enough resources left for certain memory allocations. This leads to the detection of the error, as the system encounters a situation where it cannot allocate memory when required. In a fault-tolerant system, even if a failed memory allocation occurs, it does not

necessarily result in a service failure. The system may employ mechanisms such as spare units to complete the operation and maintain service delivery. Therefore, a single failed memory allocation, by itself, may not cause a service failure or outage. However, if the entire system becomes incapable of delivering its service correctly due to a series of errors or significant resource depletion, a failure occurs. This failure indicates that the system is no longer able to fulfill its intended function, impacting its users and potentially leading to an outage. During an outage, the system becomes completely unavailable, and its services cannot be accessed or utilized by users. In the context of the given example, an outage could happen if the memory leak issue is not addressed in a timely manner, leading to severe resource exhaustion that renders the system inoperable. In summary, the presence of a fault and subsequent error can be indicated by symptoms like memory consumption or depletion. Anomaly detection and monitoring can help identify deviations from expected system behavior. Alerts can be generated to notify system administrators or developers about these anomalies, allowing them to take corrective actions. If the errors and issues persist and prevent the system from delivering its services correctly, a failure occurs, potentially resulting in an outage.

In the following, aiming to provide unified terminology and avoid confusion, we collectively refer to all these terms as *incidents*. This term comprises a broader scope and universally applies to any unplanned event or occurrence that disrupts the normal state, behavior, or output of a system, network, or service.

2.2.2 AIOps and Incident Management Procedure

To date, a universally accepted formal definition of AIOps is yet to emerge due to its novelty and its broad scope involving numerous specialized domains bridging research and industry. While some definitions focus solely on the capabilities and benefits of AIOps, without delving into its conceptual framework and operational processes, several research efforts have taken the initiative to propose a comprehensive definition and outline related subareas (refer to Table 2.1). These definitions commonly converge on two key points. Firstly, AIOps entails the application of artificial intelligence to enhance, fortify, and automate a wide range of IT operations systems. Secondly, AIOps emphasizes the provision of complete visibility, control, and actionable insights into the past, present, and potentially future states of the system under consideration. Other definitions also emphasize the importance of various aspects such as robust data collection, data ingestion and effective querying capabilities, scalable infrastructure, and efficient real-time automated operations. It is important to note that AIOps extends beyond the management of incidents and the automation of maintenance processes. We concur with the findings of [232] that AIOps includes two primary subareas: incident management and resource management, as depicted in Figure 2.2. In a few words, resource management covers efficient allocation and utilization of resources for IT operations, leveraging AI techniques for optimal parameterization. This involves identifying, monitoring, and allocating resources such as servers, storage, network bandwidth, and virtual machines in a well-optimized manner. The objective is to ensure effective utilization of available resources, avoiding bottlenecks, excessive utilization, or underutilization.

Based on our review of various definitions and with the objective of providing a comprehensive understanding of AIOps, with its concepts, building blocks, capabilities, scope, and application to incident management, we propose the following definitions:

Table 2.1: Available AIOps definitions with corresponding capabilities.

Work	Provided Definition	Capabilities
[242]	"AIOps platforms combine big data and machine learning functionality to support all primary IT operations functions through the scalable ingestion and analysis of the ever-increasing volume, variety, and velocity of data generated by IT. The platform enables the concurrent use of multiple data sources, data collection methods, and analytical and presentation technologies"	Performance Analysis, Anomaly Detection, Event Correlation, IT Service Management, and Automation
[76]	"AIOps is about empowering software and service engineers (e.g., developers, program managers, support engineers, site reliability engineers) to efficiently and effectively build and operate online services and applications at scale with artificial intelligence (AI) and machine learning (ML) techniques"	High Service Intelligence, High Customer Satisfaction, High Engineering productivity
[256]	"AIOps is a methodology that is on the frontier of enterprise IT operations. AIOps automates various aspects of IT and utilizes the power of artificial intelligence to create self-learning programs that help revolutionize IT services"	Improving human-AI collaboration, Monitoring and Proactive IT work, Efficient time saving, Faster Mean Time To Repair
[40]	"AIOps is an emerging interdisciplinary field arising in the intersection between the research areas of machine learning, big data, streaming analytics, and the management of IT operations"	Efficient Resource Management and Scheduling, Complex failure management

AI for Operating Systems. AIOps is a transformative concept that establishes a symbiotic relationship between humans and machines. It leverages advanced artificial intelligence, including machine learning algorithms and data analytics techniques, to process a large volume and variety of data generated or collected by IT systems (commonly known as big data). The primary aim is to enable autonomous and intelligent capabilities for managing and optimizing complex information technology ecosystems. At its core, AIOps is built upon intelligent algorithms executed on historical and real-time data this is supposed to be collected and queried efficiently and effectively. This data is converted into actionable and innovative insights and prescriptive recommendations with the goal to optimize operational performance, provide situational awareness, and take initiatives to detect, diagnose, predict, and resolve incidents.

Incident Management Procedure. The incident management procedure is a systematic and well-structured approach aimed at efficiently and effectively handling incidents within an organization's IT environment. This procedure includes a predefined series of steps and processes to be followed, whether incidents arise unexpectedly or are predicted in advance. Its primary objective is to ensure a consistent and coordinated response, regardless of the source of the incident report, from internal or external users, abnormal system behavior, or proactive prediction mechanisms. The procedure involves the qualification, classification, prioritization, diagnosis, and resolution of incidents. Throughout the entire process, clear communication channels are established to keep stakeholders informed of the incident's status

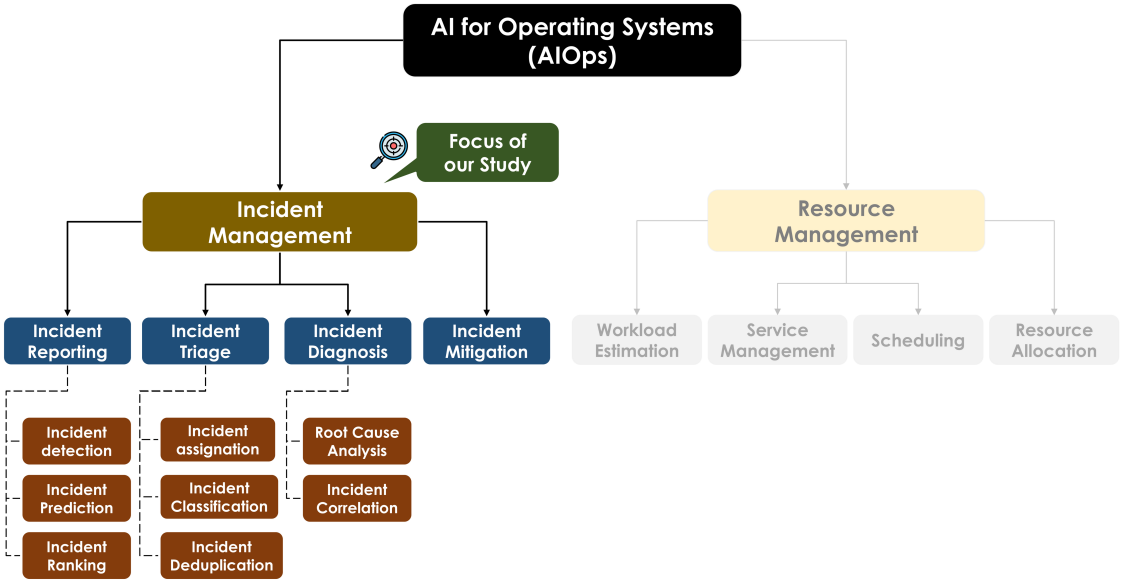


Figure 2.2: Exploring the research landscape of AIOps subareas with a focus on Incident Management.

and any relevant actions taken. Timeliness is of the essence, and every effort is made to minimize critical time metrics time to report (TTR), time to engage (TTE), time to diagnose (TTD), and time to mitigate (TTM). Furthermore, upon successful resolution of incidents, a crucial post-incident review is initiated. This stage serves to identify valuable lessons learned, enhance existing processes, and proactively prevent similar incidents in the future.

In the following, we will provide an extensive review of the essential subcategories within the incident management procedure. Our goal is to examine the most relevant approaches proposed for each of these phases. It is important to note that not all incidents go through every step, as certain steps may not be necessary in specific situations. For example, it is not possible to both detect and predict an incident at the same time. It is worth mentioning that some previous work has already defined certain terms related to incident management. For instance, Chen et al. [64] defined the incident management procedure as a three-step process involving incident reporting, triage, and mitigation. However, their definition remains quite generic and does not delve into the subcategories of these phases, such as addressing the problem of incident classification and correlation. On the other hand, Notaro et al. [232] focused on studying failures and developed a taxonomy based on proactive and reactive approaches, with significant emphasis on the reporting phase. However, their research appears to neglect other important phases. Additionally, Zhang et al. [326] provided a formal definition that includes detailed phases like assignment, prioritization, fault localization, and mitigation. However, this survey solely concentrates on software bugs and does not generalize to other specific areas. In this work, our aim is to cover all these use cases under a unified taxonomy, regardless of the terminology used (failures, bugs, anomalies, etc.) or the specific industry focus. Therefore, we categorize the incident management phases according to the available contributions and in accordance with the terminologies outlined in Figures 2.1 and 2.2 as follows:

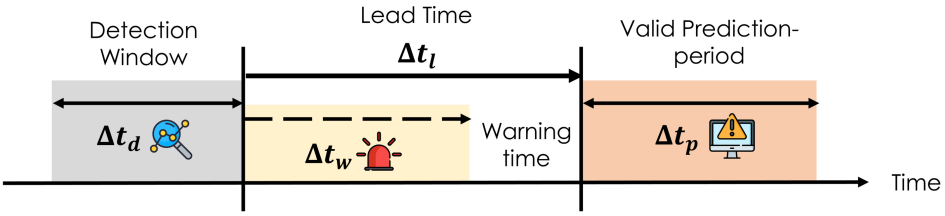


Figure 2.3: Time relations in online failure prediction.

Incident Detection. Incident detection refers to the process of identifying and recognizing deviations from normal operations indicating the presence of abnormal behavior or faulty events that may indicate the occurrence of an incident (e.g., error or anomaly). This process involves monitoring and analyzing various data sources (e.g., KPI metrics, system logs, and user reports). For instance, the domain of *anomaly detection* falls within this subcategory.

Incident Prediction. Incident prediction refers to the process of forecasting, anticipating, or estimating the likelihood of potential incidents, primarily failures, before they occur. It involves leveraging available historical data, along with other relevant factors, to proactively identify and assess potential risks and vulnerabilities. By analyzing patterns, trends, abnormal events, and anomalies in historical and current data using advanced analytics techniques, incident prediction aims to enable organizations to take preventive actions and minimize the impact of future incidents. Incident prediction can be categorized into offline and online methods. The offline category includes Software Defect Prediction (SDP) [83, 173], which entails assessing failure risks by executing specific functional units or analyzing portions of the source code. Another technique in this category is fault injection[269, 226], where deliberate faults are introduced into a functioning system through stress testing to evaluate its fault tolerance level. Conversely, online prediction occurs during system runtime. It involves techniques like software rejuvenation [8, 285], which addresses resource exhaustion and prevents unexpected system failures caused by aging or accumulated faults in software systems. Additionally, online prediction involves estimating the remaining useful lifetime of the system. This category also invloves real-time predictions of hardware and software failures, taking into account time constraints, as depicted in Figure 2.3. For a prediction to be considered valid, it must be made with a lead-time (Δt_l) greater than the minimal warning time (Δt_w). Moreover, the prediction is only considered valid if the failure occurs within a specific time period called the prediction period (Δt_p). To make these predictions, data up to a certain time horizon (Δt_d), referred to as the data window size, is utilized.

Incident Prioritization. Incident prioritization is the process of categorizing and ranking incidents based on their urgency, impact, and business priorities. It involves evaluating the severity of the incident, considering factors such as the affected systems, services, and the potential business impact. Incident prioritization ensures as well that resources are allocated appropriately, with relatively critical incidents receiving immediate attention and resources.

Incident Assignment. Incident assignment entails the allocation of incidents to the relevant individuals or teams responsible for investigating and resolving them. This crucial process

involves analyzing the information contained in incident reports, considering factors such as the nature and complexity of the incident, as well as the skills and availability of the assigned personnel. This process is commonly referred to as incident triage in several work [59, 329]. However, triage in a general sense, does not refer only to assignment but also to incident classification and identification of duplicate incidents.

Incident Classification. Incident classification involves the systematic grouping and categorization of incidents based on their distinct characteristics, symptoms, or impact. This classification process establishes a structured framework that enhances the understanding and effective management of incidents. Surprisingly, despite its crucial role in optimizing incident management time response, incident classification has not received sufficient coverage or extensive attention. While some studies, such as [240, 10], have approached this issue by treating it as a content optimization problem, others have included it as part of prioritization [161, 281]. Some researchers have even considered it within the scope of duplicate detection [24, 134]. However, we believe that there are inherent differences between these categories. Incident classification primarily focuses on associating the incident with a specific topic or category, regardless of the assigned personnel, incident priority, presence of similar incidents, or whether it is a new incident.

Incident Deduplication. Near-duplicate incident detection is the process that efficiently identifies incidents that are closely related or exhibit significant similarities, grouping them into specific buckets that correspond to distinct problems. This process involves real-time analysis of incoming incidents to determine their similarities, overlaps, and commonalities with historical incidents that pertain to the same topic. By eliminating duplicates, incident deduplication reduces redundancy, streamlines incident management efforts, and prevents unnecessary resource allocation. In fact, Anvik et al. [11] conducted a comprehensive empirical study using bug repositories from Eclipse and Firefox, which revealed that a significant portion (20%-40%) of bug reports are flagged as duplicates by developers. This study provides concrete evidence of the necessity to detect duplicate bug reports, shedding light on the phenomenon of bug-report duplication. The process of detecting near-duplicates can be considered as a further refinement of the incident classification problem. As already explained in the introduction, It is commonly referred to as an unsupervised solution, which aims to accurately identify incidents that are highly similar within a given particular class.

Root Cause Analysis. Root cause analysis (RCA), also known as root cause diagnosis, plays a pivotal role in the incident management procedure. It is a systematic process that aims to investigate and identify the underlying causes and contributing factors of incidents, which we commonly refer to as faults. As depicted in Figure 2.4, RCA delves into the fundamental fault or primary trigger behind the occurrence of an incident, recognizing that it may not necessarily stem from a faulty manipulation or defective source code. Its objective goes beyond addressing the symptoms; instead, it seeks to uncover and understand the root cause, even if it originates from external factors. In related studies, this process is often referred to as Fault Localization [303]. These studies focus on pinpointing the specific set of components (such as devices, network links, hosts, software modules, etc.) that are associated with a fault that has caused or may cause a particular failure. However, root cause analysis goes beyond component localization and extends its investigation to faulty actions or risky external factors that lead to abnormal behaviors within the system.

Incident Correlation. Incident correlation study involves the analysis of multiple incidents

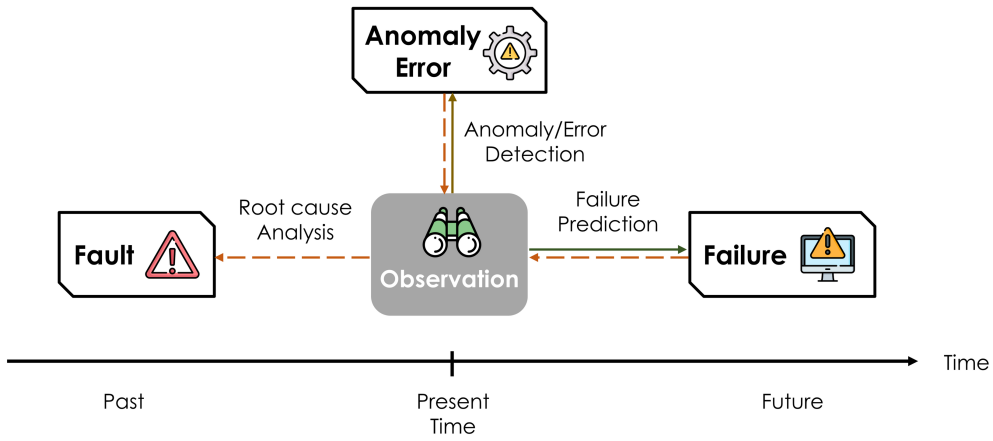


Figure 2.4: Distinction between Root Cause Analysis, Anomaly Detection and Failure Prediction.

or events with the aim of identifying relationships, dependencies, or shared characteristics among them. Through this process, a holistic perspective of the incident landscape can be achieved, allowing for a deeper understanding of the impact and the ability to uncover potential hidden patterns or trends. Incident correlation works alongside root cause analysis to assess how the underlying causes and faulty components or behaviors can affect other incidents. This helps in facilitating more efficient incident resolution. The task is considered challenging due to the inherent complexity and interdependence among components in software systems [14].

Incident Mitigation. Incident mitigation also known as remediation, refers to the process of minimizing the impact and severity of an incident. It involves taking proactive and automatic measures to contain, resolve, or reduce the effects of the incident. Incident mitigation can include implementing temporary workarounds, applying fixes or patches, activating backup systems, or engaging specialized resources or teams to restore normal operations. As mentioned in [232], contributions related to remediation actions have been relatively fewer compared to incident prediction, detection, triage, and diagnosis tasks. This could be attributed to the fact that once the underlying problem is identified through diagnosis, the necessary recovery steps become readily identifiable and achievable. In many cases, historical incidents with similar resolutions can be referenced, eliminating the need for complex models.

2.3 Desiderata for Effective Incident Management

Developing intelligent, data-driven approaches for incident management is a complex process that goes beyond traditional machine learning techniques. Simply relying on high-performing machine learning or big data mining models is insufficient for successfully adopting AIOps solutions. To ensure the effectiveness of such solutions, they must adhere to a set of established criteria, which we refer to as desiderata. Drawing from numerous reviewed studies, including [76, 202, 194, 335, 177], we have compiled a comprehensive list of requirements that should be considered, either fully or partially, when constructing AIOps solutions. These requirements

are as follows:

1. **Trustablity.** The literature claims that the requirements for employees skills and mindsets change with the introduction of AIOps [246]. Manual activities tend to shift towards adaptation and auditing tasks while dealing with AI requires a different approach focused on recognizing patterns from raw data, deviating from the traditional developer mindset. This transition raises questions about trust in AI capabilities and what it can offer. Consequently, adopted AIOps approaches should incorporate years of field-tested engineer-trusted domain expertise iteratively and interactively into the learning, updating, and explanation phases of sophisticated machine learning models built on raw data. IT professionals possess valuable domain knowledge and insights acquired through years of experience in managing and troubleshooting IT systems. While not all of their best practices may scale with the AIOps trends, their expertise often extends beyond raw data analysis. They have a deep understanding of the underlying technology, infrastructure, applications, and business requirements. It is crucial to fully leverage and model this expertise into AIOps solutions. This can be achieved by providing mechanisms that incorporate the human in the loop, allowing for interaction, updates, and corrections to the models when necessary.

2. **Interpretability.** AIOps solutions should prioritize interpretability, even if it comes at the expense of model performance. In the context of AIOps, interpretable models are preferred when high-performing models lack interpretability. Model transparency enables users to fully understand, interact with, and reason about the recommendations made by the model, which can help gain support from upper management in following those recommendations. However, interpreting AIOps models comes with certain constraints and requirements. In a study by [202], different factors influencing AIOps model interpretation are investigated across three key dimensions. (1) Internal Consistency which assesses the similarity between interpretations derived from AIOps models trained under the same setup. It examines whether the interpretations obtained from an AIOps model are reproducible when the model is trained with the same data and implementation across multiple executions. (2) External Consistency which focuses on the similarity between interpretations derived from similar-performing AIOps models on a given dataset. Intuitively, interpretations derived from a low-performing interpretable model could be trustworthy only if the interpretable model has the same interpretation as other machine learning models on a given dataset. (3) Time Consistency which captures the similarity between interpretations derived from an AIOps model across different time periods. AIOps models should not only reflect the trends observed in the most recent training data but also capture and reflect trends observed over a longer period. It is important to note that some previous work, such as [25] in defect prediction, has shown that models trained on one time period may not generalize well when tested on a different time period. Additionally, the size of the training data can impact the derived interpretations of the models.

3. **Scalability.** AIOps solutions must efficiently handle large-scale data in complex IT environments where significant amounts of monitoring and log data are expected. These environments can encompass thousands to millions of nodes, including servers, network devices, and applications. To go beyond effective modeling and accurate results, it is

crucial for AIOps solutions to be implemented within robust architectures that excel at ingesting, storing, and processing big data efficiently. Scalable architectures and data processing frameworks play a key role in distributing the workload and effectively handling the high volume of data. Additionally, when considering the adopted approaches, AIOps solutions should leverage scalable computing techniques, such as distributed and federated learning, as discussed in studies like [88, 28, 227], to enable parallel processing and distributed data analysis. Scalability also invloves optimizing the utilization of computational resources. AIOps solutions should possess the capability to dynamically allocate and distribute resources based on the data volume and processing requirements. This ensures efficient resource utilization and minimizes bottlenecks that could hinder performance.

4. **Maintainability and Adaptability.** The concepts of maintainability and adaptability are crucial in AIOps, as they aim to minimize the need for ongoing maintenance and repetitive fine-tuning. This consideration is essential because DevOps engineers, who are responsible for managing and maintaining these solutions, often have a multitude of responsibilities and may not possess extensive expertise in machine learning. Therefore, AIOps solutions should strive for a high degree of automation and self-management of routine tasks such as data preprocessing and regular model training to reduce the reliance on continuous manual interventions. To achieve this, self-adjusting algorithms and automated pipelines can be employed [34]. In addition, leveraging advanced machine learning techniques such as transfer learning [237] and one-shot learning [290] can greatly benefit AIOps solutions. Instead of training models from scratch, pre-trained models that have been developed and fine-tuned by machine learning experts can be utilized to handle new data patterns.

5. **Robustness.** AIOps solutions need to be built upon robust and stable machine learning models that can handle a wide range of scenarios and exhibit resilience to variations in data patterns. These models should be designed to be less sensitive to the noisy and incomplete data commonly encountered in real-world IT environments [76]. To ensure the reliability of the modeling process, robust preprocessing techniques, such as systematic data cleaning and effective imputation methods, can be employed. In addition, AIOps solutions must be capable of detecting and adapting to concept drift, which refers to the shifts in underlying data distributions that occur in dynamic IT environments [201]. Robust algorithms and models, such as those based on online learning, can be leveraged to handle concept drift and maintain up-to-date insights in the face of evolving data patterns [50, 49]. Furthermore, AIOps solutions should generalize well across different IT environments. To achieve this, they should be trained on diverse and representative data that captures the underlying patterns and relationships applicable across various scenarios.

6. **In-context Evaluation.** Unlike conventional machine learning evaluation scenarios, such as cross-validation, which are often not directly applicable to AIOps due to the unique characteristics of real-world IT environments, the concept of in-context evaluation emphasizes the need to assess the solution's performance in a context that closely resembles its actual production usage. Traditional evaluation methods typically assume that the data used for evaluation is identically distributed, which may not hold true in

IT environments. Real-world data often exhibits temporal dependencies, concept drift, and dynamic patterns, which require specialized evaluation techniques that consider these factors. To conduct in-context evaluation, it is important to create evaluation frameworks that capture the intricacies of the production environment. This involves using datasets with a broad range of scenarios, including normal operations, various types of incidents, and different environmental conditions. In addition to dataset selection, evaluating AIOps solutions in context also requires defining appropriate evaluation metrics and benchmarks that align with the desired outcomes and objectives. Furthermore, in-context evaluation may involve conducting experiments and simulations that mimic real-world conditions, allowing for comprehensive testing of the AIOps solution's performance and robustness.

2.4 Proposed Taxonomy

As previously mentioned in 1.4.1, our aim is to present a taxonomy that categorizes the work related to AIOps for incident management into primary groups, covering a diverse range of factors that are crucial to consider when assessing the necessity, design, implementation, and reproducibility of the reviewed methods. These categories are carefully chosen to provide a comprehensive and inclusive framework for analyzing the various dimensions that impact AIOps for incident management. Each of the categories is explained in detail below.

1. **Context.** It represents the environmental factors that drive and surround the proposed approach and includes:

   A. Focus Area. It refers to the specific research area in which the proposed approach fits within the incident management procedure. More precisely, it addresses one of the distinct phases of the incident, as identified in our categorization, which encompasses reporting, triage, diagnosis, or mitigation and their subcategories.

   B. Maintenance Strata. It refers to the different layers or components of the system as highlighted in Figure 1.3 that are targeted by the method under examination.

   C. Scoop and Industry Focus. This concept refers to the specific industrial application area of the method being studied. Some methods may be exclusively dedicated to a particular application domain, such as IoT systems, cloud infrastructures, software systems, etc. Different industries have varying requirements and constraints, and the reviewed method may need to be tailored accordingly. For instance, a method that works well in one industry may not be as effective in another.

2. **Data Availability and Processing.** This category refers to factors pertaining to the characteristics of the data and their representation, as well as the requisite measures necessary to render them actionable for efficient analysis and interpretation.

   A. Data Sources. It refers to the nature of data that the method is built upon. The proposed taxonomy includes various types of data sources, such as log metrics, source code, key performance indicators, topology (environmental characteristics), alerting signals, execution traces, network traffic, etc.

   B. Data Types. Consideration should be attributed to how data is represented. Even when using the same data source, different representations are possible. For example, source code can be represented as an Abstract Syntax Tree (AST), a sequence of predefined frames, or plain natural language text. To facilitate this understanding, a taxonomy of data types has been adopted. This taxonomy includes structured data, sequential data, graph data, time series (both univariate and multivariate), textual data, hierarchical data, etc.

   C. Required Feature Engineering Process. This refers to any necessary preprocessing steps to select or transform the data before it can be inputted into the model.

3. **Model Design.** This category highlights the details about the design and implementation of the approach by clarifying the methodology adopted, the learning paradigm or research area employed, and how the approach was evaluated.

   A. Approach. It represents the principal approach employed to address the problem at hand, taking into consideration the context and the data utilized. It elucidates the way in which the authors formalized and tackled the problem, as well as their subsequent course of action. For instance, in the realm of predictive models, the authors may have implemented neural network architectures like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), or Transformers, etc., or opted for other methods such as clustering, nearest neighbor search, dimensionality reduction, or descriptive models like frequent pattern mining, etc.

   B. Paradigm. The paradigm provides an abstract demonstration of how the approach was carried out and trained to attain its objective. For instance, in the case of training predictive models, it may involve discerning whether the approach was supervised, semi-supervised, or unsupervised, or whether it entailed pure one-shot, multitasking, reinforcement, or transfer learning, etc.

   C. Evaluation Metrics. This relates to the evaluation metrics used to assess how well the method performs compared to other techniques currently used in the field.

   D. Package Availability. This factor is highly important to reveal the accessibility of both data and model packages for reproducibility purposes and to guarantee that the work can be utilized in other contexts.

4. **Particularities.** The final category of our taxonomy concerns essential factors that highlight specific attributes and desired outcomes associated with AIOps. We focus on four key factors: interpretability, transferability, human-in-the-loop, and scalability. Interpretability refers to the clarity of the tools provided by the authors, which facilitate understanding of the model's mechanisms and experimental results. Transferability denotes the model's ability to be applied to different contexts and domains with similar data representations, motivations, or configurations. Human-in-the-loop emphasizes the incorporation of human expertise and knowledge to guide the model, provide feedback, make decisions, and validate results. Scalability refers to the model's capacity to scale up or down based on various hyperparameters, such as system size. We also examine any other pertinent factors explicitly claimed to be adhered to by the approach, such as security or maintainability.

In the following, we will delve into major factors that are exclusively relevant to our studied domain. Our focus is specifically to explain and explore the different data sources and outline the evaluation methods employed for the proposed AI approaches.

2.5 Data Sources and Types

Data plays the most crucial role in incident management, serving as the fundamental building block that guides the design of the approach used to identify predictive or descriptive patterns within it. The primary objective is to use this data to effectively accomplish the desired task at hand. These data sources offer valuable insights into the IT infrastructure, application performance, and user behavior. In an industrial setting, data is derived from a multitude of sources, including physical or software components, and can also be generated or edited by humans. One key characteristic of this data is its unstructured nature, lacking a standardized format and displaying non-homogeneity. Consequently, data collected from different sources may possess vastly different formats, requiring a pre-processing stage to perform subsequent analysis. Furthermore, data from the same source can be represented in various ways. We categorize data sources based on the following convention.

1. **Source Code.** It represents the fundamental building blocks of software, including various units such as functions, file codes, classes, and modules. It reflects the design, structure, and logic of different functionalities and services within the software system. In previous works, source code has been represented in diverse ways, including through code metrics, which are handcrafted features derived directly from the source code. Code metrics play a critical role in software defect prediction by quantifying different aspects of the codebase that can impact software quality. These metrics cover a wide range, from static module-level metrics [208] for procedural languages, such as Lines of Code (LOC), Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM), and Depth of Inheritance Tree (DIT), which provide insights into module complexity and potential defect-proneness. At the class level [66], metrics like Number of Methods (NOM), Weighted Methods per Class (WMC), and Response for a Class (RFC) offer indications of class complexity and the potential occurrence of defects. Machine learning algorithms are trained using these metrics to identify patterns and relationships between code quality and defects. Another representation of source code is in the form of Abstract Syntax Trees (AST) [294], which capture the syntactic structure of the code by breaking it down into constituent elements such as expressions, statements, functions, classes, and variables (Figure 2.5). The AST also represents the relationships between these elements, including their nesting and dependencies. Additionally, source code has been modeled using the program spectrum [2, 52], which provides execution information from specific perspectives, such as conditional branches or loop-free intra-procedural paths and also as a sequence of tokens [229, 322] in many research works, particularly in Fault Localization.

2. **Topology (Environment Features).** Topology refers to the structure, either physical or logical, of the IT environment. It encompasses information about the components, connections, and spatial relationships within the system. This data source offers valuable insights into the overall architecture, providing details about servers,

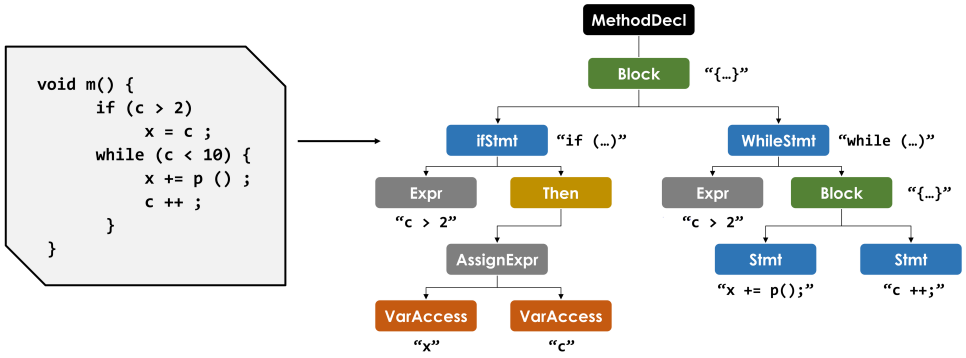


Figure 2.5: Sample Java method and its Abstract Syntax Tree (AST).

```
1 public void setTemperature(Integer temperature) {
2   // ...
3   logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);
4   if (temperature.intValue() > 50) {
5     logger.info("Temperature has risen above 50 degrees.");
6   }
7 }
```

⇓

```
1 0 [setTemperature] DEBUG Wombat - Temperature set to 61. Old temperature was 42.
2 0 [setTemperature] INFO Wombat - Temperature has risen above 50 degrees.
```

Figure 2.6: An example of logging statements by SLF4J and the generated logs [123].

network devices, databases, and their interdependencies. Additionally, topology data may include configuration settings, software versions, and other relevant features of the system. The presence of topology is crucial as it establishes the context necessary for identifying meaningful and actionable patterns within the data. Without the constraints and context provided by topology, the detected patterns, while valid, may be misleading or distracting. Topology has found extensive use in various areas, such as determining causality and localizing faults [187, 247], ranking incident [188], prediction of incident [177], as well as enhancing explainability in the IT environment [248].

3. **Event Logs.** Logs consist of human-readable statements generated by software applications, operating systems, or devices to describe events or actions. They serve as valuable records, providing information about system activities, errors, warnings, and other relevant events. Timestamps are typically included in logs, along with details such as the event's source, severity, and description. Analyzing logs is essential for understanding the sequence of events leading to an incident, identifying abnormal behaviors, debugging issues, and ultimately pinpointing the root cause. In general, logs are semi-structured text that is produced by logging statements (e.g., `printf()`, `logger.info()`) within the source code. For instance, Figure 2.6 displays two log messages generated by corresponding logging statements in the code. The initial words of the log messages (e.g., "Wombat") are determined by the logging framework (e.g.,

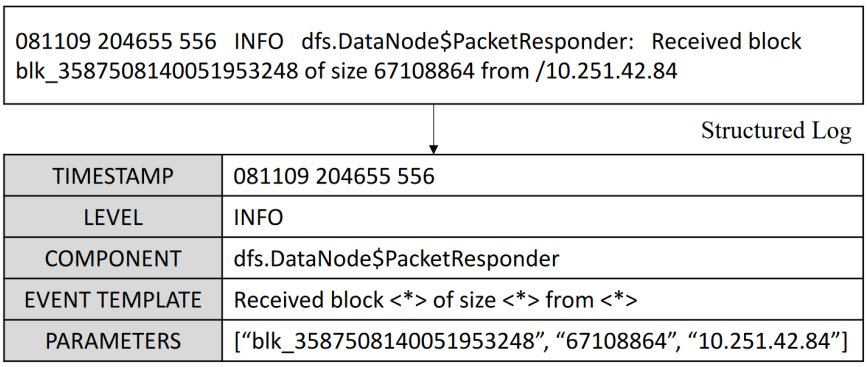


Figure 2.7: Log parsing example [318].

SLF4J[1]), and they follow a structured format. On the other hand, the remaining words (e.g., "50 degrees") are unstructured as they are written by developers to describe specific system runtime events. Once logs are collected, they need to be parsed to be utilized in various downstream log mining tasks, such as anomaly detection.

Parsing log messages is a crucial step in making logs usable for different analytical tasks. This process aims to transform the semi-structured log messages into structured log events by extracting (constant parts) and variables (variable parts) [123]. In the given example depicted in Figure 2.7, a log parsing scenario is presented, showcasing a log message obtained from the Hadoop Distributed File System (HDFS) [318]. A log message consists of two main components, the message header, and the message content. The message header, which is determined by the logging framework, is relatively straightforward to extract, including details like verbosity levels (e.g., INFO). On the other hand, extracting essential information from the message content proves to be more challenging due to its unstructured nature, primarily consisting of free-form natural language written by developers. Typically, the message content comprises both constants and variables. Constants represent fixed text provided by developers (e.g., the word "Received"), describing a particular system event. On the other hand, variables correspond to the dynamic runtime values of program variables, carrying contextual information. The set of constants forms the event template.

Numerous log parsing techniques have been proposed, including clustering-based approaches (e.g., LKE [103], LogSig [280]), heuristic-based methods (e.g., iPLoM [204], SLCT [284]), Evolutionary Algorithms such as MoLFI [212], and Frequent Pattern Mining techniques like Logram [73]. These algorithms are evaluated based on factors such as offline or online parsing mode, coverage, and alignment with domain knowledge. Parsed logs have been used in log mining algorithms with different approaches, including structured features [318, 122], log sequences [91, 209], graphs [225] and Finite State Automatas (FSA) [193].

4. **Key Performance Indicators (KPIs).** These metrics serve as performance indicators for assessing the health and efficiency of IT infrastructure and services. They

[1] `https://www.slf4j.org/`

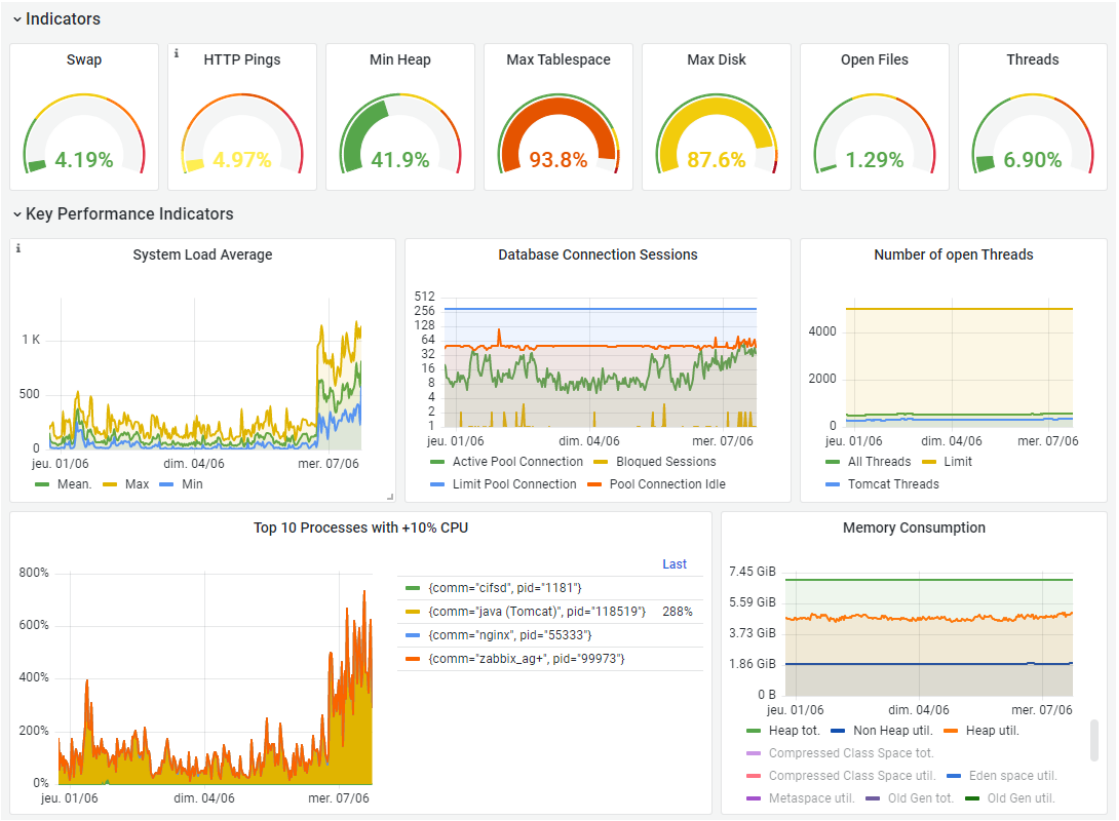


Figure 2.8: Examples of Key Performance Indicators (KPIs) from Infologic maintenance dashboard.

provide quantitative measurements that offer insights into system performance, availability, reliability, and response times. Examples of these metrics include response time, error rates, disk reads, resource utilization (such as CPU, Swap, and Memory Consumption), and up-time. In Figure 2.8, we illustrate a sample set of real-time monitored metrics within our company, Infologic, including disk and min heap utilization, as well as the number of open threads, etc. Typically, these metrics are represented as univariate or multivariate time series, which are utilized for various purposes such as detecting abnormal trends [297, 182], predicting failure by monitoring certain measures [174, 195], estimating the remaining useful lifetime of physical components, or storage capacity [337], identifying recurrent and unknown performance issues [183], and conducting root cause analysis and incident correlations [136, 266]. For instance, Lu et al. [195] utilize SMART (Self-Monitoring, Analysis, and Reporting Technology) multivariate time series data that combines disk performance and disk location data to accurately predict disk failures.

5. **Network Traffic.** This type of data involves the analysis of data packet flow within a computer network, including key information such as source and destination IP addresses, ports, protocols, and packet sizes. This data source provides valuable insights into communication patterns, network congestion, anomalies, and potential security threats. By analyzing network traffic, it becomes possible to identify and address network-related issues, pinpoint performance bottlenecks, and detect signs of malicious

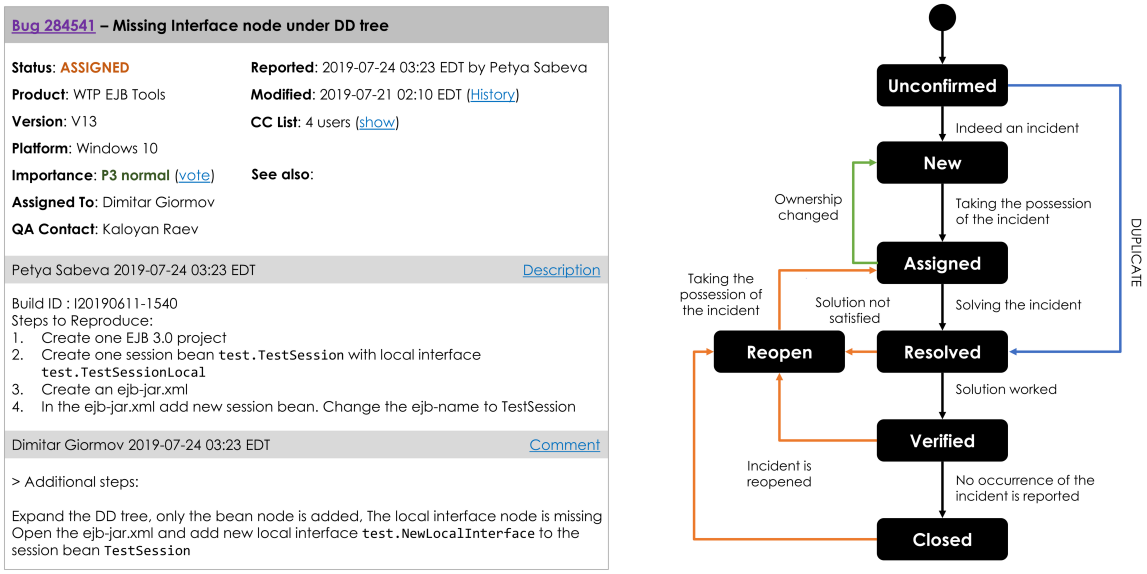


Figure 2.9: [Left] Example of an incident report for a bug in Eclipse. This bug is about a missing node of XML files in Product Web Tools Platform [320]. [Right] Bug report lifecycle according to [326].

activities that could lead to significant incidents. Different approaches have been employed to model network traffic data. For instance, in one study [159], SNMP data was utilized to monitor network links and diagnose anomalies in network traffic. The authors treated flow measurements as multivariate time series collected over time, enabling the separation of traffic into normal and anomalous subspaces. In a subsequent work by the same authors [160], static features such as source and target destination addresses or ports were incorporated into the traffic data to detect and diagnose security threats and service outages. Another approach, presented in [295], involved representing network traffic as image-like structures using the IDX file format for encrypted traffic classification. This process involved mapping the network traffic data onto a 2D grid, where each grid cell represented a specific traffic attribute, such as packet size, source IP address, or destination port. The intensity or color of each pixel in the image reflected the value or frequency of the corresponding network traffic attribute at that particular location.

Furthermore, network traffic has been represented as graphs by [23] to localize the sources of performance problems in networks. Probabilistic inference graphs were constructed from the observation of packets exchanged in the network infrastructure. Nodes of the inference graph are divided into root cause nodes (corresponding to internal IP entities), observation nodes (corresponding to clients), and meta-nodes, which model the dependencies between the first two types of nodes. Each node is also associated with a categorical random variable modeling the current state (up, troubled, down), which is influenced by other nodes via the dependency probabilities.

6. **Incident Reports.** Incident reports are valuable sources of information that comprehensively document the details, impact, discussions, and resolution of incidents. These

reports are typically initiated by developers, testers, or end-users, capturing crucial information to aid incident management. As illustrated in Figure 2.9(left), incident reports commonly include an identification number and a title, along with contextual features such as the timeline of when the incident was reported and modified, the affected system or service's topology, the severity or importance of the incident, the assigned programmer responsible for resolution, and the incident's resolution status (e.g., new, unconfirmed, resolved). Of utmost importance, incident reports contain a detailed description of the issue, including information on how to reproduce the problem, stack traces (in the case of a bug), and the expected behavior. Additional comments within the report may include discussions about potential solutions, diagnosis and root cause analysis, and actions taken to mitigate the incident. Attachments such as proposed patches, test cases, or screenshots may also be included to provide further context and support. Furthermore, incident reports should offer historical context to leverage past knowledge from similar incidents. Tagging relevant information from previous incidents enables to the application of valuable lessons learned in the current resolution processes.

The challenge in processing incident reports lies in the diversity of data types, including structured, semi-structured, and unstructured data. For example, environment characteristics are typically presented in structured or tabular formats [64, 320, 238], while stack traces, problematic SQL queries and user traces fall into the category of semi-structured data [326, 238]. On the other hand, the description of the problem and the comments section dedicated to the analysis and diagnosis of the problem consist of unstructured natural language text [59, 320, 165], which requires data normalization. Various approaches have been employed to encode these different data types. For instance, both Chen et al. [59] and Pham et al. [238] utilized the FastText algorithm [41] for text encoding. Another approach employed by [165] was the use of Word2Vec [214], which builds pretrained subword vectors based on an external corpus and then fine-tunes them using historical incident data. Contextualization data, on the other hand, was handled using exponential family embeddings in the work of [238]. In addition, some researchers explored Assignment information in incident reports to create what is referred to as a "Tossing Graph," aiming to reduce the need for reassigning incidents to other developers [309, 38, 135].

The textual incident report plays a crucial role in the incident triage process and undergoes various stages throughout its lifecycle. Figure 2.9(right) illustrates the lifecycle of an incident report. Initially, when an incident is reported, the report is marked as UNCONFIRMED. At this stage, a verification process is conducted to ensure that the incident is not a duplicate and is indeed a new issue. If it is confirmed to be a new incident, the status is changed to NEW. Next, an assignment mechanism is employed to assign the incident report to the most qualified and available developer to address the problem. The status is then updated to ASSIGNED. The assigned developer works on reproducing and localizing the incident, aiming to fix it. Once the incident has been resolved, the status of the report is changed to RESOLVED. In some cases, if the tester is not satisfied with the solution, the incident may be reopened, and the status is set to REOPEN. Conversely, if the tester verifies that the solution has effectively resolved the issue, the status is changed to VERIFIED. The final status of an incident report is CLOSED, indicating that no further occurrences of the incident have been reported.

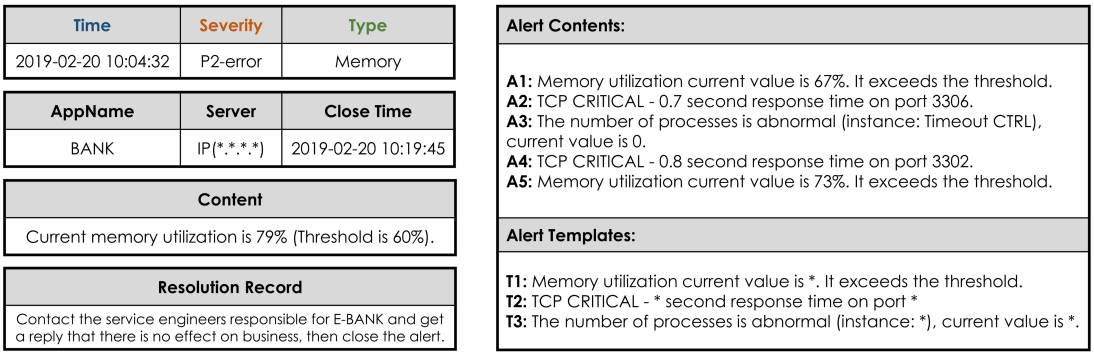


Figure 2.10: [Left] Example of an alert on memory consumption from [334]. [Right] Explanation of alert template extraction.

Different outcomes can follow the RESOLVED status. If the developer successfully addresses the incident by making necessary code changes, the status is updated to FIXED. However, if the developer is unable to resolve the incident for any reason, the status is set to WONTFIX. In cases where the developer identifies the report as a duplicate of an existing incident, the status is changed to DUPLICATE.

7. **Alerting Signals.** Alerting signals are notifications generated by monitoring systems or tools when specific thresholds on time series metrics or conditions on event occurrences are surpassed. These signals serve as indications of abnormal behavior, potential issues, or breaches of established thresholds. Examples of triggering events include high CPU usage, low disk space, high latency time, or application errors. Alerting signals act as an early warning system, allowing proactive identification and resolution of emerging problems before they escalate into failures. It's important to note that alerting signals are not raw or unprocessed data directly collected from monitored systems, such as KPI metrics or event logs. Instead, they are refined data derived from metrics or events based on a set of predefined rules. For example, Figure 2.10 illustrates an example alert generated by the AlertRank Framework [334], indicating that the current memory utilization has exceeded the threshold of 79%, leading to a P2-error severity. This alert is derived from the analysis of memory consumption over time and generated using alert templates within the source code, similar to log events, as shown in Figure 2.10. Alerting signals are commonly used to identify and prioritize critical issues within a system [334]. They can also be utilized for categorizing similar or identical problems into specific categories [332], as well as deducing correlations among multiple simultaneous events [215].

8. **Execution Traces.** In addition to the main data sources mentioned above, which are utilized in the incident management process to develop data-driven approaches for incident detection, diagnosis, triage, and resolution, there are other data sources that are specifically relevant to certain tasks. One such example is execution traces, which provide a hierarchical description of the modules and services invoked to fulfill a user request and are employed in the diagnosis task. These traces capture the flow of control,

method invocations, input/output data, and interactions with external dependencies. Execution traces are particularly valuable for diagnosing complex or intermittent problems that are challenging to reproduce.

Stack traces, for example, are detailed reports that provide information about the executed methods and their associated packages during a crash. They can be obtained through system calls in various programming languages. In Java, stack traces are presented in descending order, with the top of the stack trace indicating the most recent method call. Stack traces have primarily been utilized in the context of crash deduplication, which involves identifying near-duplicate reports that indicate the same bug or error. In the field of research, stack traces have been modeled using graphical representations [150]., sequence-based approaches [251, 48, 87], or vectorization techniques such as n-grams and TF-IDF for information retrieval purposes [171, 258]. Another prominent type of execution trace comprises SQL queries executed to retrieve a service or important data. Significant emphasis has been placed on analyzing SQL workloads in the context of diagnosing incidents. This analysis aims to identify schema issues in data models within databases and improve performance, such as recommending and selecting indexes [82, 57], detecting anti-patterns [61], and identifying insider threats [156]. Parsing SQL queries efficiently is crucial for feeding them into analytical models to extract essential components such as tables, predicates, and projections [12, 7]. This enables their utilization as static features in incident management. Alternatively, some methods treat SQL queries as natural language, employing techniques such as Query2Vec [133] to capture their semantic meaning.

Heap dumps in Java memory analysis are another type of data that can be considered. They are snapshots of the Java heap memory taken at a specific moment in time. The Java heap is the region of memory where objects are allocated and deallocated during the execution of a Java application. A heap dump captures the complete state of the Java heap, including all objects and their attributes such as instance variables and references. This provides a detailed view of the memory usage within the Java application. Heap dumps are particularly useful for analyzing memory-related issues, such as memory leaks [140, 316]. In the related literature, heap dumps are commonly represented as trees [164], graphs [207], or hierarchies [249].

2.6 Evaluation Metrics

To comprehensively evaluate the quality and performance of data-driven approaches in incident management tasks, it is crucial to assess them using appropriate metrics, also known as figures of merit. While machine learning metrics are commonly used to evaluate predictive models, it is important to note that relying solely on these metrics, such as contingency metrics, may not accurately reflect the models' performance in real-world scenarios, especially when considering time constraints. For example, in the case of incident prediction, the goal is to predict incidents while minimizing false alarms and maximizing the coverage of actual incidents. However, predicting incidents after the designated prediction period (Δt_p) is not considered accurate since the incident has already occurred, leading to suboptimal allocation of time and resources. Therefore, we introduce a set of established metrics, focusing primarily on two main tasks: detecting and predicting incidents. It is worth mentioning that

Table 2.2: Contingency table.

	True Failure	True Non-failure	Sum
Prediction: Failure (Failure Warning)	True positive (TP) (Correct Warning)	False positive (FP) (False Warning)	Positives (POS)
Prediction: No failure (No Failure Warning)	False Negative (FN) (Missing Warning)	True Negative (TN) (Correctly no Warning)	Negatives (NEG)
Sum	Failures (F)	Non-Failures (NF)	Total (N)

these metrics can be adapted for other tasks as well, and several other metrics have been proposed to evaluate triage, diagnosis, and the effectiveness of automated remediation actions for restoring services and systems, which will be briefly covered.

To organize the metrics effectively, we categorize them based on the nature of the model output. This categorization includes metrics suitable for classification tasks such as Software Defect Prediction (SDP), scalar prediction, or regression tasks like Remaining Useful Lifetime estimation (RUL), as well as metrics for assessing the correlation between achieved results and reality or the trade-off between gain and complexity when comparing different methods, such as Fault Localization and Incident Correlation.

2.6.1 Classification Metrics

Classification metrics are commonly derived from four cases, as shown in Table 2.2. A prediction is classified as a true positive if an incident occurs within the prediction period and a warning is raised. Conversely, if no incident occurs but a warning is given, the prediction is considered a false positive. If the algorithm fails to predict a true incident, it is categorized as a false negative. Finally, if no true incident occurs and no incident warning is raised, the prediction is labeled as a true negative. To compute metrics like precision and recall, the contingency table is populated with the number of true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN). The prediction algorithm is applied to test data that was not used to determine the parameters of the prediction method. This allows the comparison of prediction outcomes against the actual occurrence of incidents. The four possible cases are illustrated in Figure 2.11. It's worth noting that the prediction period (Δt_p) is instrumental in determining whether an incident is counted as predicted or not. Therefore, the choice of Δt_p also has implications for the contingency table and should align with the requirements of subsequent steps in the incident management process.

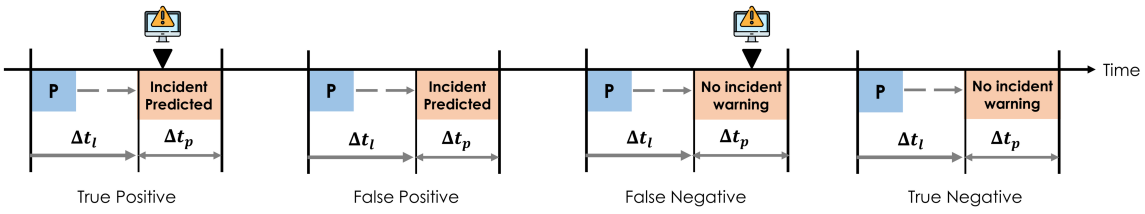


Figure 2.11: A timeline showing true incidents and all four types of predictions TP, FP, FN, TN.

Table 2.3: Metrics obtained from the contingency table.

Metric	Formula	Other names
Precision	$\frac{TP}{TP+FP}$	Confidence
Recall	$\frac{TP}{TP+FN}$	Support Sensitivity
False positive rate	$\frac{FP}{FP+TN}$	Fall-out
Specificity	$\frac{TN}{TN+FP}$	True negative rate
False negative rate	$\frac{FN}{FN+TP}$	1 - recall
Negative predictive value	$\frac{TN}{TN+FN}$	
False positive error rate	$\frac{FP}{FP+TP}$	1 - precision
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	
Odds ratio	$\frac{TP \times TN}{FP \times FN}$	

2.6.1.1 Contingency Table Metrics

The metrics presented in Table 2.3 are derived from the contingency table (see Table 2.2). They are commonly used in pairs, such as precision/recall, true positive rate/false positive rate, sensitivity/specificity, and positive predictive value/negative predictive value. Different research areas may use different names for the same metrics, so the leftmost column indicates the commonly used terminology, while the rightmost column lists alternative names.

Precision is the ratio of correctly identified incidents to the total number of predicted incidents. Recall, on the other hand, is the ratio of correctly predicted incidents to the total number of true incidents. For example, a prediction algorithm with a precision score of 0.8 correctly identifies incidents with a probability of 0.8 and produces false positives with a probability of 0.2. A recall of 0.9 indicates that 90% of true incidents are predicted, while 10% are missed. It's important to note that improving precision (reducing false positives) often results in a decrease in recall (increasing false negatives) and vice versa. To balance the trade-off between precision and recall, the F-Measure is used as the harmonic mean of the two, assuming equal weighting.

One limitation of precision and recall is that they don't consider true negative predictions. Therefore, it is necessary to consider other metrics in combination with precision and recall. The false positive rate is the ratio of incorrectly predicted incidents to the total number of non-incidents. A lower false positive rate is desirable, provided that the other metrics do not deteriorate. Specificity is the ratio of correctly not raised incident warnings to the total number of non-incidents, while the negative predictive value (NPV) is the ratio of correctly not raised incident warnings to the total number of not raised warnings. Accuracy is defined as the ratio of all correct predictions to the total number of predictions made.

Accuracy appears to be an appropriate metric for incident prediction due to the rarity of incidents. Achieving high accuracy by always classifying the system as non-faulty may be misleading because it fails to capture any incidents, resulting in a recall of zero. However, it is important to consider true negatives when assessing incident prediction techniques. Let's

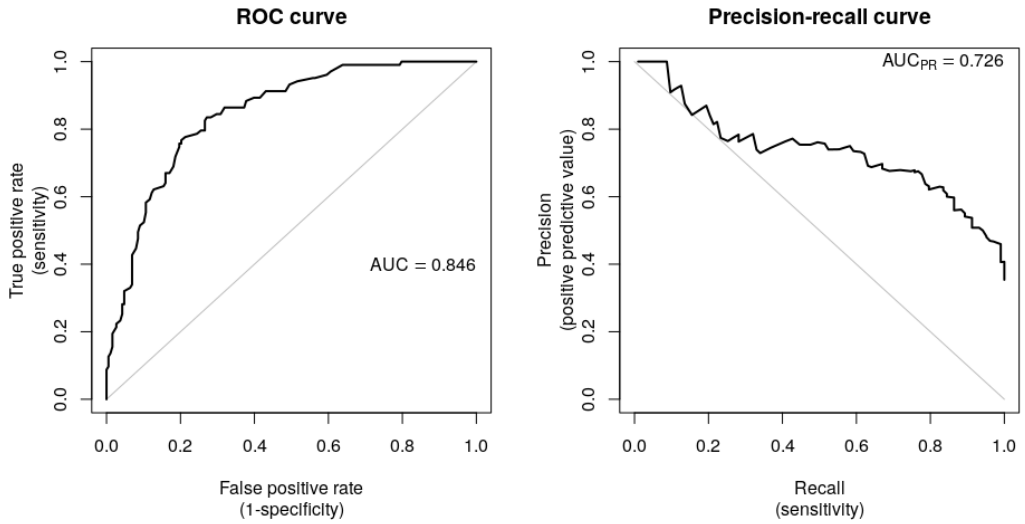


Figure 2.12: Example of Precision/Recall, and ROC curves.

consider an example from [260]. Two prediction methods perform equally well in terms of true positives (TP), false positives (FP), and false negatives (FN), resulting in the same precision and recall. However, one method makes ten times more predictions than the other because it operates on more frequent measurements. The difference between these methods is reflected only in the number of true negatives (TN), which becomes apparent in metrics that include TN. True negatives are counted by considering predictions made when no incident was imminent and no warning was issued.

It is noteworthy that the quality of predictions depends not only on algorithms but also on factors such as the data window size (Δt_d), lead-time (Δt_l), and prediction period (Δt_p). Predicting incidents at an exact point in time is highly unlikely, so predictions are typically made within a specific time interval (prediction period). The number of true positives is influenced by (Δt_p): a longer prediction period captures more incidents, increasing the number of true positives and impacting metrics like recall.

2.6.1.2 Precision and Recall-Curves

Incident predictors often utilize an adjustable decision threshold. When the threshold is set low, incident warnings are raised easily, increasing the chances of capturing true incidents (resulting in high recall). However, a low threshold also leads to many false alarms, resulting in low precision. Conversely, if the threshold is set very high, the situation is reversed. To visualize this trade-off, precision/recall curves are used, plotting precision over recall for various threshold levels. An example is shown in Figure 2.12 (right).

Similar to precision/recall curves, the receiver operating characteristic (ROC) curve (Figure 2.12 (left)) plots the true positive rate versus the false positive rate (sensitivity/recall versus 1-specificity, respectively). This curve assesses the model's ability to distinguish between incidents and non-incidents. The closer the curve is to the upper-left corner of the ROC space, the more accurate the model is. The Area Under the Curve (AUC) is defined as

Table 2.4: Metrics used for regression tasks in incident management tasks.

Metric	Formula
Mean Absolute Error (MAE)	$\frac{1}{n}\sum_{i=1}^{n}\lvert\hat{y}_i - y_i\rvert$
Root Mean Squared Error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$
Mean Absolute Percentage Error (MAPE)	$\frac{1}{n}\sum_{i=1}^{n}\left\lvert\frac{\hat{y}_i - y_i}{y_i}\right\rvert \times 100\%$
R-squared (R2)	$1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$
Explained Variance Score	$1 - \frac{\mathrm{Var}(y - \hat{y})}{\mathrm{Var}(y)}$

the area between the ROC curve and the x-axis. It is calculated as:

$$\mathrm{AUC} = \int_0^1 \mathrm{tpr}(fpr^{-1})\, dfpr$$

where tpr and fpr represent the true positive rate and false positive rate, respectively. The AUC measures the probability that a data point from an incident-prone situation receives a higher score than a data point from a non-incident-prone situation. By summarizing the capacity of a prediction algorithm to discriminate between incidents and non-incidents, the AUC converts the ROC curve into a single number. A random predictor has an AUC of 0.5, while a perfect predictor achieves an AUC of one.

2.6.2 Regression Metrics

Regression metrics are commonly used in tasks such as remaining useful lifetime estimation or anomaly detection, particularly when applied to time series metric data to identify outliers. Unlike classification metrics, the metrics used for regressors remain consistent across conventional machine learning models, as shown in Table 2.4. Mean Absolute Error (MAE) is a metric that represents the average absolute difference between the predicted values $\hat{y}_i$ and the actual values y_i. It provides a measure of the average magnitude of errors. Several works have utilized MAE for regression tasks [235]. Root Mean Squared Error (RMSE) is similar to MAE, but it takes the square root of the average squared differences between the predicted values and the actual values. RMSE penalizes larger errors more significantly. It has been used, for example, in anomaly detection [166]. Mean Absolute Percentage Error (MAPE) measures the average percentage difference between the predicted values and the actual values. It is particularly useful when evaluating the accuracy of predictions relative to the scale of the target variable [181]. R-squared (R2) is a metric used to represent the proportion of variance in the target variable that is explained by the predicted values. R2 ranges from 0 to 1, where a value of 1 indicates a perfect fit and a value of 0 indicates no improvement over a naive baseline [269]. Finally, the explained variance score, similar to R2, measures the proportion of variance in the target variable that is explained by the predicted values. However, it is based on the variance of the residuals and can be used as an alternative metric for evaluation [269].

2.6.3 Other Metrics

In addition to the commonly used regression and classification metrics, there exists a set of specialized metrics that are particularly relevant for evaluating the effectiveness and performance of specific incident management tasks, such as fault localization, incident correlation, and incident deduplication. These metrics are often specifically designed for incident management purposes and may have limited applicability outside of this domain. Some of these metrics have been developed in alignment with specific research studies, while others are unique to the field of incident management.

In fault localization research, several specific metrics are utilized to evaluate the effectiveness of different techniques. One such metric is T-Score, which estimates the percentage of code that a programmer can ignore before identifying the first faulty location in the program [253, 190]. Another metric commonly used is EXAM (Expense metric), which measures the percentage of program statements that need to be examined before encountering the first faulty statement [137, 139]. In addition to these metrics, other research work have also employed the Wilcoxon signed-rank test [300] as a statistical evaluation method. This test serves as an alternative to the paired Student's t-test when the assumption of a normal distribution in the population cannot be made. In their study, Wong et al. [302] utilize this test to compare the effectiveness of two techniques, denoted as α and β. The test examines the one-tailed alternative hypothesis that β requires the examination of an equal or greater number of statements compared to α. By determining the confidence level at which the alternative hypothesis can be accepted, one can assess whether technique α is statistically more effective than β.

In the field of software defect prediction, there are cases where it is more useful to evaluate the classes based on their predicted number of defects in a ranking manner. One approach to assess the performance of the prediction model is by calculating Spearman's correlation coefficient [273], as demonstrated in the study conducted by [74]. Another method used in this context is the cumulative lift chart, which compares the performance of two different models or strategies by plotting the cumulative gain against the number of cases. This approach has been also employed by [96].

In the domain of Intrusion Detection Systems within practical network settings, Mirheidari et al. [215] conducted a comprehensive comparison of alert correlation algorithms. The study aimed to evaluate the performance of these algorithms using both quantitative and qualitative measures. The quantitative assessment focused on accuracy, while the qualitative evaluation delved into additional aspects such as Extendibility and Flexibility. These aspects refer the algorithm's adaptability, localizability, and capacity to adjust to new conditions. Furthermore, the evaluation considered the algorithm's ability to parallelize tasks and the associated memory requirements. This evaluation aligns with the desirable attributes emphasized in Section 2.3, which outlines the desired characteristics for AIOps solutions in incident management.

In the context of incident triage and deduplication, specific performance metrics have been adopted to improve the evaluation of these processes. These metrics include Mean-reciprocal rank [72], Recall rate of order k [262], and Average Hit Ratio [6] [289, 148]. It is noteworthy that there are some metrics specifically designed to assess the process rather than a particular algorithm or data-driven approach. While we previously discussed metrics such as Mean Time To Report (MTTR), Mean Time To Engage (MTTE), Mean Time to Diagnose (MTTD), and

Mean Time to Mitigate (MTTM), other research works [64, 242] have introduced alternative terminology. For example, MTTR may be referred to as Mean Time to Repair, and MTTD as Mean Time to Detect. Furthermore, there are additional valuable measures that have been considered, such as Mean Time Between Failures (MTBF) [45], which evaluates the average duration between consecutive incidents or failures to assess system reliability.

2.7 Review of AIOps Approaches for Incident Management

As mentioned earlier, the scattered nature of AIOps solutions poses challenges when it comes to comparing and applying them effectively. Consequently, several review papers have concentrated on specific tasks such as anomaly detection [123, 335, 54, 70], failure prediction [260, 78, 96, 36], incident triage [6, 326], and root cause analysis [303, 215, 272]. While it is worth noting that some research studies, similar to ours, have conducted comprehensive reviews of the incident management process (e.g., [232, 256, 40]), they may not cover all the necessary tasks or lack clear terminology, well-defined desiderata, and requirements for constructing an effective AIOps model. In the subsequent sections, we will delve into the most pertinent works for each task within the incident management procedure. For detailed mappings of these research works as well as review, experience, and use-case study papers to the categorization presented in Section 2.4, please refer to our Survey [250].

2.7.1 Incident Detection

Incident detection approaches are reactive methods of incident management that aim to track and identify abnormal states or behaviors in a system. Their purpose is to either anticipate failures before they occur or mitigate the consequences of failures after they have happened. This is driven by the understanding that, despite employing advanced prediction techniques, it is impossible to completely eliminate the occurrence of failures. These methods also aid in comprehending the causal relationships, as well as understanding the temporal characteristics that lead to incidents. Automated incident detection typically requires a variety of monitoring tools, ranging from basic print statements (which form the fundamental unit of system logs) to more complex instrumentation techniques or entire frameworks [232]. Incident detection methods often leverage unsupervised learning approaches, primarily because acquiring high-quality, sufficient, and balanced data labels poses significant challenges. Among the notable techniques employed in this context, we find clustering methods, dimensionality reduction techniques, and auto-encoders. Additionally, other approaches such as Graph Mining and Statistical Models, have been leveraged in this context.

At the technical layer, specifically at the network level, Lakhina et al. [159] propose an anomaly detection method for network traffic analysis using SNMP data. The authors apply Principal Component Analysis (PCA) to link flow measurements collected over time to separate traffic into normal and anomalous subspaces. Anomalies are identified by reconstructing new observations using abnormal components. If the reconstruction error exceeds a predefined threshold based on explained variance, the data point is considered anomalous. In a subsequent work [160], a similar approach incorporating additional features such as source and target destination address or port is proposed, using traffic feature distribution entropy. More recently, Deng and Hooi [84] addresses the challenge of detecting anomalous events in

the context of cyber security attacks by analyzing high-dimensional time series data, specifically sensor data. The authors propose an approach that combines structure learning with graph neural networks. Furthermore, they leverage attention weights to provide explainability for the detected anomalies.

At the hardware level, accurate methods for anomaly detection rely on analyzing multivariate time series system metrics using auto-encoders. Multi-Scale Convolutional Recurrent Encoder-Decoder (MSCRED) [325] constructs multi-scale signature matrices to represent different system statuses at various time steps. It utilizes a convolutional encoder to capture inter-sensor correlations and an attention-based Convolutional Long-Short Term Memory (ConvLSTM) network to identify abnormal time steps by capturing temporal patterns. Similarly, OmniAnomaly [275] focuses on capturing normal patterns in multivariate time series using Variational auto-encoders (VAE), by learning robust representations through techniques like stochastic variable connection and planar normalizing flow. This approach provides interpretations for detected entity anomalies based on the reconstruction probabilities of constituent univariate time series. USAD [19] employs adversarial training in the auto-encoder architecture to effectively isolate anomalies while maintaining fast training speeds. Experimental results focus also on the scalability and robustness of the approach. On the other hand, SR-CNN [252] has been the first attempt to propose an approach that combines Spectral Residual (SR) and Convolutional Neural Network (CNN) techniques.

CloudPD [267] introduced conventional machine learning techniques for anomaly detection in cloud environments, addressing both the application and functional layers. The approach involved utilizing various measures at the virtual machine (VM) and application machine levels, including operating system variables and application performance metrics. The paper proposed three unsupervised machine learning methods: k-nearest neighbors (k-NN), Hidden Markov Models (HMMs), and k-means clustering. In the context of anomaly detection for seasonal key performance indicator (KPI) time series, Donut [317] employs deep Variational Autoencoders (VAEs) and provides solid theoretical explanations. The Donut framework can operate effectively in both unsupervised and semi-supervised settings. Notably, the authors emphasize the importance of model explanations and propose a novel interpretation in the latent z-space to enhance understanding of the model outcomes. F-Fade [55] is an approach to detect anomalies in edge streams, which are commonly used to capture interactions in dynamic networks. It proposes a frequency-factorization technique to model the time-evolving distributions of interaction frequencies between node pairs. This approach has proven to effectively operate in an online streaming setting while requiring constant memory. In a recent study, Xie et al. [314] introduced Trace-VAE as a solution for handling microservice traces. They proposed a novel approach using a dual-variable graph variational autoencoder to effectively model the intricate structures of the traces and detect anomalies.

Log-based approaches have emerged as effective methods for detecting anomalies in functional and business services. Fu et al. [103] proposed a clustering-based technique in which log entries were mapped to their corresponding template versions, enabling the identification of line templates. Subsequently, a finite state machine (FSM) was learned to model program workflows based on the log evidence. This FSM-based model facilitated the verification of correct program execution and the detection of software problems. In another study by [318], text analysis and information retrieval techniques were applied to console logs and source code for anomaly detection in large-scale data centers. State variables and object identifiers

were automatically extracted from parsed logs, and their frequency across different documents was analyzed using principal component analysis (PCA) and term inverse-document frequency (TF-IDF). Anomalies were detected using a threshold-based rule on the reconstruction error. More recent work has focused on sequential recurrent neural networks (RNNs). DeepLog [91] introduced the use of long short-term memory (LSTM) networks to learn patterns from logs and predict the probability distribution of the next log key based on the observation of previous log keys. An abnormal log key was identified if it did not appear in the top-k keys ranked by probability. The paper also proposed an online learning strategy based on user feedback and attempted to provide explanations using a finite-state machine. LogAnomaly [209], while adopting the language model approach, employed template embeddings (template2vec) to extract semantic information and automatically match and merge similar log keys. This process eliminated the need for human feedback. LogRobust [330] addressed the challenge of log instability caused by changing log statements and noise in log processing. It utilized LSTM layers, semantic vectors, and attention mechanisms to compute an anomaly score directly, rather than predicting the next probable log keys.

2.7.2 Incident Prediction

Incident prediction approaches are proactive methods designed to prevent failures by addressing both static aspects, such as source code, and dynamic aspects, such as the availability of computing resources. The ultimate objective is to suggest preventive measures or take immediate actions as early as possible. These strategies vary extensively regarding the taxonomy we proposed (i.e., the scoop, data used, area of application, etc.).

Software Defect Prediction. SDP is a method used to estimate the likelihood of encountering a software bug within a functional unit of code, such as a function, class, file, or module. The core assumption linking SDP to failure occurrence is that code with defects leads to errors and failures during execution. Traditionally, defect-prone software is identified using code metrics to construct defect predictors, as discussed in 2.5. Nagappan et al. [223] propose an SDP approach based on code complexity metrics. However, they highlight the challenge of multicollinearity among these metrics, making the problem more complex. To address this, they employ Principal Component Analysis (PCA) to obtain a reduced set of uncorrelated features and use linear regression models for post-release defect prediction. Menzies et al. [211], shift their focus from static code metrics to the choice of prediction models, advocating for the use of Naive Bayes with logarithmic features. While early SDP contributions primarily focused on single-release perspectives, another category of works, known as changelog approaches [220, 236], concentrate on software history as a more influential factor for estimating defect density. In addition to introducing the benchmark SDP dataset AEEEM in [96], the authors compute various change-related metrics, such as the number of revisions, refactoring, and bug fixes per file, which are correlated with the number of future defects. Their approach relies on code entropy, churn, and a Learning to Rank (LTR) method. Nam et al. [224] address transfer learning for software defects by employing an extended version of Transfer Component Analysis (TCA) to learn common latent factors between source and target projects.

Critiques of traditional code metrics point out their handcrafted and simplistic nature. An alternative approach involves parsing the source code using ASTs. Wang et al. [295] question the ability of code metrics to capture semantics and distinguish between code regions with the

same structure but different semantics. They propose using latent semantic representations and training a Deep Belief Network (DBN) on AST-parsed code to learn semantic features. More recently, Li et al. [173] explore the use of Convolutional Neural Networks (CNN) for SDP. They extract a subset of AST nodes representing various semantic operations during parsing. These nodes are mapped to numerical features using embeddings and fed into a 1D convolutional architecture.

Software Aging and Rejuvenation. Software aging is a phenomenon whereby a software system gradually degrades in performance and reliability over time. Known causes of software aging include memory leaks and bloats, unreleased file locks, data fragmentation, and numerical error accumulation [51]. Garg et al. [106] propose a method for estimating the time-to-exhaustion of various system resources, including free memory, file, and process table sizes, and used swap space. They utilize regression techniques and seasonal testing to identify trends and quantify the exhaustion time. In a related study, Vaidyanathan and Trivedi [285] explore the impact of software aging resulting from the current system workload. They develop a semi-Markov reward model based on available workload and resource data, where different workload scenarios are represented as model states. The association to a specific state is determined using k-means clustering. To estimate the time-to-exhaustion of memory and swap space, a non-parametric regression technique is employed separately for each workload state. The challenge of non-linear and piece-wise linear resource consumption is tackled by [8] by utilizing an ensemble of linear regression models. These models are selected using a decision tree based on the same input features as the regression model, which consists of a combined set of hardware and software host metrics.

Hardware Failures Prediction. In large-scale computing infrastructures, ensuring hardware reliability is crucial for achieving service availability goals. However, due to the sheer number of components involved and the necessity to use commodity hardware in data centers, hardware failures pose a significant challenge. For example, Google has reported that 20-57% of disks experience at least one sector error over a 4-6 year period [213]. Hard drives are the most frequently replaced components in large cloud computing systems, and they are a leading cause of server failure [291]. To address this, hard-drive manufacturers have implemented self-monitoring technologies like SMART metrics in their storage products. In the approach presented by [336], Hidden Markov and Semi-Markov Models (HMM/HSMM) are used to estimate likely event sequences based on SMART metric observations from a dataset of around 300 disks (with approximately two-thirds being healthy). Two models, one trained from healthy disk sequences and the other from faulty disk sequences, are used to estimate the sequence log-likelihood at test time, with the class being determined by the highest score. Wang et al. [298] propose a similarity-based detection algorithm that selects relevant SMART features using Minimum Redundancy Maximum Relevance (mRMR) and projects the input data into a Mahalanobis space constructed from the healthy disk population. This approach aims to detect faulty disks that deviate more from the distribution. Xu et al. [315] introduce the use of Recurrent Neural Networks (RNNs) to model the long-term relationships in SMART data. Unlike binary classification approaches, their model is trained to predict the health status of disks, providing additional information on the remaining useful life and serving as a ranking approach. These approaches are typically used in an online setting after an offline training step. However, integrating additional data and updating the characteristics of faulty disks as new failures occur presents a challenge. To address this, the

approach presented by [312] proposes the use of Online Random Forests, a model that can adaptively evolve with changing data distributions through online labeling.

In terms of other hardware components, FailureSim [79] presents an approach for assessing the status of hardware in cloud data centers using multi-layer perceptrons and RNNs. Their method focuses on assessing 13 different host failing states associated with specific components such as CPU, memory, and I/O. On the other hand, Zhang et al. [328] address network switch failures. Their method, based on system log history, involves extracting templates from logs and correlating them with faulty behavior. They compare their extraction method and similar approaches for the extraction tasks and then train a Hidden Semi-Markov Model using the obtained templates.

Remaining Useful Lifetime Estimation. The Remaining Useful Life (RUL) is a crucial real-time performance indicator for operating systems during their operational lifespan. It signifies the time remaining until the system becomes no longer functional. Accurate RUL estimation is vital for planning condition-based maintenance tasks, aiming to minimize system downtime. Numerous data-driven approaches have been proposed to model the intricate behavior of system components. According to [36], Long Short-Term Memory (LSTM) is considered a highly suitable tool for handling dynamic data in RUL problems. For example, in a study by [337, 308], a vanilla LSTM model was employed to predict the RUL of aircraft engines. In another research conducted by [203], a hybrid approach combining Convolutional Neural Networks (CNN) and LSTM was used. Reinforcement Learning has also been utilized to enable models to update themselves based on learned experiences, even from incorrect decisions. Simulation environments are particularly advantageous in this context. For instance, in [31], a Transfer Learning approach was developed. This approach learned from states, actions, and rewards to generate an optimal reward policy. These algorithms focus on sequentially predicting RUL for a specific type of pumping system.

Software Failure Prediction. Predicting system failures from an application perspective involves exploring potential failures that may occur in various aspects, such as jobs, tasks, processes, VMs, containers, or nodes. Existing approaches tackling this problem predominantly rely on system metrics, service states, and topology. For instance, Cohen et al. [69] propose an approach based on Tree-augmented Bayesian Networks (TANs) to associate observed variables with abstract service states. This allows for forecasting and detecting Service Level Objective (SLO) violations and failures in three-tiered Web services. The system observes system metrics like CPU time, disk reads, and swap space, modeling their interdependencies. Through a greedy strategy, the optimal graph structure, including the most relevant input metrics, is selected. While originally developed for detection, this approach can also be utilized for diagnosing failures due to the interpretability properties of TANs. A framework for predicting system availability in datacenters has been proposed by [53], utilizing auto-regressive models and fault-tree analysis. This framework detects component-level symptoms, such as high CPU temperature, bad disk sectors, and memory exhaustion, which serve as the leaves of the fault tree. By considering these symptoms and the tree structure, a model of dependencies in combinational logic determines the availability state deterministically, allowing for the tracking of errors before they lead to failures. The HORA prediction system [241] adopts a holistic approach by leveraging architectural knowledge in conjunction with online KPIs data to predict Quality of Service (QoS) violations and service failures in distributed software systems. Bayesian Networks are employed to establish component depen-

dency and failure propagation models. These models associate component failures, predicted from system metrics using auto-regressive predictors, with system-wide problems. LSTM networks are also utilized in a comprehensive characterization study conducted by [132] on a workload trace dataset from Google. In this study, failures are predicted at the job and task level, where a job consists of multiple tasks, with each task representing a single-machine command or program. Failures are predicted based on resource usage, performance data, and task information, including completion status, and user/node/job attributes.

2.7.3 Incident Prioritization

As a large number of incidents can be reported simultaneously, it becomes time-consuming and resource-intensive to handle all of them at once. However, certain incidents require immediate attention due to their importance or severity. To address this issue, various data-driven approaches have been proposed to rank incidents or alerts based on prioritization factors. Some of these techniques can also be applied to other scenarios, as they are not solely dedicated to ranking but also involve detection and diagnosis mechanisms. For instance, Tian et al. [282] introduce a machine learning-based approach that recommends priority levels for bug reports by considering factors such as temporal information, textual content, author details, related reports, severity, and product information. These factors are extracted as features, which are then used to train a discriminative model capable of handling ordinal class labels and imbalanced data. In another study by [60], a large-scale empirical analysis of incidents collected from 18 real-world online service systems at Microsoft reveals that many incidents are considered insignificant and not prioritized for immediate resolution, even after identifying their root cause. These incidents are referred to as *Incidental Incidents*. To address this issue, the authors propose DeepIP (Deep learning based Incident Prioritization), which utilizes historical incidents containing incident descriptions and topology information to prioritize incidents. DeepIP employs an attention-based Convolutional Neural Network (CNN) specifically designed to identify incidental incidents.

To tackle the problem of Threat Alert Fatigue, Hassan et al. [120] present NODOZE, a ranking system that identifies suspicious activities. The volume of generated alerts often exceeds the capacity of cyber analysts to investigate them, resulting in the risk of missing true attack alerts among false alarms. NODOZE constructs a causal dependency graph for each alert event using contextual and historical information of threat alerts. Anomaly scores are assigned to the edges based on the frequency of related events occurring in the enterprise, and these scores are propagated along neighboring edges using a novel network diffusion algorithm. The aggregate anomaly score is then used for triage. Zhao et al. [334] propose AlertRank, an automatic and adaptive framework for identifying severe alerts. AlertRank utilizes a range of powerful and interpretable features, including textual and temporal alert features, as well as univariate and multivariate anomaly features for monitoring metrics. The XGBoost ranking algorithm is employed to identify severe alerts among all incoming alerts, and novel methods are used to obtain labels for training and testing.

2.7.4 Incident Assignment

Numerous data-driven approaches have been proposed to optimize the process of incident assignment by automatically assigning incidents to the appropriate service team and individ-

ual. Typically, these approaches involve training a classifier using historical incident reports that contain textual information, topology data, or prioritization scores. The trained classifier is then used to assign new incidents. In a semi-supervised text classification approach presented by [319], the authors address the issue of limited labeled incident reports in existing supervised methods. They combine a naive Bayes classifier with expectation-maximization, leveraging both labeled and unlabeled incident reports. By iteratively labeling unlabeled incident reports, they improve the classifier's performance. Additionally, they employ a weighted recommendation list that considers the weights of multiple developers during classifier training to enhance performance. More recently, Lee et al. [165] were the first to propose the use of a Convolutional Neural Network (CNN) and pre-trained word embeddings for incident assignment. In another study, Xi et al. [309] propose iTriage, an incident assignment approach that considers three crucial aspects: textual content, metadata (topology), and tossing sequence of incident reports. They utilize a sequence-to-sequence model to jointly learn features from textual content and tossing sequence, followed by a classification model that integrates features from textual content, metadata, and tossing sequence. DeepCT [59] is another approach highlighting that incident triage is a continuous process that incorporates intensive discussions among engineers. They use a GRU-based model with an attention-based mask strategy and a revised loss function to incrementally learn knowledge from discussions and update incident triage results.

DeepTriage [238] have been proposed to address incident assignment challenges such as imbalanced incident distribution, diverse input data formats, scalability, and gaining engineers' trust in the incident assignment process. The approach combines multiple machine learning techniques, including gradient-boosted classifiers, clustering methods, and deep neural networks, in an ensemble to recommend the responsible team for the incident.

2.7.5 Incident Classification

As discussed earlier, the primary objective of incident classification is to enhance the diagnosis of incidents, thereby centralizing the efforts of the maintenance team. However, this category has often been overlooked in the review process and is generally associated with deduplication, triage, or prioritization. Nevertheless, there exist research works that align perfectly with this category, offering approaches to organize a large volume of incidents or alerts into representative sets of issues or topics. One such study by [332] addresses the handling of alert storms. Their approach comprises two stages: alert storm detection and alert storm summary. In the alert storm summary stage, irrelevant alerts are filtered out using an alert denoising method that learns alert patterns from the system's normal states. Alerts reflecting service failures are then clustered together based on textual and topological similarities. From each cluster, the most representative alert is selected, forming a concise set of alerts for investigation. Another notable contribution, presented by [183], proposes a Hidden Markov Random Field (HMRF) based approach for automatically identifying recurrent performance issues in large-scale software systems. Their approach formulates the problem as an HMRF-based clustering problem, which involves learning metric discretization thresholds and optimizing the clustering process.

In the context of classifying textual incident reports, Xia et al. [310] introduce a novel framework for bug triaging that utilizes a specialized topic modeling algorithm called multi-feature topic model (MTM). MTM extends Latent Dirichlet Allocation (LDA) by considering

product and component information from bug reports, effectively mapping the term space to the topic space. They also introduce an incremental learning method named TopicMiner, which leverages the topic distribution of a new bug report to assign an appropriate fixer based on the fixer's affinity to the topics. Yang et al. [321] also employ LDA to extract topics from bug reports and identify related bug reports for each topic. Their approach first determines the topics of a new bug report and then utilizes multiple features (such as component, product, priority, and severity) to identify similar reports that share the same set of features as the new bug report.

2.7.6 Incident Deduplication

Incident deduplication detection aims to identify the most similar incidents among a set of historical incidents, which exhibit slight differences but primarily address the same problem. This work can be categorized into basically two main categories.

The first category focuses on presenting techniques for detecting duplicate incident reports using their descriptions and characteristics. Hiew [126] made the initial attempt to detect duplicate bug reports based on the textual information of incident reports. His approach transforms the textual part of incident reports into word vectors, calculates the similarity between them, and ranks candidate duplicate incident reports based on their similarity to a given incident. Runeson et al. [257] conducted another pioneering work where they considered additional textual features such as software versions, testers, and submission dates, and performed large-scale experimental studies on industrial projects. Sureka and Jalote [278] proposed a novel approach to detecting duplicate incident reports based on N-grams. Additionally, Banerjee et al. [24] proposed considering word sequences when calculating the textual similarity between incident reports.

The second category primarily relies on designing similarity metrics that can reflect the semantic crash similarity between execution reports, specifically stack traces. Lerch and Mezini [171] employed the TF-IDF-based scoring function from Lucene library [196]. Sabor et al. [258] proposed DURFEX system which uses the package name of the subroutines and then segment the resulting stack traces into N-grams to compare them using the Cosine similarity. Some alternative techniques propose to compute the similarity using derivatives of the Needleman-Wunsch algorithm [228]. In [48], the authors suggested adjusting the similarity based on the frequency and the position of the matched subroutines. Dang et al. [75] proposed a new similarity measure called PDM in their framework Rebucket to compute the similarity based on the offset distance between the matched frames and the distance to the top frame. More recently, TraceSim [289] has been proposed to take into consideration both the frame position and its global inverse frequency. Moroo et al. [219] present an approach that combines TF-IDF coefficient with PDM. Finally, we outline some earlier approaches that used edit distance, as it is equivalent to optimal global alignment [26, 216].

2.7.7 Root Cause Analysis

Root cause analysis approaches aim to determine the underlying faults that give rise to software bugs, errors, anomalies, or hardware failures. More concretely, root cause analysis is a diagnostic task that needs to be performed when reporting incidents, either after or in parallel with the triage process. In complex systems, it is necessary to first isolate and restrict

the analysis to the faulty component or functionality, a process known as Fault Localization. Specifically, Fault Localization involves identifying a set of components (devices, hosts, software modules, etc.) that serve as the initial trigger for an error within the system. It is important to note that fault localization can operate at different layers, including the physical, application, and functional layers. To simplify, we can differentiate between two types of fault localization, technical fault localization (which encompasses network and hardware layers) and software fault localization (which includes functional and business layers). Software fault localization centers around the analysis of the source code, regardless of whether the software in question is deployed on numerous or just a few machines. Additionally, we explore other techniques for root cause diagnosis.

Technical Fault Localization. FChain [230] is a fault localization system designed for pinpointing faulty components in an online cloud environment. It operates as a black-box solution, relying on low-level system metrics to detect performance anomalies. Components displaying anomalies are then sorted based on manifestation time and examined sequentially using a discrete Markov model. Various techniques, such as analyzing interdependencies between components and studying the overall propagation trend, are used to filter out spurious correlations. Hotspot [277], on the other hand, utilizes Monte Carlo Tree Search (MCTS) to efficiently explore attribute combinations and measure their correlation with sudden changes in the Page View metric. Similarly, Squeeze [185] proposes a comparable method using a combined top-down and bottom-up search strategy. It introduces a novel correlation metric called Generalized Potential Score (GPS). These enhancements enable Squeeze to localize root causes even in cases with lower statistical significance. Li et al. [179] also apply a pattern mining approach to structured logs in order to discover association rules of the form $X \longrightarrow Y$, using the FP-growth mining algorithm. Here, Y represents a predefined attribute combination that describes a failure. The authors also present five different use cases that are applicable in large-scale service infrastructures. With Sherlock, [23] focus on localizing the sources of performance problems in enterprise networks by constructing probabilistic inference graphs based on the observation of packets exchanged within the network infrastructure. The nodes of the inference graph are divided into three types: root cause nodes (corresponding to internal IP entities), observation nodes (corresponding to clients), and meta-nodes that model the dependencies between the other two types. Each node is associated with a categorical random variable that represents its current state (up, troubled, down), which is influenced by other nodes through dependency probabilities. The inference graph is learned by observing the packets exchanged between nodes during normal operation. Once the graph is constructed, measurements from the observation nodes can be utilized to obtain a set of state-node assignment vectors, which correspond to the estimated operational state of the network.

Software Fault Localization. A software fault localization approach typically yields a set of suspicious statements or components. In contrast to SDP, this method relies on observed failure patterns obtained from production runs and unit tests, rather than predictions of the likelihood of a code component entering a defective state.

Numerous approaches have been developed to tackle the problem of software fault localization. One prominent category is program spectrum-based techniques, which rely on the similarity between program execution profiles obtained from execution traces. These profiles represent both successful and faulty runs of programs. For example, Renieres and Reiss [253]

employ nearest neighbor search to compare a failed test with a similar successful test, using the Hamming distance as a measure of similarity. Another well-known technique, Tarantula [138], utilizes coverage and execution results to compute the suspiciousness score of each statement. This score is based on the number of failed and successful test cases covering the statement, as well as the total number of successful and failed test cases. Subsequent research in this field has proposed refinements to the suspiciousness scoring, such as Ochiai [1], Crosstab [301] and DStar [302].

Statistical debugging approaches have also been explored. Liu et al. [190] introduce a statistical debugging method called SOBER, which analyzes predicate evaluations in failing and passing runs. By estimating the conditional probability of observing a failure given the observation of a specific predicate, the approach identifies predicates with higher probabilities, indicating their potential involvement in software bugs or their proximity to them. Abreu et al. [2] later proposes a Bayesian reasoning approach known as BARINEL, which incorporates a probabilistic framework for estimating the health probability of components. This model, based on propositional logic, captures the interaction between successful and failed components. Furthermore, data mining techniques have shown promise in fault localization due to their ability to unveil hidden patterns in large data samples. Cellier et al. [52] discuss a combination of association rules and Formal Concept Analysis as a means to assist in fault localization. This technique aims to identify rules that associate statement coverage with corresponding execution failures, measuring the frequency of each rule. A threshold is set to determine the minimum number of failed executions covered by a selected rule. The generated rules are then partially ranked using a rule lattice, and the ranking is examined to locate the fault.

Other RCA Techniques. Many other approaches have been proposed to assist in the diagnosis of root causes to cope with the challenging inherent complexity and inter-dependency between components in software systems. For instance, X-Ray [14], is a tool that addresses the challenge of troubleshooting performance issues by providing insights into why certain events occurred during performance anomalies. The contribution of the described technique is indicated as performance summarization, which works by instrumenting binaries while the applications execute. It attributes performance costs to each basic block and utilizes dynamic information flow tracking to estimate the likelihood that a block was executed due to each potential root cause. It then summarizes the overall cost of each potential root cause by aggregating the per-block cost multiplied by the cause-specific likelihood over all basic blocks. The technique can also be used differentially to explain performance differences between two similar activities. Another work conducted by [261] utilizes Hierarchical Hidden Markov Models (HHMM) to associate resource anomalies to root causes in clustered resource environments, on the different levels of container, node, and cluster. Markov models are constructed on different levels, trained with the Baum-Welch algorithm using response time sequences as observations.

2.7.8 Incident Correlation

Research on incident correlation typically focuses on studying correlations between alerting signals, occurring incidents, or alerting signals and incidents. Existing correlation algorithms primarily analyze the correlation of raw key performance indicators (KPIs) or transform the KPIs into events and analyze their correlation. For instance, [193] construct dependency re-

lationships among system components by mining co-occurrence patterns in log events using association rule mining algorithms. Luo et al. [199] formulate the correlation problem as a two-sample problem to assess the correlation between KPI time series and event sequences in online service systems. They employ the nearest neighbors method to evaluate the existence of the correlation and analyze temporal relationships and monotonic effects Luo et al. [199]. Wu et al. [306] propose a methodology and efficient algorithm for discovering leaders among a set of time series based on lead-lag relations. By analyzing lagged correlations and constructing a graph-based representation, they compute a leadership rank that quantifies the influence of each time series. CoFlux [276] is an unsupervised approach for correlating Key Performance Indicators (KPIs) in internet service operations management. CoFlux addresses the challenge of separating fluctuations from normal variations in KPIs with different structural characteristics and determines correlation, temporal order of fluctuations, and directional consistency between KPIs using robust feature engineering.

Correlating alerts has been an important concern in Intrusion Detection Systems. For example, GhasemiGol and Ghaemi-Bafghi [108] utilize alert partial entropy to determine the probability of alerts indicating the same information. Density-based spatial clustering of applications with noise (DBSCAN) is used to group alerts. Tan et al. [279] propose a multivariate correlation analysis system for attack detection by extracting geometric correlations between characteristics of network traffic. Their solution employs anomaly-based detection, focusing on legitimate network traffic patterns, and utilizes the Mahalanobis distance to measure similarity between traffic records. Bateni and Baraani [27] present an Enhanced Random Directed Time Window (ERDTW) alert selection policy based on sliding time windows analysis. ERDTW classifies time intervals into relevant (safe) and irrelevant (dangerous) based on attributes described in mathematical logic rules and expressions. For example, if a time interval contains numerous alerts with the same IP address, it is more likely to be flagged as dangerous.

2.7.9 Incident Mitigation

Through the triage and diagnosis steps of incident management, valuable knowledge is gained, including the identification of incident scope, retrieval of historical duplicates, and analysis of root causes. This knowledge enables the initiation of automatic repair actions known as mitigation or remediation actions. Incident mitigation has received less attention compared to reporting and diagnosis tasks, as it is often a consequence of the outcomes of those processes. Once the underlying problem has been clarified through diagnosis, the recovery steps become readily identifiable and attainable without the need for complex models. However, our commitment extends to providing a list of research works that focus on resolution tasks, even when triage or diagnosis are involved.

In a study conducted by [338], similarity-based algorithms are proposed to suggest resolutions for recurring problems based on incident tickets. The approach retrieves k suggestions for ticket resolution using a k-NN approach. Similarity between tickets is evaluated using a combination of numerical, categorical, and textual data, with individual and aggregate similarity measures defined. The solution is further extended to address false-positive tickets in both historical and incoming data. This is achieved by classifying tickets as real or false using a binary classifier and weighing ticket importance based on the prediction outcome. The final solution recommendation considers both importance and similarity. The paper also explores

ideas for improving feature extraction, such as topic discovery and metric learning. Wang et al. [293] propose a cognitive framework based on ontologies to construct domain-specific knowledge and suggest recovery actions for IT service management tickets. The approach involves analyzing free-form text in ticket summaries and resolution descriptions. Domain-specific phrases are extracted using language processing techniques, and an ontology model is developed to define keywords, classes, relations, and a hierarchy. This model is then utilized to recommend resolution actions by matching concept patterns extracted from incoming and historical tickets using similarity functions like the Jaccard distance. In a work conducted by Meta [184], natural language processing techniques are employed to predict repair actions for hardware failures based on closed incident tickets. Through the analysis of raw text logs, up to five repair actions are recommended. Ding et al. [89] propose an automated mining-based approach for suggesting appropriate healing actions. The method involves generating signatures of an issue using transaction logs, retrieving historical issues based on these signatures, and suggesting a suitable healing action by adapting actions used for similar past issues.

2.8 Discussion

As mentioned in 1.2, despite the existence of a wide range of AIOps solutions proposed to address various incident management tasks using advanced artificial intelligence techniques, certain limitations and gaps have been identified. In this thesis, we aim to address these gaps by applying specialized techniques in specific use-cases, which can be easily extended and transferred to other domains. One such technique is Subgroup Discovery, a promising data mining approach employed to tackle data quality issues, including imbalanced, noisy, and complex data. By utilizing Subgroup Discovery, we propose an alternative approach to detect and diagnose incidents, moving away from conventional classifiers that may require revision and can be less effective when deployed in real-world industrial scenarios. Additionally, we incorporate new techniques of explainable artificial intelligence, considering the complexity of explanations to avoid overwhelming users. This ensures that the explanations provided are both interpretable and actionable. Addressing performance challenges, we focus on a specific issue related to crash report deduplication triage, driven by industrial needs. To tackle this problem efficiently, we employ a fast and effective retrieval technique called Locality Sensitive Hashing embedded into a deep Siamese neural network to support a wide range of crash bucketing similarity measures.

Chapter 3

Introduction to Subgroup Discovery with a Practical Application to SQL Workload Analysis

Subgroup Discovery and its generalization, Exceptional Model Mining, aim to provide sophisticated frameworks for the extraction of insightful and interpretable patterns from vast datasets characterized by abnormal distributions in relation to the overall data. These frameworks are designed to address specific target problems within specialized contexts. Notably, they have demonstrated remarkable efficiency in handling complex, diverse, and extensive datasets, while offering flexibility in user interaction and the integration of domain-specific knowledge. Despite their successful application in diverse domains such as physics, education, and neuroscience, it is surprising that their utilization in the context of AIOps for incident management tasks has received relatively less attention compared to predictive models. This is particularly astonishing considering the inherent challenges posed by data quality, complexity, and the difficulties in training accurate models in real-production scenarios. Therefore, our primary objective is to formally introduce and provide an overview of this domain, emphasizing its relevance in the field of AIOps. Firstly, we establish the foundational elements that are crucial for tackling any related data mining task. These definitions will serve as a solid foundation for the subsequent contributions presented in this thesis, with a particular focus on Chapters **3**, **4**, and **5**. Subsequently, we explore a practical and direct application of these concepts to a real-world problem involving the identification of SQL data patterns that exhibit correlations with various performance degradation issues.

3.1 Introduction

"It is a very sad thing that nowadays there is so little useless information"[1]. This quote by Oscar Wilde serves as a reminder that the value of information in terms of its diversity, volume, and velocity has far surpassed what was witnessed over a century ago when its worth as a precious commodity was already acknowledged. When it comes to the new digital ERA and AIOps environments, enterprises are equipped with powerful and scalable tools capable of collecting enormous amounts of telemetry data, ranging from terabytes to even petabytes, on a daily or monthly basis. Within this landscape, every individual piece of data holds valuable potential that can drive decision-making processes. However, the diverse nature of data, with its disparate formats and structures, as well as the vast volumes involved, necessitates the use of effective tools to address the challenges of cleaning, imputing, normalizing, and fitting this data into suitable models in order to extract meaningful patterns.

In various AIOps scenarios, the construction of supervised machine learning models for anomaly detection, failure prediction, and fault localization is a common practice. However, AIOps encounters notable challenges related to data quality [76, 64, 40]. These challenges are highlighted by the absence of clear ground truth labels, highly imbalanced datasets with substantial noise, and other similar factors. Furthermore, even when these aforementioned challenges are addressed, obtaining a sufficient number of labels to learn "what is abnormal" in the context of AIOps can be difficult. This is primarily due to the continuous changes in system behavior, the complex dependencies and relationships among components and services within large-scale service systems, the frequent need for model updates, and the incorporation of human knowledge and trust in the process (commonly known as the "human in the loop" paradigm). Despite the numerous challenges faced by predictive models, it is noteworthy that significant efforts have been invested in the development of such models within the AIOps domain. Recently, there has been a growing emphasis on considerations such as interpretability and scalability. However, it is surprising that there has been relatively limited adoption of pattern mining techniques (e.g., Formal Concept Analysis [52, 89] and Frequent Pattern Mining [73, 179]) despite their capability to extract informative patterns from large datasets, which can assist in detection, diagnosis, and mitigation tasks. Descriptive models, particularly Subgroup Discovery [15, 162, 304], offer distinct advantages in addressing the challenges of data diversity, complexity, and quality, as well as providing flexibility for interactive mining in collaboration with human experts.

Subgroup Discovery is a significant component of Knowledge Discovery in Databases research field, which aims to tackle the common challenge of extracting relevant knowledge from vast amounts of raw data. Its primary objective is to identify subsets of data instances that exhibit compelling distributions in relation to a predefined target concept. Subgroup Discovery, also known as Supervised Rule Discovery, operates as both a descriptive and supervised technique. The term descriptive denotes that the results are intended for descriptive purposes, enabling interpretation by human experts within a specific context. On the other hand, the term supervised signifies that the user specifies the property of interest, the target, which serves as the focus of the knowledge discovery process. The Subgroup Discovery Process relies on the establishment of its fundamental pillars. Firstly, the pattern languages used to define contextualized data patterns are diverse and heterogeneous, using combina-

[1] Wilde, Oscar: A Few Maxims For The Instruction Of The Over-Educated, Saturday Review, 1894.

tions of itemsets, numerical vectors, graphs, and more. These patterns are developed in conjunction with a well-defined target concept. Secondly, given that the number of potential patterns grows exponentially with the input data, intelligent enumeration techniques become essential. These techniques involve exhaustive search techniques but also heuristic search, considering the exploration/exploitation trade-off. Their purpose is to generate a concise set of interesting patterns, guided by the concept of *interestingness* that assess the quality of retrieved patterns and guide the complexity of the search. Exceptional Model Mining [95], while operating within the same description space as SD, extends its capabilities by enabling the handling of multiple complex target concepts.

We leverage this promising data mining approach to address the analysis of large workloads of SQL queries effectively. Specifically, when it comes to database administration (DBA) tasks, the analysis of query workloads plays a crucial role in identifying schema issues and enhancing overall performance. While DBAs can easily pinpoint individual queries that consistently lead to performance problems, the challenge lies in automatically identifying subsets of queries that share specific properties and simultaneously impact target measures, such as latency time. In this context, patterns are defined based on combinations of query clauses, topology, alerting signals, and key performance indicator (KPI) metrics. These patterns assist in answering pertinent questions like "What factors contribute to slow SQL queries?" or "How can we characterize queries that excessively consume I/O communication?". The automated discovery of such patterns within an extensive search space, followed by presenting them as hypotheses, serves the purpose of assisting in issue localization and root cause analysis. This scenario aligns with the principles of Subgroup Discovery. In the following sections, we demonstrate how to instantiate and develop this versatile data mining framework to identify potential causes of SQL workload issues.

Roadmap. The remainder of this chapter is organized as follows. In Section 3.2, we start by providing formal definitions of the Subgroup Discovery task and delving into its various components in comprehensive detail. This includes an examination of the pattern syntax, including the dataset and target concept, as well as the search space and selection criteria, such as the measure of interestingness. To enhance understanding, we illustrate each component using relevant examples drawn from our specific use case. Additionally, we provide a brief overview of the distinctive features of Exceptional Model Mining. Continuing forward, Section 3.3 presents the framework we have implemented to analyze various performance issues pertaining to SQL workloads. We demonstrate in Section 3.4 the effectiveness of Subgroup Discovery in this context through both quantitative and qualitative experiments. To conclude the chapter, Section 3.5 provides discussion and potential avenues for future research

3.2 Overview of Subgroup Discovery

Subgroup Discovery is a technique for detecting local patterns [218] that has been formally coined by [304, 152]. However, the idea of discovering interesting subgroups in a database can be traced back to [270]. The objective of Subgroup Discovery is to identify subsets of objects within a dataset, known as subgroups, whose distribution of target labels statistically deviates from the overall dataset. In other words, these subgroups exhibit distinct patterns that differ from what is typically observed in the dataset as a whole. To provide a general definition of Subgroup Discovery, we refer to the work of [125]. While there may be slight

variations in the details of Subgroup Discovery definitions found in the literature [152, 15], there is a general consensus regarding the fundamental nature of the task.

Definition 1 (Subgroup Discovery)**.** In Subgroup Discovery, we assume we are given a so-called population of individuals (objects, customers, ... etc.) and a property of those individuals we are interested in. The task of Subgroup Discovery is then to discover the subgroups of the population that are statistically "most interesting", i.e. are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest.

Based on this definition, subgroups are generated by descriptions that capture the properties of individuals, providing interpretability and comprehensibility to the user. The definition further necessitates a statistically significant distribution of the property of interest. To assess the significance of this deviation, a quality measure is employed, taking into consideration both the subgroup's ability to generalize and its capacity to highlight intriguing deviations from the norm with respect to a chosen target concept. To further illustrate these concepts, let's consider a toy example dataset in Table 3.1 consisting of properties associated with a small set of SQL queries. As an example, the columns `Verrou` and `Cumulof` provide information about the queried database tables within the SQL query. Additionally, the column `Verrou.data` represents a predicate in the `WHERE` clause (further details on the parsing of SQL queries and their representation in a structured format can be found in Section 3.3). In addition to the query information itself, this dataset includes additional details such as the query topology (representing environmental features), alerting signals that occur during query execution, and a report of their executions. For convenience, the execution time is provided as a numerical value, along with a discrete value indicating whether the query is considered slow or not with the associated number of rows returned by each query.

Table 3.1: Illustrative dataset: execution report of SQL queries.

O	FROM		WHERE			ENV features		Alerts	ASH	q_{nrows}	q_{time}	
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
	`Verrou`	`Cumulof`	`Verrou.ik`	`Verrou.date`	`Cumulof.ik`	Soft version	Server name	manyActiveSessions	Concurrency	nrows	time	slow
o_1	1	0	1	0	0	v2	LYN	Alarm	22	10	2.15	0
o_2	1	0	1	1	0	v1	BLV	Critical	3	1	15.81	1
o_3	0	1	0	0	1	v1	BLV	Critical	15	27	1.14	0
o_4	1	1	0	1	1	v2	LYN	Alarm	31	12	10.87	1
o_5	1	1	1	0	1	v3	LYN	Alarm	11	25	2.1	0
o_6	1	0	1	2	0	v3	LYN	Critical	6	100	17.93	1
o_7	1	1	1	1	1	v2	LYN	Info	27	1	15.8	1
o_8	0	1	0	0	1	v2	BLV	Alarm	9	37	9.95	0
o_9	1	0	1	0	0	v3	BLV	Critical	10	112	8.95	0
o_{10}	0	1	0	0	1	v2	BLV	Alarm	7	1	14.7	1
o_{11}	0	1	0	0	0	v2	LYN	Info	25	16	1.0	0

Let's consider the task of characterizing slow queries. In this case, the column `slow` holds specific information that is of particular interest, and it serves as the target concept for Subgroup Discovery. The main goal is to identify subgroups of queries that exhibit common properties while maximizing the proportion of queries that are classified as slow. This objective aligns with the fundamental aim of Subgroup Discovery, which is to uncover interpretable connections between various characteristics (descriptive variables) and the specific property

of interest (in this example, slow queries), as discussed in [270]: "Clearly, the result of a data mining session should never be a listing of the members of such a subgroup. Rather, it should result in a (characteristic) description of the subgroup".

For instance, considering queries that include the attribute `Verrou.date`. It can be hypothesized that these queries tend to be slower compared to others. Remarkably, 100% of these queries are categorized as `slow`, while the overall proportion of slow queries is only 45%. This particular subgroup can be described by the pattern: (`Verrou.date` $\geqslant 1$). Another interesting subgroup consists of queries that belong to the software version v2 and include the attribute `Cumulof.ik` as a predicate. This subgroup can be described as a conjunction of two basic patterns: (`Cumulof.ik` $= 1 \wedge$ `Soft.version` $=$ v2). However, given the large number of attributes involved, the possible combinations of conjunctive patterns become vast. Consequently, it becomes challenging to identify the most significant and discriminant descriptions and present the corresponding statistics to a human expert. This is where an automatic Subgroup Discovery approach proves to be highly valuable. Such an approach conducts a deep search among candidate hypotheses and evaluates each of them using a function that measures their interestingness (e.g., the ratio of slow queries within the subgroup compared to the overall ratio).

In addition to the fully automatic approach, the involvement of human expertise can greatly enhance the Subgroup Discovery process, which is known for its iterative and interactive nature. For example, the pattern (`Verrou.date` $\geqslant 1$) can provide valuable insights to experts, suggesting that an index may be missing on the `Verrou.date` attribute. This information can prompt experts to take mitigation actions, such as adding an index, and then reapply the Subgroup Discovery approach to determine if the issue has been resolved.

While the previous examples focused on binary properties (slow or not), Subgroup Discovery can also handle numerical properties. For instance, considering the target concept as the numerical attribute `time`, a statistically interesting subgroup pattern could be (`Verrou` $\geqslant 1 \wedge$ `manyActiveSessions` $=$ Critical). This pattern indicates that queries with both conditions have an average `time` of 14.22s, which is considerably larger than the dataset's overall average of 9.12s. It is worth noting that if we had chosen the binary target `slow`, this particular subgroup would not have been as impressive, as it consists of only 3 queries, with one of them being slightly below the 10s threshold. Furthermore, the size of subgroups often plays a role in assessing their quality. Discriminant subgroups that cover a larger number of queries are typically considered more statistically significant.

More formally, the Subgroup Discovery task can be described by a quadruple $(\mathcal{D}, \mathcal{L}, T, Q)$. Here, $\mathcal{D}$ represents the dataset containing objects or individuals characterized by their descriptive features. $\mathcal{L}$ refers to the pattern language or pattern syntax, which represents the search space containing all potential candidate subgroup descriptions that can be formed from the dataset. The target concept T specifies the property of interest for the discovery task. Finally, Q indicates the selection criteria used for evaluating and filtering the candidate subgroups. These selection criteria can consist of constraints on the subgroups or generally an interestingness measure that scores the candidate subgroups based on factors such as the distribution of the target concept and the number of instances covered. Additionally, an optional parameter k can be utilized to determine the number of subgroup patterns to be returned. The result of the Subgroup Discovery task is either the complete set of subgroups in the search space that satisfy all the specified constraints or the top k subgroups based on the chosen interestingness measure. Figure 3.1 provides an overview of the key components

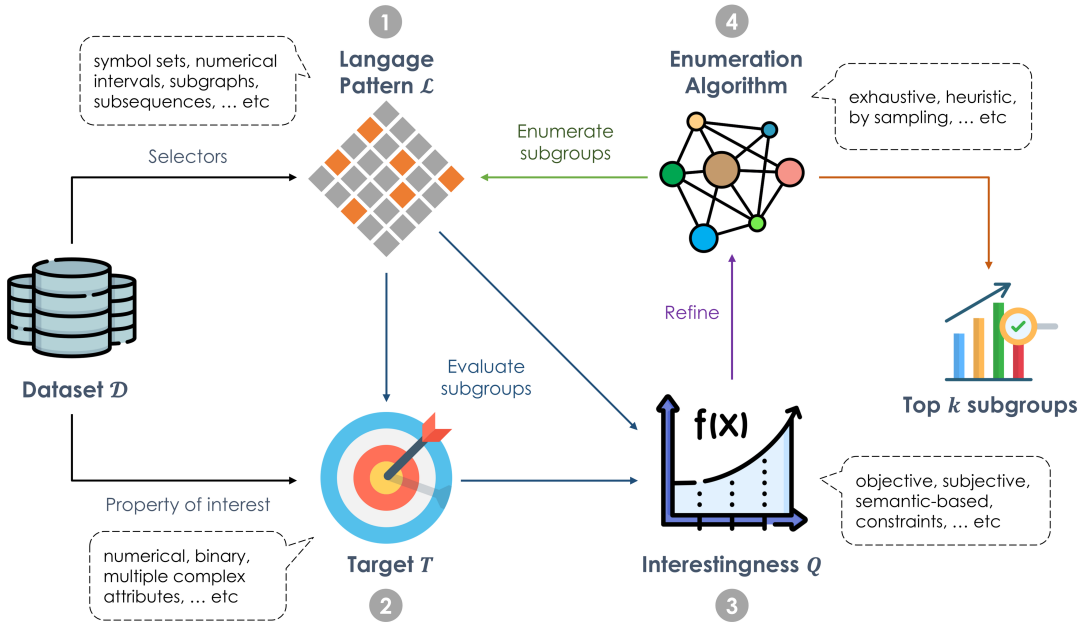


Figure 3.1: Building blocks of a Subgroup Discovery task (Summary).

involved in a Subgroup Discovery task, which we will delve into in the upcoming subsections.

3.2.1 Dataset

The different data sources are unified into a dataset $\mathcal{D} = (O, A)$ defined as an ordered pair of a set of instances (also called individuals, cases or data records) $O = \{o_i\}_{1 \leqslant i \leqslant n}$ and a set of attributes $A = \{a_j\}_{1 \leqslant j \leqslant m}$. Each attribute $a : O \longrightarrow dom(a)$ is a function that maps objects to values in its domain $dom(a)$. Consequently, $a(o)$ denotes the value of the attribute a for the object o. $dom(a)$ is given by $\mathbb{R}$ if a is numerical, by a finite set of categories C_i if a is nominal (categorical), or by $\{0, 1\}$ if a is Boolean. A nominal attribute with a total ordering of its values is called an ordinal nominal attribute. These notations are illustrated in Table 3.1 with a dataset of 11 objects $O = \{o_1, ..., o_{11}\}$ referring to queries described by 12 attributes. For example, `Server.name` is a nominal attribute with two possible values: LYN and BLV. `manyActiveSessions`, which corresponds to an alert, is an ordinal attribute with three levels: Info, Alarm, Critical. `time` is a numerical attribute that indicates the execution time of a query. `slow` is a binary attribute, taking the value 1 when the query time exceeds 10 seconds and 0 otherwise. The queries in the table are parsed to track the occurrence of each token associated with each clause. Consequently, each token is represented as a numerical attribute $a_j \in A$ where $\text{dom}(a_j) = \mathbb{N} \subset \mathbb{R}$. For instance, `Verrou.data` is a numerical attribute that denotes the number of times the `Verrou.data` attribute appears in the `WHERE` clause.

The definition of Subgroup Discovery presented above is not limited to data structured in a single table. In the previous discussion, we made use of the single-table assumption [305], where all the data is contained within a single table, with instances represented as rows and their characteristics (attributes) as columns. This assumption simplifies the analysis

for many practical applications. However, several Subgroup Discovery algorithms have been developed specifically for the multi-relational setting, where data is distributed across multiple tables. In Chapter 5, we will address this scenario when our target concept is organized with multiple attributes hierarchically. Nonetheless, it is possible to transform data consisting of multiple tables into a single table through propositionalization and aggregation methods. This conversion approach, while enabling analysis using the single-table assumption, can result in very large data tables and may involve a loss of information [153].

3.2.2 Search Space

The search space initially involves isolating the target concept T from the remaining attributes, which will be utilized to define the subgroup pattern syntax. For convenience, let's assume we are dealing with a single target attribute $t \in A$ that is suitable for the target application (further details about the target concept will be discussed in the next section). For instance, if our objective is to characterize slow queries, the target attribute T would represent the execution time (q_{time}). Consequently, the crucial question arises: which attributes should we employ to describe interesting subgroups? These attributes are referred to as descriptive attributes denoted by $A_D \subseteq A \backslash T$, with $|A_D| = m_D$ and which are used to represent by *intent* a subgroup which is defined by *extent* as a subset of objects $S \subseteq O$.

A pattern language $\mathcal{L}$ is then defined based on descriptive attributes A_D, and it consists of an extensive set of subgroup descriptions (also known as patterns or intents). Each subgroup description is constructed using selection expressions known as selectors (also referred to as conditions or basic patterns). A basic pattern $sel : O \longrightarrow \{\text{True}, \text{False}\}$ represents a constrained selector that identifies a subset of objects based on their descriptive attribute values and determines whether an instance belongs to the pattern. For example, the selector (`Server.name` = BLV) evaluates to true for the instances $\{o_2, o_3, o_8, o_9, o_{10}\}$, where the attribute `Server.name` has the value BLV.

These selectors are combined using boolean formulas in order to form subgroup descriptions, employing a propositional description language. The most prevalent setting in Subgroup Discovery typically focuses on conjunctive combinations of selectors. This choice is motivated by the fact that such subgroup descriptions are more easily comprehensible and interpretable by human domain experts [168]. Specifically, in the case of strictly conjunctive subgroup descriptions, we define the covering relation as follows.

> **Definition 2** (Description and covering relation)**.** A pattern $P = sel_1 \wedge sel_2 \wedge .... \wedge sel_{m_D}$ is a subgroup description that covers all instances, for which all selectors sel_j evaluate to True. Equivalently it is also written as a set of selectors: $P = \{sel_1, sel_2, ..., sel_{m_D}\}$. In particular, the pattern $P_\varnothing = \varnothing$, which is given by the empty conjunction, describes all instances in the dataset.

We write $P(o)$ for the boolean value that indicates if an instance $o \in O$ is covered by the pattern P. The set of instances, for which this formula evaluates to true, is called the *subgroup cover* or the *extent*.

> **Definition 3** (Extent)**.** The extent of a pattern P is defined as: $sg(P) = \{o \in O \mid$

$P(o)$ = True} and denotes the set of instances which are covered by P. $|sg(P)|$ is the count of these instances. The complement $\neg P$ of a subgroup pattern P covers all instances that are not covered by P

Patterns are partially ordered in $\mathcal{L}$ by a *specialization* (or a *generalization*) relationship:

Definition 4 (Specialization $\sqsubseteq$)**.** A subgroup pattern P_{spec} is called a specialization of another subgroup pattern P_{gen} and we note $P_{gen} \sqsubseteq P_{spec}$ iff $P_{spec} \Longrightarrow P_{sgen}$. P_{gen} is then a generalization of P_{spec}. Trivially, a generalization covers all instances that are covered by its specializations:

$$P_{gen} \sqsubseteq P_{spec} \Rightarrow sg(P_{gen}) \supseteq sg(P_{spec})$$

This can be illustrated by an example from Table 3.1 considering the two patterns: $P = (\texttt{Verrou.date} \geqslant 1)$ and $P' = (\texttt{Verrou.date} \geqslant 1 \wedge \texttt{manyActiveSessions} = \text{Critical})$. We have $P \sqsubseteq P'$ since $P' \Longrightarrow P$ and we have $sg(P) = \{o_2, o_4, o_6, o_7\} \supseteq sg(P') = \{o_2, o_6\}$

It is worth noting that there exists a wide range of description languages in the literature. These include itemsets [274], hyper-rectangles [143], sequences [206, 112], graphs and trees [144, 86, 16], etc. These languages define the space or set of possible descriptions, which in turn determine the set of possible subsets of records that can be considered in an analysis task. In this thesis, as mentioned earlier, we specifically focus on attribute-value data [233] for all our contributions and use cases.

Categorical Attributes. When it comes to defining selectors for descriptive attributes in order to create more complex patterns in the search space, we need to distinguish between categorical and numerical attributes. The simplest and most commonly used selectors for nominal attributes check for value-identity, which can be represented as $sel_{a_j=v}(o) = \text{True} \iff a_j(o) = v$. An example of this type of selector is the previously mentioned $sel_{\texttt{Server.name} = \text{BLV}}$. Another option for nominal attributes is to use negations of values as selectors instead of using the values directly. In the case of ordinal attributes, we can choose a more sophisticated selector $sel_{a_j \geqslant v}(o) = \text{True} \iff a_j(o) \geqslant v$. This can be useful, for example, when we are interested in data queries that are associated with at least a certain number of alarm alerts, i.e., $sel_{\texttt{manyActiveSessions} \geqslant \text{Alarm}}$.

Numerical Attributes. Determining suitable selectors for numerical attributes poses a greater challenge compared to categorical attributes. While value-identity selectors can be used for numerical attributes, they yield meaningful results only when the attribute has a limited range and does not rely on a continuous scale. For example, a selector such as $(\texttt{Verrou.date} = 1)$ works perfectly fine when the numbers belong to $\mathbb{N}$, and it would be unexpected to encounter a query with a high frequency of this attribute, say 100 occurrences. However, when it comes to attributes like `concurrency`, which represents the level of concurrency or contention experienced by a session during a specific sample interval, interpreting selectors based on a single value becomes difficult. Instead, selectors for numerical attributes are typically specified using intervals that cover the domain of the attribute: $sel_{a_j \in [lb,ub]}(o) = \text{True} \iff lb \leqslant a_j(o) \wedge a_j(o) \leqslant ub$. The values lb and ub appearing in the selectors of a numerical attribute are referred to as cut points. Open, half-open, or closed intervals can be used to define the selectors. In Subgroup Discovery, there are two approaches to identify the optimal set of intervals [105]. The first approach is online discretization, where

the mining algorithms determine the best intervals during the automatic search for the top subgroups. The second approach involves determining the intervals used as selectors through an offline discretization step prior to the search. Then, any Subgroup Discovery algorithm can be applied. Various discretization methods exist for numerical attributes. These methods include Equal-width discretization, Equal-frequency discretization, and others. These techniques are unsupervised, meaning that the discretization is performed independently of the chosen target concept. Additionally, there are supervised discretization methods such as Entropy-based discretization [99] and Chi-Merge discretization [146], which take into account the binary target concept when performing the discretization.

3.2.3 Target Concept

The Subgroup Discovery task varies depending on the type of the target concept T. Whether it is a binary target, numerical target, or a complex target like in Exceptional Model Mining.

3.2.3.1 Binary Target Concepts

In the case of a binary target, the property of interest is represented by a pattern P_T. For any subgroup description P, the instances that are covered by P_T are referred to as positive instances (denoted as $tp(P)$). The remaining instances in P are considered negative instances (denoted as $fp(P)$). We use the notation $p_P = |tp(P)|$ and $n_P = |fp(P)|$ to represent the number of positive and negative instances covered by the subgroup description P, respectively. The distribution of the target concept for a subgroup description can be fully characterized by the target share $\tau_P = \frac{p_P}{p_P+n_P} = \frac{p_P}{|sg(P)|}$, which represents the proportion of positive instances within the subgroup. The main objective is to identify subgroup descriptions where the target share is either unexpectedly high or unexpectedly low. For instance, if we rely on the `slow` attribute as our target concept, and considering the previously mentioned pattern $P = (\texttt{Cumulof.ik} = 1 \wedge \texttt{Soft.version} = \text{v2})$, we have $tp(P) = \{o_4, o_7, o_{10}\}$ and $fp(P) = \{o_8\}$ and the target share $\tau_P = \frac{3}{4} = 0.75$.

3.2.3.2 Numerical Target Concepts

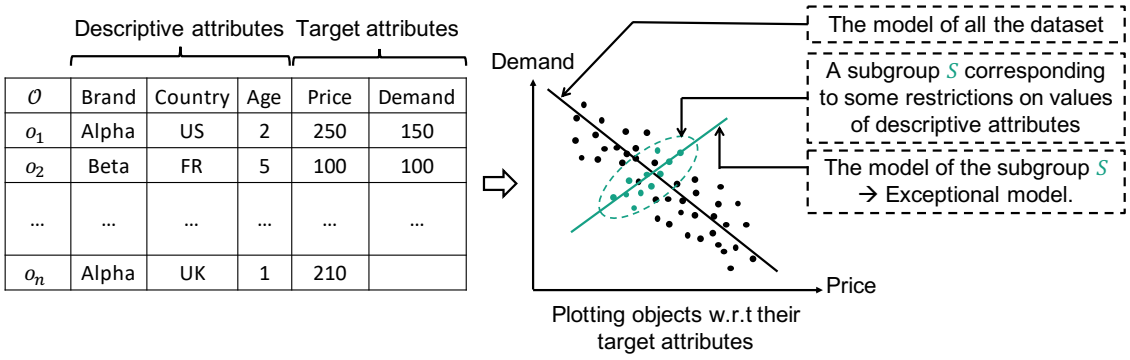
In many scenarios, numerical target attributes can be discretized into binary targets, similar to what we have done when transforming the numerical attribute `time` into the binary attribute `slow`. However, this approach can lead to misleading results and a loss of valuable information. For example, the last pattern indicates that 75% of the subgroup queries are classified as slow, with one object $o_8 \in fp(P)$, yet the execution time $time(o_8) = 9.95$ is only slightly below the threshold of 10 seconds. This highlights the need to consider the complete distribution of the numerical target attribute. To address this, the target distribution for a subgroup description P should be compared with respect to one or more distributional properties of the numerical target attribute. These properties may include the mean value μ_P, the median med_P, or the variance σ_P^2.

3.2.3.3 Complex Target Concepts

Another variant that has emerged in Subgroup Discovery is Exceptional Model Mining (EMM) [167] which is considered as a generalization of SD. In contrast to traditional Subgroup

Discovery, this approach does not rely on a single attribute to define the property of interest. Instead, it utilizes a set of model attributes. In Exceptional Model Mining, a distinct model is constructed for each subgroup. This model consists of a fixed model class specific to the mining task, as well as model parameters that depend on the values of the model attributes within the subgroup. The objective of Exceptional Model Mining is to identify subgroup descriptions where the model parameters significantly deviate from those of the model built from the entire dataset. Several EMM models have been proposed, involving target variables such as regression [94], correlation [95], classification with a target [92], preference among targets [81], Bayesian networks [93], and more. Despite the increased complexity of EMM models, the algorithmic strategies used to identify exceptional subgroups remain similar to those of SD. In fact, the search space for subgroups in EMM remains similar to that of SD. We explore the application of EMM in the next Chapter 5.

Figure 3.2 showcases an example from [32], illustrating the investigation of the relationship between price and demand in the field of economics. According to the economic law of demand, it is generally expected that an increase in the price of a product will lead to a decrease in its demand, resulting in a negative slope when regressing demand on price in a typical regression model. However, there exist specific scenarios where the relationship between price and demand deviates from this conventional expectation. In some cases, when certain conditions are met, people tend to buy more of a product as the price increases. This exceptional behavior challenges the negative slope assumption and can result in a positive slope in the regression line. To explore these exceptional cases, EMM is applied to a dataset comprising product objects. Each product is characterized by descriptive attributes such as category, age, and brand, along with two target variables: price and demand. In [94], a regression model is used to analyze the relationship between price and demand. By leveraging EMM, it becomes possible to identify attribute restrictions that define subgroups with regression models of the target variables significantly differing from the overall regression model observed across the entire dataset.



$\mathcal{O}$	Brand	Country	Age	Price	Demand
o_1	Alpha	US	2	250	150
o_2	Beta	FR	5	100	100
...	...	...	...	...	...
o_n	Alpha	UK	1	210	

Figure 3.2: Example of EMM with a regression model on two target attributes [32].

3.2.4 Interestingness Measures

The task of subgroup discovery is to extract the most interesting subgroups from a large set of potential candidates within the search space. In pursuit of this goal, several interestingness measures have been proposed in the literature, tailored to different types of target concepts.

In this chapter, we will focus on showcasing a selection of interestingness measures specifically designed for binary and numerical target concepts, as applied in our use case of SQL workload analysis. In the subsequent chapters, we will introduce additional interestingness measures specifically designed for more complex target attributes adapted to our use cases. We define an interestingness measure as follows:

Definition 5 (Interestingness measure)**.** A measure $\phi : \mathcal{L} \to \mathbb{R}; P \mapsto \phi(P)$ is a mapping that evaluates the quality of a subgroup pattern P w.r.t. the property of interest T. The greater is $\phi(P)$, the more interesting is P.

Indeed, It is important to mention that there has been extensive research on defining similarity measures and exploring their criteria. In the context of knowledge discovery in databases, the goal is to identify patterns that are valid, novel, potentially useful, and understandable [100]. One framework for assessing interestingness is proposed by [107], who break it down into nine criteria. These criteria, to some extent, overlap with each other and include conciseness, coverage, diversity between patterns, surprisingness, and actionability. The authors categorize these criteria into three groups: objective measures, subjective measures, and semantics-based measures. Objective criteria can be computed solely using raw data, often utilizing statistical or information-theoretic measures. Subjective measures like novelty and surprisingness involve the consideration of the user's prior knowledge [271, 80] (refer to 3.2.7). Semantics-based measures, such as utility and actionability, take into account the meanings of subgroup descriptions in the specific application domain.

In the following, we present measures that can serve as a function ϕ to assess the quality of subgroup patterns when the target attribute is either binary or numerical. It is widely acknowledged that discriminant subgroups of interest are those that maximize the deviation of the target attribute T compared to the overall dataset, while also possessing a sufficiently large size. Preferably, discriminant subgroups should be substantial in size, indicating their significance and reducing the likelihood of their existence in the dataset being attributed to chance. Many of the existing measures belong to the renowned family of `Klösgen` functions [152], which are defined based on the parameter $a \in [0, 1]$.

$$\texttt{Klösgen}_a(P) = sup(P)^a \cdot (\mu(sg(P)) - \mu(O)),$$

where the support $sup(P) = \frac{|sg(P)|}{|O|}$ measures the proportion of objects from the set O belonging to the subgroup P, and the target mean $\mu(sg(P)) = \sum_{o \in sg(P)} \frac{T(o)}{|sg(P)|}$ represents the average value of the target attribute T within subgroup P. Hence, a higher value of $sup(P)$ corresponds to a higher value of $\texttt{Klösgen}_a(P)$. However, maximizing $\texttt{Klösgen}_a(P)$ also involves maximizing the deviation of $\mu(sg(P))$ from the overall mean $\mu(O)$. The choice of parameter a determines the relative importance of $sup(P)$ in the final measure of interestingness, leading to measures with different statistical interpretations. These measures are presented in what follows.

Average function u**.** Also called unusualness [263], it refers to the `Klösgen` function with $a = 0$, denoted as $u(P) = \mu(sg(P)) - \mu(O)$. This measure is utilized when there is a desire to evaluate subgroups without considering the impact of $sup(P)$ on the score. Typically, in conjunction with a minimum size constraint for returned subgroups, it helps avoid retrieving

very small subgroups. $u(P)$ provides a subgroup ordering that is identical to another popular measure: $\texttt{Lift}(P) = \frac{\mu(sg(P))}{\mu(O)}$.

`WRAcc` measure [163]**.** It is one of the most popular measures in SD. It corresponds to the `Klösgen` function with $a = 1$:

$$\texttt{WRAcc}(P) = sup(P) \cdot (\mu(sg(P)) - \mu(O)) .$$

For the specific case when T is binary, it can be written as:

$$\texttt{WRAcc}(P) = \texttt{Pr}(o \in sg(P) \wedge T(o) = 1) - \texttt{Pr}(o \in sg(P)) \cdot \texttt{Pr}(T(o) = 1),$$

where `Pr` is the probability of an event to happen. Theoretically, the more P is statistically dependent on true target values ($T(o) = 1$), the higher is $|WRAcc(P)|$.

`Mean-test`. One drawback of `WRAcc` is that it tends to assign high scores to subgroups with large support, even if they lack uniqueness or unusualness. Consequently, many methods have favored the use of the `Mean-test` measure, which corresponds to the `Klösgen` function with $a = 0.5$:

$$\texttt{Mean-test}(P) = \sqrt{sup(P)} \cdot (\mu(sg(P)) - \mu(O)) .$$

From a statistical standpoint, it has been demonstrated that this measure yields an equivalent ordering compared to the Binomial test [168].

`T-score`. One limitation of the `Klösgen` functions is that they do not explicitly optimize the dispersion of the target attribute within subgroups. This can result in inconsistent conclusions, especially when the dataset contains numerous outliers. In some cases, the mean $\mu(sg(P))$ might not accurately represent the target values in the subgroup P, especially if P includes a few outliers with extreme values of T. To address this concern, one measure that incorporates subgroup cohesion is the `T-score` [239], defined as:

$$\texttt{T-score}(P) = \frac{\sqrt{sup(P)}}{\sigma(P)} \cdot (\mu(sg(P)) - \mu(O)) ,$$

where $\sigma(P)$ represents the standard deviation of target values T within subgroup P. A smaller value of $\sigma(P)$ indicates higher cohesion among the target values in P, leading to a higher `T-score`. This measure reflects the significance of the deviation of target values within a subgroup using a Student's t-test. However, it is important to note that a direct statistical interpretation of the `T-score` should be avoided when the target attribute is not normally distributed and the subgroup size is small, such as $|sg(P)| < 30$.

Median-based measures `q_med`. An alternative approach to mitigate the influence of outliers on subgroup scores is to employ the median $\texttt{med}(sg(P))$ of T in the `Klösgen` function, instead of the average $\mu(sg(P))$. The use of the median estimator is advantageous as it offers greater robustness to noise [170]:

$$\texttt{q\_med}(P) = sup(P)^a \cdot (\texttt{med}(sg(P)) - \texttt{med}(O)) .$$

3.2.5 Exploring the Search Space

Once the pattern language is established and the appropriate subgroup interestingness measure is selected based on the specific use case, the next step involves exploring the search space using efficient and preferably scalable algorithms to identify the top-k subgroups. In the following, we present a typical subgroup discovery task, which aims to find the top-k subgroups based on the defined quality measure. This task is formally analogous to the generic problem outlined in [95].

Problem 1 (Top$-k$ Subgroup Discovery Problem)**.** Given a user specified parameter k, find the top-k subgroups with the highest values of the interestingness measure ϕ. Formally, find the subgroup set:

$$\mathcal{R} = \{P \in \mathcal{L} \mid rank(P) \leqslant k\},$$

where $rank(P)$ gives the rank of P w.r.t. its score ϕ, that is: $rank(P) = |\{P' \in \mathcal{L} \mid \phi(P') > \phi(P)\}| + 1$.

The computational complexity of the subgroup discovery (SD) problem is known to be prohibitive due to the exponentially increasing size of the search space $|\mathcal{L}|$ with respect to $|A_D|$. To address this challenge, various algorithms have been proposed to efficiently explore the search space. These algorithms can be categorized into heuristic, exact, and anytime approaches, each with its own trade-offs in terms of optimality and scalability.

Heuristic approaches are employed when the search space is too large for exhaustive exploration. While these strategies may not guarantee the discovery of optimal patterns, they offer tractability and faster execution times. The objective is to develop strategies that enable the discovery of high-quality patterns while considering diversity. Several strategies have been introduced in the literature [163, 197], with the most common being the use of breadth-first search, as exemplified by the well-known beam search algorithm. Notable works [287, 243] demonstrate algorithms that utilize beam search to discover high-quality, non-redundant subgroups. Sampling-based methods [42, 43] have also been sparsely used in subgroup discovery as heuristic approaches. These methods offer the advantage of discovering high-quality patterns within a short amount of time. Typically, a statistical distribution is designed based on the optimization of quality criteria, allowing patterns that optimize those criteria to have a significantly higher probability of being generated. Exhaustive algorithms, on the other hand, prioritize the guarantee of the discovery task at the expense of execution time and feasibility. Various techniques have been proposed to render the search more tractable. These techniques include compressing the search space using equivalence classes and closure systems [111], pruning the number of candidates using anti-monotone constraints [142] and employing optimistic estimates for the quality of subgroup specializations [111, 170, 29]. Furthermore, another category of algorithms that combines the properties of the three first categories is known as anytime algorithms, which allow for the retrieval of the best set of patterns at any given moment during the search [44, 30].

In the following sections, we delve into two methods that we have extensively explored in our use cases, both within this chapter and in Chapter **5**. These methods include: (1) an exact algorithm based on a depth-first search, and (2) a heuristic algorithm utilizing beam search. It is important to note that our focus does not center on introducing significant contributions to

the exploration of the search space in the field of Subgroup Discovery. Instead, our objective is to showcase the application and evaluation of these methods within the context of our specific use cases.

Depth-first algorithm. This approach exhaustively explores the search space $\mathcal{S}$ in a depth-first manner. After defining an order relation between patterns, the search space forms a lattice structure with the empty pattern as a supremum and the pattern containing all the selectors as infimum. Then, this lattice is explored in depth. We start from the empty pattern $P = (a_i \in dom(a_i) \mid a_i \in A_D)$, i.e., no restriction for any attribute. Then, a refinement operator is recursively applied on selectors of P, continuously making it more restrictive. Refinements can be operated by adding a symbolic attribute value or adding a numerical attribute cut point. As the search space can be extremely large, a naive enumeration of subgroups fails. For this reason, the exact algorithm uses many techniques to optimize the exploration. Some anti-monotonic constraints are generally used, such as minimum support δ, i.e., if a pattern covers less than δ objects then this pattern is not refined anymore, as its refinement necessarily covers less than δ objects. Furthermore, tight optimistic estimates `TOE` [170] are used. These functions allow to efficiently upper bound all the subgroup interestingness values in a whole branch of the search space. If the `TOE` of a branch is lower than the score of the top$-k$ already found subgroup, then the branch is pruned, as it does not contain any subgroup with a score higher than the already found top$-k$. Refer to [170] for other optimization strategies details.

Beam-search algorithm. The most popular heuristic approach is Beam-search [68]. This approach performs a heuristic level-wise search over the pattern lattice. It requires specifying the width parameter $w \in \mathbb{N}$, which is the maximum number of patterns kept in each level of the lattice. It starts from the empty pattern $P = (a_i \in dom(a_i) | a_i \in A_D)$. Then, it recursively goes to the next level by refining patterns of the current level and selecting the top$-w$ refined patterns that maximize the interestingness. These top$-w$ patterns are then refined again to continue to a deeper level. At the end, the algorithm selects the top$-k$ subgroups among all the top$-w$ ones selected from each level.

3.2.6 Avoiding Redundancy

The process of selecting interesting subgroups based solely on discriminative measures can lead to significant overlap among the identified patterns, where distinct patterns cover nearly the same set of objects. This limitation becomes more pronounced in a top-k approach, as the limited result set $\mathcal{R}$ may exclude other potentially interesting subgroups. Consider the example shown in Table 3.1, where two different patterns, $P = (\texttt{Verrou} \geqslant 1)$ and $P' = (\texttt{Verrou.ik} \geqslant 1)$, exhibit high correlation since they cover similar sets of objects. To address the need for presenting diverse yet interesting patterns, various solutions have been proposed in the literature. In our approach, we propose utilizing the Jaccard similarity [231], which measures the similarity between two subgroup patterns as the fraction of their extent intersection over their extent union. For the example patterns P and P', the Jaccard similarity is computed as follows:

$$sim(P, P') = J(P, P') = \frac{|sg(P) \cap sg(P')|}{|sg(P) \cup sg(P')|} = \frac{6}{7}$$

Greedy Selection. The greedy approach follows an iterative process to construct the non-redundant subgroup set $\mathcal{R}'$. In each iteration, the best subgroup pattern $P^\star$ is identified from the initial subgroup set $\mathcal{R}$ and added to $\mathcal{R}'$. Subsequently, all subgroups in $\mathcal{R}$ that exhibit a similarity exceeding a specified threshold with $P^\star$ are removed. This process is repeated until $\mathcal{R}$ becomes empty. However, it is important to note that this technique requires the user to determine an appropriate threshold for the Jaccard similarity. Choosing a threshold that is too large may still result in overlapping subgroups while selecting a small threshold can lead to the suppression of potentially interesting subgroups. As a result, the performance of this method is highly sensitive to threshold choice, which often requires empirical selection.

Hierarchical Clustering. To provide a comprehensive and easily interpretable overview of the resulting subgroups, we employ agglomerative hierarchical clustering [17] on the result set $\mathcal{R}$. This clustering technique starts with each subgroup forming a separate cluster at the bottom of the hierarchy. As we move up the hierarchy, pairs of clusters are merged based on their similarity, using the Jaccard dissimilarity defined as $1-$Jaccard similarity. The resulting output is a binary clustering tree, also known as a dendrogram. Hierarchical clustering offers a hierarchy of partitions, allowing the flexibility to select a specific partition by truncating the tree at a desired level. This means that instead of relying on a dissimilarity threshold like the greedy approach, users can specify the number of non-redundant subgroups patterns without explicitly setting a dissimilarity threshold.

In addition to the aforementioned approaches, there are alternative techniques that specifically address the issue of redundancy among the discovered patterns. One such approach is Subgroup Set Discovery [29, 286]. Rather than focusing solely on individual subgroups, this methodology aims to identify sets of high-quality non-redundant subgroups. The scoring of pattern sets in this approach combines components that evaluate the aggregated individual interestingness of the contained subgroups, as well as components that measure the diversity of the subgroup patterns.

3.2.7 Background Knowledge and Subjective Interestingness

The incorporation of background knowledge, also referred to as prior knowledge, is widely acknowledged as a crucial aspect of successful data mining tasks in real-world scenarios. This is because the extracted patterns should not only be statistically significant but also relevant to the user's specific preferences. For instance, a pattern may be deemed interesting by one user, while another user, may not find it compelling. This can be attributed to several factors. Firstly, the first user may have already anticipated the existence of that pattern based on her existing knowledge or because she possesses a deeper understanding of the underlying data distribution. Alternatively, the pattern may be associated with a topic that does not capture the interest of the second user. In subgroup discovery, considering background knowledge is particularly valuable in different stages, starting with the data preparation phase. During this phase, background knowledge assists in generating an appropriate search space (e.g., by considering the normality/abnormality information of attribute values, we can derive discretization bounds to refine the search [18]). Additionally, algorithms exploit background knowledge to optimize the search by avoiding the exploration of uninteresting attribute combinations. However, our main focus lies in the integration of background knowledge into the interestingness measures. These measures can be adapted during the search algorithm to account for the importance of attributes and their known correlations.

Several studies have proposed interestingness models that go beyond considering only the data and take the user's perspective into account. This subjective nature of interestingness introduces a challenge in pattern mining, which was recognized in prior research [271] and has gained attention in the past decade. Notably, a significant work by [80] presents a comprehensive and statistically-founded framework that incorporates user background knowledge as constraints on a probability distribution, representing the user's uncertainty about the data. The maximum entropy (MaxEnt) distribution is employed to model the prior information, capturing patterns that are surprising given the user's beliefs. Furthermore, this framework also prioritizes patterns that are easily understandable to the user. The subjective interestingness $SI(P)$ of a pattern P is defined as the ratio between the information content $IC(P)$ and the description length $DL(P)$ of the pattern, reflecting the information density within the pattern. Additionally, when a pattern is presented to the user, the background model is updated to incorporate this new information as already known by the user. This allows for the discovery of fresh and non-redundant patterns. A visual representation of this framework can be found in Figure 3.3 from the work by [32]. In our research presented in Chapter 5, we leverage the subjective interestingness framework to drive our work forward.

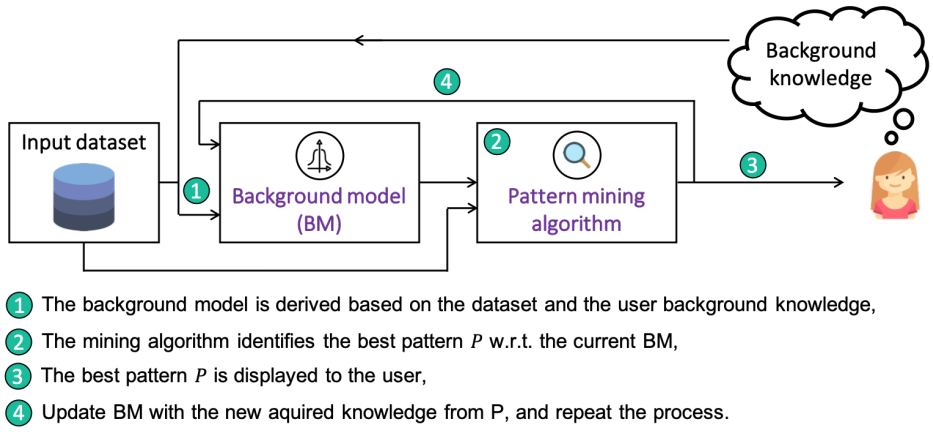


Figure 3.3: Overview of the subjective interestingness framework proposed in [80].

3.3 SQL Workload Analysis Problem

The SQL workload analysis problem has garnered significant attention in the database and software engineering communities. The aim is to enhance the reliability and efficiency of interacting with data in large-scale systems. Query workload analysis has proven effective in addressing various related problems, employing data-driven strategies. These methods automatically analyze logs and executed queries to perform tasks like index recommendation [58, 57], anti-pattern detection [13, 97, 61], and modeling user and application behavior [283, 323]. The usability of these methods depends heavily on how the data is represented. Thus, several approaches have been proposed to transform the data into simplified forms before carrying out the main task, such as workload compression [57], efficient parsing [157], or embedding [133] of SQL queries. Then, a myriad of Machine Learning methods have been evaluated for different workload analysis tasks. However, many of these approaches

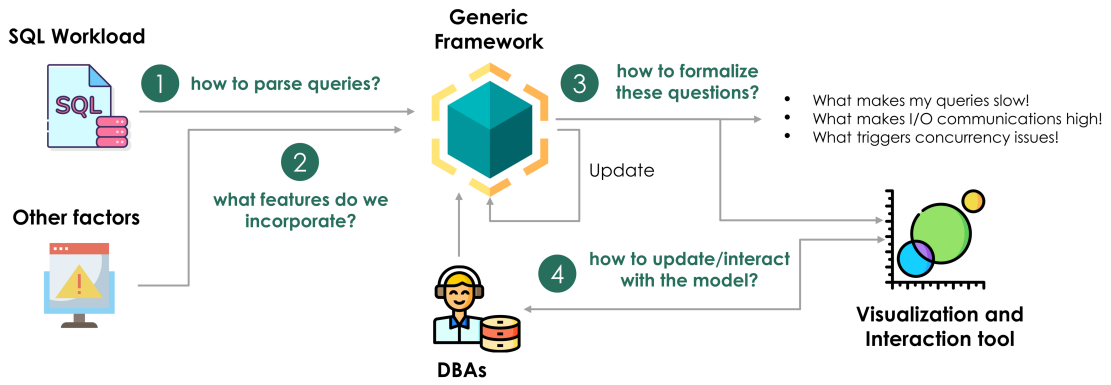


Figure 3.4: Addressing key questions in Subgroup Discovery for SQL workload analysis.

rely on clustering-based methods, which are impractical for identifying subsets of data that specifically differentiate a desired property. In contrast, various major commercial database systems have developed tools to automate this task, such as query planners and optimizers. For instance, Microsoft SQL Server has included the index selection feature in its Tuning Advisor since SQL Server 2000 [4]. Although these tools are widely used by DBAs, they remain specific and non-generic, limited to certain features. Utilizing a query optimizer, for example, necessitates examining individual cases to identify issues in each query separately.

Hence, we believe that the Subgroup Discovery approach offers an efficient solution to this problem, as it can uncover various types of tasks and provide answers to the broader question: How to characterize SQL queries that exhibit certain properties of interest? For instance, determining the factors that contribute to slow queries can be extremely beneficial for performance optimization problems. Similarly, several other questions of this nature may arise, such as characterizing queries that excessively consume I/O communication or identifying contexts where SQL queries significantly escalate concurrency issues.

Building a Subgroup Discovery framework to tackle this problem, however, requires addressing several key questions, as illustrated in Figure 3.4. (1) Firstly, the data needs to be introduced to the algorithm in the correct format. (2) Secondly, it is essential to identify the most suitable and actionable data that can be augmented with SQL queries for contextualization. (3) Thirdly, choosing the appropriate settings regarding the building blocks of the subgroup discovery approach is important. (4) Lastly, an interaction mechanism must be introduced to enable iterative learning and collaboration with the user.

3.3.1 Data Preprocessing

We performed our analysis on a workload W consisting of 150K unparametrized SQL statements. These statements were collected from over 400 databases supervised within our company, all of which share a nearly identical database schema. To ensure efficiency, we parsed the queries to extract tables and attributes specific to each type of SQL clause. Subsequently, we enriched the queries by incorporating various database metrics and supervision alerts. This augmentation process yielded thousands of properties that greatly assist in contextualizing the Subgroup Discovery approach.

Table 3.2: Example of parsing an SQL query.

<table>
<tr><th colspan="2">Raw SQL query</th></tr>
<tr><td colspan="2">SELECT m.ik
FROM model AS m
JOIN prod AS p
WHERE m.ik = p.ik
AND m.uex = p1
AND (m.uex in collection0
OR m.ik in collection1)
AND (m.dossierinfo = p3
GROUP BY m.ik
HAVING (COUNT(DISTINCT p.ik) = p2)
AND (SUM(m.nbembal) = MAX (p.nbembal))</td></tr>
<tr><th>Our parsing result</th><th>Parsing result of [205]</th></tr>
<tr><td>SELECT_model.ik ⟶ 1</td><td>SELECT_ik ⟶ 1</td></tr>
<tr><td>FROM_model ⟶ 1
JOIN_prod ⟶ 1</td><td>FROM_model ⟶ 1
FROM_prod ⟶ 1</td></tr>
<tr><td>WHERE_model.ik ⟶ 3
WHERE_model.uex ⟶ 1
WHERE_model.dossierinfo ⟶ 1
WHERE_prod.ik ⟶ 1</td><td>WHERE_ik ⟶ 4
WHERE_uex ⟶ 1
WHERE_dossierinfo ⟶ 1</td></tr>
<tr><td>GROUPBY_model.ik ⟶ 1</td><td>GROUPBY_ik ⟶ 1</td></tr>
<tr><td>HAVING_prod.ik ⟶ 1
HAVING_model.nbembal ⟶ 1
HAVING_prod.nbembal ⟶ 1</td><td>HAVING_ik ⟶ 1
HAVING_nbembal ⟶ 2</td></tr>
<tr><td>COUNT_prod.ik ⟶ 1
SUM_model.nbembal ⟶ 1
MAX_prod.nbembal ⟶ 1</td><td></td></tr>
</table>

3.3.1.1 Query Transformation

A common preprocessing step involves decomposing, parsing, and tokenizing SQL queries. This process results in the formation of a numerical vector where dimensions represent the usage count of data tables and attributes [12, 7, 5]. To accomplish this, we utilized the readily available Mozilla parser [221], which provides an SQL syntactic tree in XML format. We further parsed this tree to normalize case sensitivity and eliminate irrelevant terms such as constants and logical operators. Each token is then associated with the clause it belongs to by appending a prefix indicating the relevant clause. For example, in Table 3.2, the queried table `model` appears in the `FROM` clause of the query, resulting in the token `FROM_model`. For every token, we record the frequency of its appearance within each SQL clause.

We made two extensions to the Mozilla parser[2]. Firstly, we expanded its functionality to handle not just SQL queries, but also Hibernate queries used in our ERP as the ORM layer. This involved adding support for the reserved keywords used in Hibernate queries, such as `JOIN FETCH`, to the original parser. Secondly, and perhaps more significantly, our parser stands out from several existing parsers [12, 7, 5] due to its ability to handle nested queries while preserving the query's structure. Unlike many parsers [205, 5], we do not

[2] `https://github.com/klahnakoski/mo-sql-parsing/pull/26`

Table 3.3: Additional features used with parsed SQL queries.

Query properties	Topology		Alerts	Oracle ASH	
query	serverName	dbMemory	manyActiveSessions	application	concurrence
day	declination	sgaMax	blockedSessions	configuration	network
hour	softwareVersion	dbProcesses	poolAlmostFull	administrative	cpu
time	codeVersion	jdbcMin	anomalyASH	systemI/O	userI/O
nrows	dbVersion	jdbcMax		queuing	scheduler
		dbCursorsMax		commit	

remove substitutes for temporary table names, known as `aliases`. Instead, we utilize aliases to determine the table to which a column in a `SELECT` clause or a predicate in `WHERE` or `GROUP BY` clause belongs. This approach proves beneficial when dealing with nested queries, join clauses, or queries involving multiple tables. As a result, unlike the method proposed by [205], our parser encodes two columns with the same name from different tables differently, as illustrated in the example in Table 3.2. Additionally, following the work of [82], we treat function calls present in the SQL statement as independent clauses. Lastly, it is important to mention that we do not group semantically equivalent queries into a canonical form, as done in [157]. This is because the manner in which a query is written can affect its execution plan and, consequently, its execution time.

3.3.1.2 Other Data Sources

As highlighted in Table 3.3, we augmented our parsed SQL queries with the following additional information into the data model:

Topology. Each database is queried by our in-house developed Enterprise Resource Planning software (ERP), serving as the application. In our analysis, we take into account the application identifier (`serverName`), as well as its major and minor versions (`softwareVersion`, `codeVersion`). The relevant properties of the database include its vendor/version (`dbVersion`), one of six main schema families (`declination`), the size of the database server memory (`dbMemory`), the maximum database memory usage (`sgaMax`), the maximum number of processes (`dbProcesses`), the minimum and maximum size of the connection pool (`jdbcMin`, `jdbcMax`), and the limit on the number of cursors per database session (`dbCursorMax`).

Active Session History (ASH). Active Session History (ASH)[234] was introduced in Oracle 10g and later implemented in other database systems like PostgreSQL. ASH provides information about active sessions that are waiting for system resources such as CPU, System I/O, or Network. It offers a temporal distribution of sessions waiting for each resource category, captured at regular intervals. This data is valuable for diagnostics and tuning purposes. For instance, it can help identify queries that unexpectedly consume excessive network resources by generating a high number of network waiting sessions.

Alerts. Our monitoring system generates rule-based alerts when anomalies are detected in the database environment. For our analysis, we consider: (1) `manyActiveSessions` for an unusually high number of active sessions, (2) `blockedSessions` when certain sessions remain blocked for a significant duration, (3) `poolAlmostFull` when the connection pool size approaches its maximum limit, and (4) `anomalyASH` for abnormalities in the distribution of ASH data, such as an increased proportion of sessions waiting for network or system I/O. We

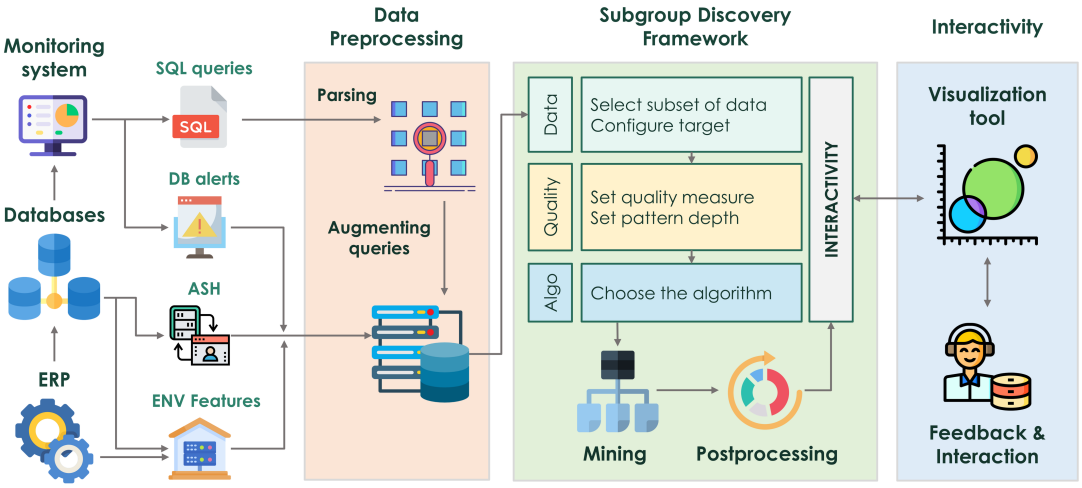


Figure 3.5: Overview of our Subgroup Discovery framework for SQL workload analysis.

enrich the queries with any alerts that coincide with their execution. Each alert is assigned one of four levels: Info, Alarm, Critical, or Blocking.

These features have been carefully selected in collaboration with our DBAs. It is important to note that our methodology is highly flexible, allowing for the consideration of additional numerical and categorical properties as needed. It is worth highlighting that no previous work has explored such a combination of high-dimensional features alongside the expressive representation of SQL queries.

3.3.2 Overview on the Framework

Figure 3.5 provides an overview schema of the implemented framework, which illustrates the entire process from data preprocessing to mining interesting subsets of queries that indicate specific properties of interest. The framework handles a large batch of SQL queries along with their associated topology (i.e., environmental features) and calculates the corresponding alert levels that co-occur with the execution time of queries. Efficient parsing of queries is performed to convert them into a structured format, enriched with relevant information, and then fed into a subgroup discovery model. The framework offers the user the ability to select subsets of data to be used in the pattern syntax. Furthermore, the user can configure the target based on the specific use case they wish to perform, such as characterizing slow queries or retrieving subsets of queries that demonstrate concurrency issues. To facilitate comparison between different results, a wide range of similarity measures is provided. Additionally, the user can choose the algorithm for search space exploration. We used the Python implementation of [169], which provides several subgroup discovery algorithms. However, since this implementation does not support all the relevant measures for our case study, we extended and collaborated to the framework[3] to incorporate the support measure, `T-score`, as well as median-based measures like `q_med`.

[3] `https://github.com/flemmerich/pysubgroup/graphs/contributors`

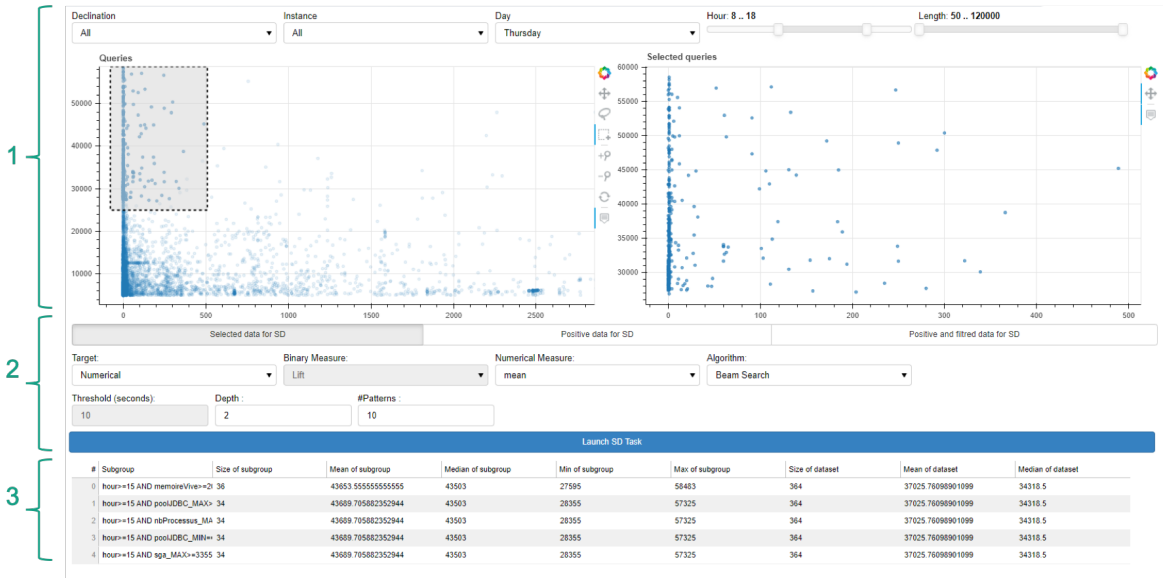


Figure 3.6: Main sections of the interactive SD tool: (1) dataset properties, (2) search strategy, and (3) results.

Figure 3.6 showcases a graphical interface designed to enhance user interaction and facilitate the selection of appropriate parameters for their task. The interactive visualization supports various data types for both input features and the target, including nominal and numerical attributes. It offers a diverse set of interestingness measures and algorithms. The main window consists of three essential panels: (1) the dataset properties, (2) the search strategy, and (3) the results.

Dataset panel: This panel enables users to select subsets of data that are of interest. They can achieve this through data point selection, where queries are plotted based on execution times, row counts, and other query properties. The graph on the right conveniently rescales the selection made on the left graph.

Search strategy panel: Here, users can configure the mining task. First, they need to define the target. There are two options available: choosing a specific attribute as the target or creating a binary target graphically by associating the positive class with the data subset selected in the right graph while considering the remaining data in the left graph as the negative class. Additionally, users can specify the interestingness measure and the mining algorithm. Finally, they can set the desired number of returned subgroups and the maximum depth of patterns.

Results: After executing the mining task, this panel displays the identified subgroups. For each subgroup, it presents the corresponding pattern along with relevant statistics such as subgroup size, median values, or target share.

3.4 Experiments

We conducted experiments to assess the effectiveness of the Subgroup Discovery approach in addressing the SQL workload analysis problem. Firstly, we validated the capability of the framework presented in 3.3.2 to characterize discriminant subgroups that exhibit statistical

Table 3.4: Datasets statistics.

Dataset	Queries	Features	FROM Tables	JOIN Tables	Projections	WHERE atts	HAVING atts	GROUPBY atts	ORDERBY atts	Sparsity
All	148796	8691	497	526	3740	3294	10	199	391	99.55%
D1	37149	4596	275	270	2036	1680	10	96	196	99.22%
D2	48823	246	1	1	86	85	2	21	11	84.27%
D3	3031	570	58	30	158	275	3	6	15	94.77%
D4	26735	3723	218	234	1658	1324	10	91	154	98.97%

significance in relation to various target problems. Subsequently, we performed a quantitative analysis to evaluate the execution time of each algorithm under different parameter settings, including the number of patterns (k) and rules depth.

3.4.1 Experimental Setup

We conducted experiments on an SQL workload consisting of Hibernate queries executed on our production environment servers over a span of one week. Due to the large number of queries, our monitoring system records only those with execution times exceeding 5 seconds. To enrich the dataset, we incorporated additional relevant information described in Table 3.3. The dataset comprises 148,796 queries, each described by 8,691 features. Table 3.4 presents an overview of the dataset characteristics. It's worth noting that the dataset is highly sparse, with only 0.45% of non-zero values. The experiments described hereafter were initially conducted on a single machine with an Intel(R) Core(TM) i5-10210U CPU @1.60GHz and 32GB RAM.

3.4.2 Qualitative Analysis

We address the following diverse set of research questions:

- ✘ **RQ1**: What factors contribute to the slow execution of queries?
- ✘ **RQ2**: Under what circumstances do queries exhibit concurrency issues?
- ✘ **RQ3**: How can we characterize queries that frequently coincide with alerts related to *blocked sessions*?

3.4.2.1 Methodology

To ensure a comprehensive analysis, we conducted experiments for each use case within a specific context based on the industrial requirements we encountered. In detail, we evaluated each research question using the following subsets of data:

- **RQ1**: We assessed this question on dataset **D1**, which comprises queries executed on all sales servers with a minimum of 100 users. The sales declination accounts for 74.04% of the data.
- **RQ2**: To address this question, we focused on dataset **D3**, which specifically considers the software version V15_2. This particular version exhibits a significantly higher occurrence of concurrency issues compared to other software versions. For our analysis, we utilized a binary target that identifies a potential issue if there are at least an average of 5 concurrent processes within a 10-second interval during query execution.

Table 3.5: Subgroup Discovery results.

ID	Target	Measure	Subgroup patterns	Size	Quality
D1	time	Median	(P_1) : WHERE_stocks.gestion.modele.lot.prod.ref.auditinfo.etat $\geqslant 1$	8	$161 \cdot$ q_med$(P_\varnothing)$
			(P_2) : FROM_ventes.cumuls.modele.cumulmultiple $\geqslant 1$	451	$21 \cdot$ q_med$(P_\varnothing)$
			(P_3) : WHERE_ventes.cumuls.modele.cumulmultiple.valzvcliX $\geqslant 1$	45	$21 \cdot$ q_med$(P_\varnothing)$
			(P_4) : WHERE_.ventes.cumuls.modele.cumulmultiple.valzvartX $\geqslant 1$	45	$21 \cdot$ q_med$(P_\varnothing)$
D2	slow $\tau_{P_\varnothing} \simeq 0.6$	Lift	(P_5) : GROUPBY_stocks.gestion.modele.mvtrealise.refexterne $\geqslant 1$	131	$\tau_P = 1$
			(P_6) : serverName = ServerX $\wedge$ systemI/O > 50	38	$\tau_P = 1$
		WRAcc	(P_7) : WHERE_stocks.gestion.modele.mvtrealise.etatsynchro $\geqslant 1 \wedge$ jdbcMax < 200	20668	$\tau_P \simeq 0.99$
			(P_8) : WHERE_stocks.gestion.modele.mvtrealise.auditinfo.datcre $\geqslant 1 \wedge$ dbVersion $= 2.3$	20675	$\tau_P \simeq 0.99$
			(P_9) : manyActiveSessions = Alarm	44	$\tau_P \simeq 93\%$
D3	**concurrence** $\tau_{P_\varnothing} \simeq 0.06$	Lift	(P_{10}) : FROM_.stocks.fichierbase.modele.produit $\geqslant 1 \wedge$ administrative $= 0.3$	8	$\tau_P = 1$
		Binomial	(P_{11}) : serverName = ServerY $\wedge$ commit $\geqslant 1.7 \wedge$ systemI/O > 10.2	51	$\tau_P \simeq 0.94$
D4	**blockedSess** $\tau_{P_\varnothing} \simeq 0.04$	Lift	(P_{12}) : JOIN_.commandesfactures.modele.histcdeligliv.applibudrist $\geqslant 1$	7	$\tau_P = 1$
			(P_{13}) : WHERE_ventes.commandesfactures.modele.cdeligliv.bonliv.datdepart $\geqslant 1$	9	$\tau_P \simeq 0.90$
		Binomial	(P_{14}) : anomalyASH = Critical	151	$\tau_P \simeq 0.85$
			(P_{15}) : poolAlmostFull = Info	124	$\tau_P \simeq 0.99$

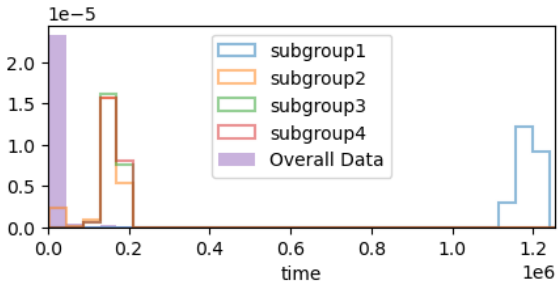
- **RQ3**: Dataset **D4** consists of a specific set of servers where we observed a notable increase in blocked session alerts.

Table 3.4 provides detailed characteristics of these four sub-datasets. For each scenario, we selected the top 10 subgroups based on the most appropriate interestingness measure for the respective problem. The measure selection was made empirically by comparing the identified subgroups using different measures. We opted for the measure that not only produced meaningful results but also offered interpretability through statistical distribution analysis and expert consultation. To ensure diversity and avoid redundancy, we processed the resulting patterns. The obtained patterns are presented in Table 3.5, which includes their respective sizes and deviant qualities compared to the dataset. In the case of binary target problems, we compared the target share τ_P for each subgroup pattern P.

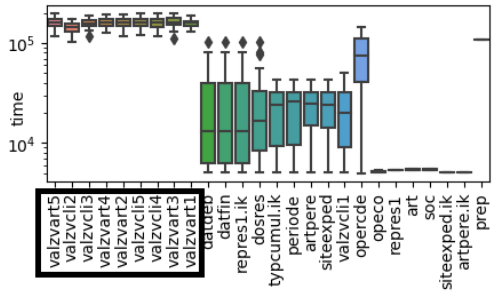
3.4.2.2 Results Interpretation

Actionable and relevant subgroups have been identified in different use cases. In dataset **D1**, we focused on subgroups whose median execution time is significantly higher than the median of the dataset, while taking into account the subgroup size. We chose the measure `q_med` instead of the `Mean-test` because we observed that the mean is more sensitive to outliers in this particular example. One subgroup pattern (P_1), consists of queries involving the attribute `auditinfo.etat` in the `WHERE` clause. It has a very large median compared to the dataset, but only a few objects. The density distribution of this subgroup, as shown in Figure 3.7a, exhibits significant divergence from the usual distribution of the original data. On the other hand, the pattern (P_2) includes all the 451 queries executed on the `cumulmultiple` table, characterized by a large median. Subgroup patterns (P_3) and (P_4) are subsets of (P_2), as they cover only queries with the attributes `valzvcliX` and `valzvartX`, respectively, in their `WHERE` clause. Figure 3.7a demonstrates that the deviation of (P_3) and (P_4) from the overall distribution is stronger than the deviation of (P_2) since they do not cover some slow queries present in (P_2). To better understand this result, we examined the `cumulmultiple` table by highlighting the distributions of its attributes in Figure 3.7b. We then confirm that mostly the attributes `valzvcliX` and `valzvartX` cause the subgroup (P_2) to be identified.

In dataset **D2**, we discretize the attribute `time` to categorize queries with an execution time exceeding 10 seconds as slow queries. Each measure yields interesting results that



(a) Subgroups distribution w.r.t time on D1 compared to overall data.



(b) Distribution of `cumulmultiple` attributes.

Figure 3.7: Statistical distributions of subgroups found on D1.

incorporate extended features, such as alerts in (P_9) and environment variables in (P_7). For instance, we discovered that all queries executed on the `mvtrealise` table, specifically on ServerX[4], when the `systemI/O` is at least 50, have an execution time of more than 10 seconds. Furthermore, whenever the `refexterne` attribute is used in the `GROUP BY` clause, the query experiences significant delays. Unlike the `lift` measure, which solely relies on subgroup precision, `WRAcc` takes into account the subgroup size. Remarkably, subgroups (P_7) and (P_8) stand out as they cover over 42% of the queries while maintaining an approximate target share of 0.9, compared to the overall one. In other words, these subgroups contain more than 70% of the slow queries.

In dataset **D3**, our objective is to identify the context in which queries face concurrency problems. The most effective measures for this analysis are `lift` and `binomial`. Despite the estimated target share is only 0.06 on the overall dataset, we managed to extract subgroups exceeding the threshold of 0.94. A notable subgroup, denoted as (P_{11}), alone represents 28% of the objects that exhibit concurrency issues.

In dataset **D4**, our focus is on extracting relevant hypotheses that demonstrate the context in which the `blockedSessions` alert is raised, specifically with blocking or critical levels. This alert typically occurs when a query locks a critical resource, causing new sessions to be put on hold. We discovered that whenever the table `applibudrist` is joined with another table, the alert is triggered. Another possible reason is a process querying a table without appropriate indexes. To validate this assumption quickly, subgroup (P_{13}) allows us to check the presence of this scenario on the `datdepart` attribute. Moreover, we observed a high correlation between the aforementioned alert and two other alerts described by subgroups (P_{14}) and (P_{15}).

3.4.3 Quantitative Analysis

We conducted an analysis of the time performance of both exhaustive and heuristic SD algorithms on the four datasets. For each scenario, we varied the number of returned subgroups (k) and the depth, representing the maximum number of selectors per pattern. The results are presented in Table 3.6. The `limit` value indicates an execution time exceeding 1,000 seconds. It is noteworthy that Beam-Search consistently completed within 274 seconds for all configurations, whereas Depth-First exceeded the 1,000-second threshold in many instances.

[4]Server names have been anonymized

Table 3.6: Execution time (in seconds) of SD algorithms.

Algo	**Beam-Search** (heuristic)				**Depth-First** (exhaustive)			
# patterns (k)	10		50		10		50	
depth	2	3	2	3	2	3	2	3
D1	20.17	90.39	113.71	274.60	`limit`	`limit`	`limit`	`limit`
D2	5.35	5.40	28.44	45.04	440.01	`limit`	458.2	`limit`
D3	0.75	0.83	3.94	4.27	0.18	0.50	0.53	0.59
D4	10.73	10.98	56.30	62.35	`limit`	`limit`	`limit`	`limit`

Notably, in the case of **D3**, Depth-First required less time than Beam-Search. This observation can be attributed to the relatively smaller number of features compared to the other datasets. These results demonstrate the impact of the algorithm parameters, namely the number of returned patterns and the depth. As these values increase, the execution time also increases. In our qualitative experiments, we used Beam-Search with a beam width of 50 for **D1** and **D4**, while we exploited results from Depth-First on **D2** and **D3**.

3.5 Discussion

In this chapter, our objective was to introduce the framework of subgroup discovery to a highly practical and valuable application in AIOps, specifically for incident management. We focused on addressing a common challenge in both incident detection and diagnosis: identifying emerging issues and root causes of performance degradation when querying databases on a low-level basis. We initially presented the fundamental components of subgroup discovery necessary to tackle a specific use case. This included the pattern syntax, which defines the search space of potentially interesting patterns for users, as well as the interestingness measures that are based on objective and/or subjective criteria. Additionally, we discussed the mining algorithms used to effectively retrieve the best patterns according to the chosen measure. Next, we introduced our framework which incorporates two supplementary steps. The first step involves a pre-processing phase that enables efficient parsing of SQL queries and the integration of relevant information for this task. The second step encompasses a post-processing phase where we propose a visualization tool. This tool aids practitioners in interacting with the framework and allows them to leverage their expertise to improve the model. Through empirical analysis, we demonstrated how this approach can uncover interesting hypotheses from queries executed on numerous databases.

Subgroup Discovery has the potential for various extensions that can enhance its results. While we have presented a direct application of this framework, our experiments highlighted the need to consider multiple targets in many cases. For example, identifying patterns of queries that return few rows but have long running times falls under the scope of Exceptional Model Mining. Furthermore, although we already employ a rich pattern language, there is an opportunity to leverage the syntactic tree structure of queries to mine more expressive tree patterns, which surpass simple conjunctions of SQL clauses. Lastly, it is crucial to consider subjectivity and practitioner preferences during the mining process. In a nutshell, this chapter serves as an introduction to Subgroup Discovery, showcasing its effective application in important use cases for incident management. However, there are challenges when dealing

Table 3.7: Mapping the contribution `SD-4SQL` to our proposed taxonomy.

Context	Focus Area	Incident Detection and Incident Correlation
	Maintenance Layer	Functional
	Scoop	ERP Software System
Data	Source and Type	SQL Queries ⟶ Tab data Topology ⟶ Tab data Alerting Signals ⟶ Tab data ASH ⟶ Tab data
	Feature Eng	Parsing of SQL queries required
Model	Approach	**Subgroup Discovery**
	Paradigm	Supervised Rule Disocovery
	Metrics	Lift, WRAcc, Median-test, T-Score
	Availability	Data and Code (`https://github.com/RemilYoucef/sd-4sql`)
Particularities		Interpretability, Human in the loop, In context-Evaluation
Contribution		First to apply SD on SQL workload Analysis

with complex data structures such as graphs and hierarchies. Additionally, when the target concept is not simply defined but derived from a complex combination of attributes, it becomes challenging to provide an interestingness measure and an effective search algorithm. Sometimes, it is necessary to perform a search when the target concept itself is modeled as another search space, for instance, when both rule components are patterns. In the upcoming chapters, we will explore these challenges, contributing to both subgroup discovery and AIOps for incident management. In the next chapter, we delve into applying subgroup discovery in the context of explainable AI to summarize explanations for similar predictions made by black box models. In Chapter **5**, we tackle another intriguing problem involving the analysis of Java memory heap dumps, which are typically represented hierarchically as a target concept. We also provide in the following Table 3.7 the positioning of our contribution according to the taxonomy introduced in Section 2.4 and its alignment with the existing landscape.

Chapter 4

Summarizing Interpretable Incident Triage Predictions with Subgroup Discovery

In large-scale systems experiencing a growing number of reported incidents, the need for an efficient Incident Triage process is crucial. Traditional approaches rely on on-call engineers (OCEs) to quickly assess incident severity and determine the appropriate service to address the issue. However, there have been advancements in predictive models that aim to automate these decision-making processes. Specifically, the assignment of incidents to the responsible individuals or teams plays a vital role in this phase. To accomplish this, sophisticated models are utilized to analyze incident reports accurately and make informed decisions. Unfortunately, many of these methods operate as black boxes, obscuring the decision-making mechanisms behind their predictions. This lack of transparency greatly hinders their adoption among practitioners who require a deeper understanding of and justification for these predictions. Nevertheless, recent progress in eXplainable Artificial Intelligence (XAI) offers hope by providing both global explanations for models and, more importantly, localized explanations for individual model outcomes. However, providing an explanation for each outcome to a human user becomes impractical when dealing with a significant volume of daily predictions, especially considering that many incidents are duplicates and can share the same local explanations. To tackle this issue, we propose leveraging Subgroup Discovery, which naturally groups objects that share similar properties according to a target problem while providing a description for each group. Nevertheless, employing Subgroup Discovery poses various challenges. In this case, the target concept is defined as an explanation model rather than a single attribute, necessitating the proposal of an appropriate similarity measure and effective yet scalable algorithms to yield meaningful results.

4.1 Introduction

The need for an effective process of incident triage becomes essential in large-scale systems with an increasing number of incidents reported by monitoring systems and equipment/software users. On the front line, on-call engineers (OCEs) have to quickly assess the severity of an incident, qualify it, ensure that all the required information is provided, and decide which service to contact for palliative and/or curative actions. To expedite and improve these decisions, the state-of-the-art incident assignment methods in recent years have provided a wide range of predictive models. These models aim to predict the appropriate service to target based on the textual information provided in the incident reports, sometimes incorporating background knowledge. Typically, these approaches involve training a classifier using historical incident reports that contain textual information, topology data, or prioritization scores. The trained classifier is then used to assign new incidents [165, 309, 59, 238]. These models usually employ sophisticated deep learning algorithms such as LSTM units, convolutional networks, text embedding approaches, and ensemble models. Unfortunately, such methods create opaque models, often referred to as black boxes, as they do not provide any explanation of their output (outcome or prediction) to the end user. This limitation drastically hinders their deployment in real-world scenarios since practitioners need to be able to understand and justify why a given incident has been routed to a particular service team instead of another. Hence, an important challenge in successfully automating incident triage is gaining practitioners' trust by providing them with an explanation for each model outcome. This explanation can also be part of the incident diagnosis, as it assists with root cause analysis.

The popularity of black box prediction models combined with the crucial need for transparency in many decision processes has led to an unprecedented interest in eXplainable Artificial Intelligence (XAI). Pioneering research work has been conducted to interpret the decisions of sophisticated models by providing what is known as white box models capable of capturing the mechanisms followed by the model to make predictions. According to to [115], these methods can be categorized into global and local explainers. Global methods such as RxREN [20], FIRM [339], and inTrees [85], aim to explain the internal logic of the model by constructing an interpretable model (e.g., a decision tree or a set of decision rules) that approximates the predictions of the original model. These methods offer the advantage of providing a comprehensive explanation for the behavior of the black box model. By learning an interpretable model that covers the entire dataset, it is possible to explain any result of a given input object. However, global methods may fail to explain all facets of the predictive process when the original model is exceedingly complex or when the data contains interactions and dependencies between variables. Moreover, these methods may not perform well when the relationship between the target outcome and the variables is nonlinear. On the other hand, local approaches, such as LIME [254], SHaP [198], Anchors [255] and many others [118, 155, 117] provide an explanation for each prediction of a given input object and have exhibited better fidelity performance compared to global methods. These approaches learn a white box model for each data instance independently based on synthetic data generated in the neighborhood of that data instance. This allows them to figure out how the model behaves locally and better capture the behavior of the black box model. Nonetheless, applying this type of approach to large datasets poses a challenge. When the total number of predictions to be explained is large, the complexity of the approach escalates, and it becomes unfeasible for practitioners to analyze each prediction separately. Few methods have

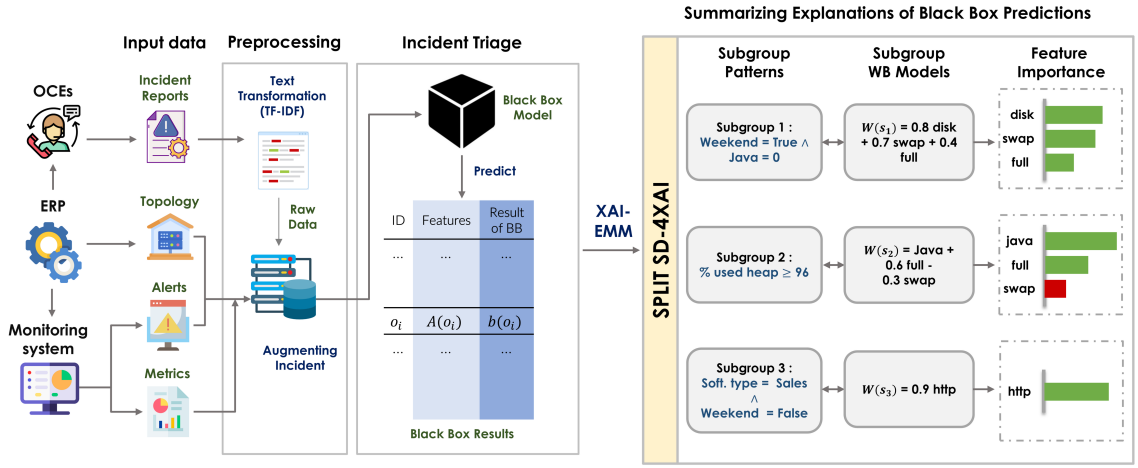


Figure 4.1: Overview of our explainable incident triage framework: (1) Input data are processed to extract relevant features, (2) a black box model uses these features to provide an accurate incident triage, (3) contextualized groups of black box predictions are explained using a Subgroup Discovery approach.

been proposed to address this issue, by grouping similar explanations into clusters [129] or by selecting a subset of representative explanations [254]. While these methods can provide a picture of the different possible explanations of a model, they do not provide the user with the contexts in which each explanation holds.

Hence, we first propose an efficient black-box model based on 170K user-reported incidents from our ERP system that we dealt with over the last 7 years. Then, we propose to take advantage of the Subgroup Discovery approach to explore its ability to group predicted objects into subgroups that support the same explanation. Instead of providing a specific explanation for the prediction of each object, we group objects into a controlled number of subgroups (i). For each subgroup, we provide an explanation that holds for all of its objects (ii). Each subgroup is then associated with a description pattern that differentiates it from the rest of the dataset. This approach is beneficial because it allows the user to interpret not only the black box outcome for each subgroup but also the nature of the objects contained within a subgroup. Formally, our objective is to present a pattern in the form of Pattern $P \longrightarrow$ Explanation e. This explanation is typically represented by a white box model, often a regression model, which allows us to interpret the weights assigned to different features. In this case, the target concept is modeled as a combination of attributes, which pertains to the problem of Exceptional Model Mining. Consequently, we need to introduce an interestingness measure to evaluate the significance of this rule in relation to a global white box model that refers to all the objects in the dataset. Additionally, an algorithm that is both scalable and efficient is required to extract meaningful subgroups with interesting explanations. We believe that this approach presents a novel solution to the challenge of explaining model outcomes, particularly when dealing with a large number of outcomes, as observed in tasks such as Incident Triage or Assignment. To provide a clearer understanding of our approach, Figure 4.1 illustrates the entire process of the proposed methodology.

Roadmap. In Section 4.2, we present the methodology conducted, starting from raw data to evaluations of the proposed black box model. Subsequently, in Section 4.3, we formalize a

novel problem of summarizing explanations for black box outcomes using Subgroup Discovery where we introduce our framework, dubbed `SplitSD-4XAI`. In this section, we also clarify the choices made in developing our framework. Moving forward, we present an extensive empirical study in Section 4.4 that demonstrates the effectiveness of our framework in identifying interpretable subgroups and providing meaningful explanations, specifically in the context of the incident assignment. Finally, in Section 4.5, we conclude this chapter with a summary, discussion of limitations, and perspectives.

4.2 Background and Methodology

We conduct our analysis on a set R of 170k incident reports that concern more than 1k servers over the last 7 years from the Software ERP system of Infologic. Each server contains at least an instance of the monitored ERP software but contains other components that are also supervised, including databases, hardware, and network. We efficiently process incident texts to extract discriminant features, then we can augment them with various attributes that allow contextualizing incidents, such as environmental features (topology), performance metrics, and standard events

4.2.1 Raw data

Incident reports. We define the set $R = \{r_1, ...r_n\}$ of incident reports described by: (1) the title r_{title}, (2) the summary r_{summary} which contains commands, stack-traces, and text written in human language (mainly French), (3) the component $r_{\text{component}}$ in which the incidents happen (e.g., the ERP software, the virtual machine, the storage disk), (4) the incident creation timestamp r_{time}, (5) a Boolean r_{internal} that indicates whether the incident has been reported internally by our company or externally by a customer. In the latter case, we include (6) the customer r_{customer} concerned by the incident.

Environment features. These features describe the environment and the component in which the incident happens. For instance, the ERP software is characterized by its application identifier, its version, and its type among 6 main families (sales, factory, finance, etc.). A virtual machine can be characterized by the number of CPUs, the size of the RAM, the size of swap memory, etc.

Performance metrics. We continuously gather time series data that captures the behavior of multiple components spanning from hardware to the business level. A wide range of memory components, including heap and non-heap memory for Java Virtual Machines running the ERP, as well as RAM and swap usage, are monitored at a high frequency. Additionally, we collect various other metrics to ensure comprehensive observability. These metrics encompass storage utilization, the frequency and average execution time of SQL queries, and the number of users connected to each ERP software instance. The availability of these diverse metrics proves highly valuable, as they provide efficient insights into the context surrounding incidents, facilitating more effective incident triage processes.

Standard events. We capture a wide range of timestamped events that occur within supervised components. These events indicate interactions with users within the ERP software, such as screen openings, data visualizations, and data creations. Additionally, events can

also arise from the automatic execution of scheduled tasks. Furthermore, various components generate distinct types of events, such as garbage collections in the Java Virtual Machine, network latency, and system alerts.

4.2.2 Text transformation

Incident titles and summaries are processed using conventional NLP techniques to extract relevant textual features. Most of these steps are achieved with the spaCy[1] Python library. The process begins with tokenization to extract a bag of words representation. Notably, N-grams representation is not utilized as it did not improve prediction accuracy in our specific use case. Special characters, stop words, and noisy terms (e.g., those with fewer than 3 characters) are removed, along with words that appear frequently in incident reports but lack discriminative value. Numbers are retained as they can provide useful information, such as error and response codes. Subsequently, each remaining token undergoes lemmatization, which involves reducing words to their base or dictionary form (lemma) that represents the canonical form of the word. Finally, we compute the `Term Frequency / Inverse Document Frequency` (`tf-idf`) for the remaining terms in each incident report. This calculation captures the importance of a term within a specific document (incident report) relative to its frequency across the entire corpus of incident reports. We retain only the top 10K terms in both r_{title} and r_{summary}. The choice of this term count is made empirically, based on maximizing prediction performance. The `tf-idf` formula is as follows:

$$\texttt{tf-idf}(t, r) = \texttt{tf}(t, r) \times \texttt{idf}(t)$$

where $\texttt{tf}(t, r)$ is the frequency of the term t in the report r, $\texttt{idf}(t) = \log\left(\frac{|R|}{\texttt{df}(t)}\right)$ is the inverse document frequency, and $\texttt{df}(t)$ is the numebr of incidents containing the term t.

4.2.3 Black box model for incident triage

We unify the different data sources into a dataset defined by a tuple (O, A, Y), where $O = \{o_i\}_{1 \leqslant i \leqslant n}$ is a set of objects that refer to the historical incident reports, $A = \{a_j\}_{1 \leqslant j \leqslant m}$ is a set of descriptive attributes, and the classes $Y = \{y_1, ..., y_n\}$ that represent the services assigned to each incident. An incident is assigned to a service (a class) $y \in \{Class_1, ..., Class_p\}$ where p is the total number of services. These notations are illustrated in Table 4.1 with a dataset containing 7 objects $O = \{o_1, ..., o_7\}$, referring to 7 incidents, each of them described by 10 attributes and a class y. For instance, the attributes $\{a_1, ..., a_5\}$ are numerical and they provide the `tf-idf` of terms that characterize r_{title} and r_{summary}. The attribute `weekend` (a_6) indicates whether the incident has happened during a weekend. Other attributes correspond to some environment features, alerts, and metrics.

The incident triage task consists in predicting the class y (the service) of an incident report r based on its attributes $\{a_1, ..., a_m\}$. It is noteworthy that, since our goal is to make the prediction at the beginning of the incident life-cycle, all the considered attributes are available from the very first moment when the incident report is created. We consider training and validation subsets of objects $O_T, O_V \subseteq O$ that we use to train prediction models denoted by b. The output $b(o) = \{b(o)_1, ..., b(o)_p\} \in [0, 1]^p$ provides a probability distribution

[1] `https://spacy.io/`

Table 4.1: Toy Example of a dataset of incidents (O, A, Y).

O	$\mathbf{r_{title}}$		$\mathbf{r_{summary}}$			$\mathbf{r_{time}}$	**ENV features**		**Alerts**	**Metrics**	**Service**
	a_1 disk	a_2 swap	a_3 full	a_4 java	a_5 http	a_6 weekend	a_7 Soft. version	a_8 Soft. type	a_9 Memory usage	a_{10} % used heap	y class
o_1	0.7	0	0.4	0	0	True	1	Sales	-	60	**TEC**
o_2	0	0.8	0.3	0	0	True	3	Sales	Blocker	50	**TEC**
o_3	0.5	0	0	0	0.6	True	2	Factory	-	60	**TEC**
o_4	0	0.5	0.9	0.6	0	True	3	Factory	Critical	97	**OT**
o_5	0	0.7	0.6	0	0	False	1	Sales	Critical	96	**OT**
o_6	0.1	0	0	0.6	0.6	False	2	Sales	Alarm	85	**OT**
o_7	0.1	0	0	0	0.9	False	1	Sales	-	60	**OT**

over the p predicted output classes. Inspired by recent work [238, 104, 59] which show in most cases that best results are achieved by DNN and ensemble models, we have opted for an LSTM-attention architecture as depicted in Figure 4.2. We also evaluated against the following black box methods: Random Forest, XGBoost. Additionally, we have considered the logistic regression and multinomial Naive Bayes as white box models to confirm that the usage of black box models effectively improve the prediction quality. We have tested each model in three different scenarios: (1) **TOP1**: predict the true service, (2) **TOP2**: predict the two most probable services, i.e., if the true service belongs to this TOP2, the prediction is considered correct, (3) **TOP3**: predict the three most probable services. Table 4.2 reports the results obtained on the validation set O_V. We use the most common performance metrics in multi-label classification i.e., Precision, Recall and F1 score in their weighted average formula. Interestingly, we observe that in all considered scenarios, our proposed model achieves the best performances compared to other models and improves the F1 score of Logistic Regression by 9%. In addition, the model shows to be very accurate when it comes to predicting the most likely services to deal with the incident. Out of 30 different services, the model manages to reach an F-score of 0.96 when the correct service belongs to the 3 most probable services predicted by the black box model (the TOP3 setting).

Table 4.2: Performance comparison of prediction models in the incident triage task.

	DNN			**RF**			**XGBoost**			**Naive Bayes**			**Logistic Regression**		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
TOP1	**0.84**	**0.77**	**0.82**	0.74	0.72	0.73	0.75	0.75	0.74	0.74	0.75	0.74	0.75	0.72	0.73
TOP2	**0.92**	**0.91**	**0.91**	0.89	0.88	0.88	0.89	0.89	0.89	0.89	0.90	0.89	0.89	0.88	0.89
TOP3	**0.97**	**0.96**	**0.96**	0.94	0.94	0.93	0.94	0.94	0.94	0.95	0.94	0.94	0.94	0.94	0.94

4.2.4 Explaining Black box outcomes

The single outcome explanation problem. This problem consists in giving an explanation e for the decision $b(o) = \{b(o)_1, ..., b(o)_p\} \in [0, 1]^p$ related to a specific object $o \in O$ where e belongs to a human-interpretable domain E [115].

White box model. In order to extract an explanation $e \in E$ for a decision $b(o)$, one of the most popular methods is to train an interpretable model that learns to imitate the decisions of b specifically in the neighborhood of the object o. This interpretable model

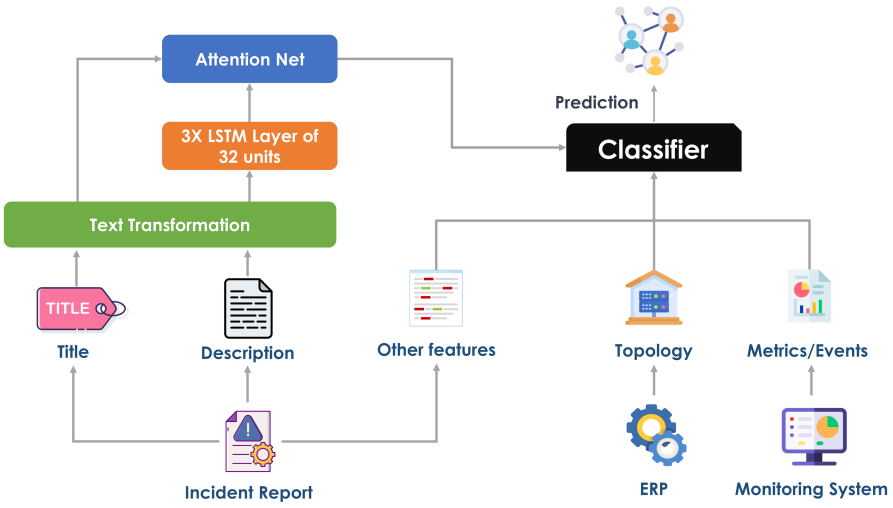


Figure 4.2: Overview of the LSTM-attention architecture employed for incident triage model.

is called a *white box model* and it is denoted w. New objects are synthetically generated from the neighborhood of o, and the model w is trained to mimic decisions of b on these objects. One may use linear regression as a white box model. We provide an illustration of the process using an example to demonstrate how the widely recognized LIME (Local Interpretable Model-Agnostic Explanations) method [254] works, as shown in Figure 4.3. In Panel (A), the two regions represent two classes that cannot be accurately predicted using a linear or logistic regression model. To explain the predictions for a specific data point, we generate a synthetic neighborhood around that particular instance. Subsequently, the black box model is applied to make predictions for this newly generated synthetic data. A linear regression model is then trained within this neighborhood, similar to a tangent in complex functions, considering the predictions from the black box model (Panel D). Finally, the interpretations are derived based on the weights of the linear model. In our case, since we deal with hundreds of attributes which are strongly linearly correlated, linear regression models generally tend to overfit and the model coefficients will have large variance, thus making the model unreliable. Therefore, we need to consider regularization techniques that shrink the linear model coefficients, and take into consideration the case where the number of objects to explain is less than the number of attributes used in explanation. In our settings, we use Ridge regression which penalizes the sum of squared coefficients (L_2 penalty). While Lasso regression is more appropriate to achieve sparsity, it has been observed that if predictors are highly correlated, the prediction performance of the lasso is dominated by ridge regression [340]. Moreover, Lasso solution is not uniquely determined when the number of attributes is greater than the number of objects.

Synthetic neighborhood. For a given object $o \in O$, we denote by $N(o)$ the set of synthetically generated instances in the neighborhood of o, plus the object o. As already highlighted, to explain o, we train a model w that imitates b on the set $N(o)$. For instance, as shown in Figure 4.3, the LIME approach initially employs a uniform distribution for each attribute a_j to generate synthetic data instances (Panel B). Then, an exponential smoothing technique

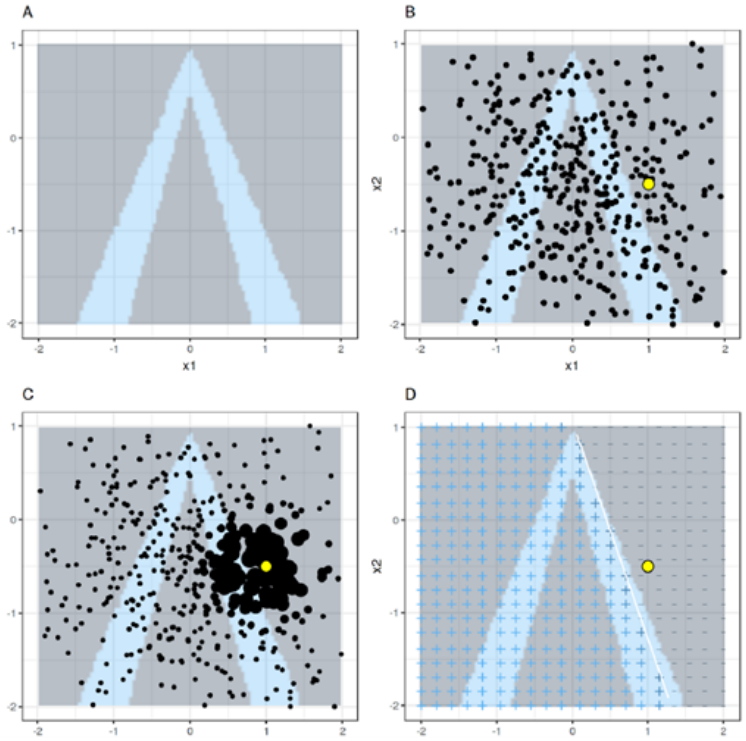


Figure 4.3: Process of explaining predictions using LIME approach. Figure from [217].

Table 4.3: Summarizing explanations of black box predictions of incidents from Table 4.1 with Subgroup Discovery. The number of explanations is reduced from 7 to 3 by grouping objects in contextualized subgroups supporting the same explanations.

O	**Pred. $b(o)_1$ of TEC**	**Pred. $b(o)_2$ of OT**	**Local model $w(o)$ of majority class**	**Subgroup model**	**Subgroup description**
o_1	**0.9**	0.1	$\text{disk} + 0.1 \cdot \text{swap} + 0.5 \cdot \text{full}$	$0.8 \cdot \text{disk} + 0.7 \cdot \text{swap} + 0.4 \cdot \text{full}$	$\text{weekend} = \text{True} \wedge \text{java} = 0$
o_2	**0.8**	0.2	$0.1 \cdot \text{disk} + \text{swap} + 0.4 \cdot \text{full}$		
o_3	**0.6**	0.4	$\text{disk} - 0.5 \cdot \text{http} + 0.3 \cdot \text{full}$		
o_4	0.3	**0.7**	$0.9 \cdot \text{java} + 0.5 \cdot \text{full} - 0.2 \cdot \text{swap}$	$\text{java} + 0.6 \cdot \text{full} - 0.3 \cdot \text{swap}$	$\%\text{used heap} \geqslant 96$
o_5	0.2	**0.8**	$\text{java} + 0.6 \cdot \text{full} - 0.3 \cdot \text{swap}$		
o_6	0.2	**0.8**	$0.8 \cdot \text{http}$	$0.9 \cdot \text{http}$	Soft. $\text{type} = \text{Sales} \wedge$ $\text{weekend} = \text{False}$
o_7	0.1	**0.9**	$0.9 \cdot \text{http} - 0.1 \cdot \text{disk}$		

is applied to assign greater importance to instances that are in proximity to the instance of interest based on their Euclidean distance (Panel C). The quality of w to mimic the behavior of b is assessed with some fidelity measures. The fidelity can be assessed in terms of functions such as MSE and F1-score, evaluated using the outcome of the black box model.

In Table 4.3, we show $b(o)_1$ (resp. $b(o)_2$) the probability of the class TEC (resp., OT) predicted by the black box model. The majority class is TEC in $\{o_1, o_2, o_3\}$, whereas it is OT in the remaining objects. The table gives the local model w trained to mimic the prediction of the majority class by the black box in the neighborhood of each object $o \in O$. For example, the local model that provides an explanation for the prediction of TEC for o_3 is $w(o) = \text{disk} - 0.5 \cdot \text{http} + 0.3 \cdot \text{full}$. This means that the higher the `tf-idf` of the words "disk" and "full" in the incident report, the higher the probability of the TEC class, in contrast, the higher the `tf-idf` of "http", the lower the probability of TEC.

In practice, we may have a large set $O_E \subseteq O$ of objects whose decisions $\{b(o) \mid o \in O_E\}$ need to be explained. Providing a specific explanation for each prediction is overwhelming for the user, and it may be even impossible for her to dig into each explanation separately. In addition, many objects that share certain properties in common may have similar explanations. i.e., an explanation can be valid for a group of objects. We aim to reduce the number of explanations by partitioning the objects into subgroups $s \subseteq O_E$ that can support the same explanation. However, we need to be able to characterize these subgroups by some common interpretable description or pattern that separates them from the rest of the dataset. To this aim, we use the concepts of Subgroup Discovery which are described hereafter.

4.3 Summarizing black box explanations with SD

None of the existing approaches have harnessed the potential of the Subgroup Discovery framework for the purpose of explaining black box predictions. In our pursuit to leverage this approach and enhance the complexity of local explainers while providing them with an interpretable context, we encountered several complex challenges. These challenges revolve around the structure of the mined data, the interestingness of subgroups, and the development of a scalable mining algorithm that optimizes a novel interestingness measure within the Subgroup Discovery (SD) framework. In the following sections, we begin by defining our pattern language, which delineates the search space for subgroups. Subsequently, we introduce our measure to assess the interestingness of each subgroup. This measure evaluates the fidelity of the subgroup model in relation to the black box model, as well as the deviations from a global model. Finally, we formally present our problem of summarizing black box explanations using Subgroup Discovery and present our mining algorithm.

4.3.1 Pattern language and subgroup model target

To define the search space of subgroup descriptions (i.e., patterns), we utilize the pattern language $\mathcal{L}$, employing the same notations as Chapter 3. The pattern language $\mathcal{L}$ is based on the descriptive attributes A and is expressed as $\mathcal{L} = \times_{i=1}^{m} sel_i$, where sel_i represents the set of all possible selectors for the attribute a_i. However, in the subgroup pattern, we have considered terms encoded in tf-idf as Boolean variables, indicating their presence or absence. For convenience, we refer to a subgroup as s instead of a pattern P, i.e., $s = sg(P)$.

Instead of providing an explanation for the prediction of each $o \in O_E$, we aim to group these objects into a limited number of subgroups that cover all the objects of O_E to explain, and for each subgroup s, we provide an explanation that holds for all its objects.

In Table 4.3, we give the predicted probability $b(o)_i$ for each class. The prediction of each object o is then explained by a local model w trained to mimic the behavior of the black box model in the neighborhood of o. This model w estimates the outcome of b using a linear equation between `tf-idf` of terms appearing in the corresponding incident reports.

We can partition the data into three subgroups whose objects can support the same explanation. For example, the first subgroup refers to all incidents that have happened in the weekend and that do not contain the word java in their text. Their predicted probability of TEC can be explained by the same relation: $(0.8 \cdot \text{disk} + 0.7 \cdot \text{swap} + 0.4 \cdot \text{full})$. Doing

this, we summarize 7 different local models in only 3 subgroups models along with a pattern that uniquely identifies the objects explained by each model. To ensure that a subgroup model holds for all the objects of the subgroup, we seek to minimize the error made by the subgroup model while imitating the black box model on the neighborhood of each object of the subgroup. These notions are formalized below.

Subgroup model. A subgroup model w_s is a white box model used to explain the predictions of b on the objects of a subgroup s. It is trained on the neighborhoods of the objects of s. The neighborhood generation process is described later in 4.3.3.1.

4.3.2 Interestingness Measure

In our approach, we focus on extracting subgroups that maximize fidelity with respect to the black box predictions. In simpler terms, our objective is to identify subgroup patterns whose subgroup models demonstrate a higher level of similarity to the behavior exhibited by the black box model for those subgroup instances. We use the Sum of Squared Errors to evaluate the fidelity of a white box model w_s, fitted on a subgroup s and its objects neighborhood, to imitate a black box model b:

$$L(s, w_s, b) = \sum_{o \in s} \sum_{o' \in N(o)} \sum_{i=1}^{p} \left(b(o')_i - w_s(o')_i\right)^2 .$$

The global loss for a set of subgroups $S = \{s_1, s_2, ...\} \subseteq \mathcal{S}$ along with their fitted models $W = \{w_{s_1}, w_{s_2}, ...\}$ is defined as: $L(S, b) = \sum_{i=1}^{|S|} L(s_i, w_{s_i}, b)$.

Controlling the number of subgroups. To control the total number of collective explanations of the predictions $\{b(o) \mid o \in O_E\}$, we propose to upper bound the number of returned subgroups with a threshold $K \in \mathbb{N}$. The goal is thus to find a subgroup set $\{s_1, s_2, ...\}$ of size at most K, whose fitted white models $\{w_{s_1}, w_{s_2}, ...\}$ minimize the loss function with respect to the black box model. The problem is formalized as follows:

> **Problem 2** (Summarizing explanations with SD)**.** Let $O_E \subseteq O$ be a subset of objects whose predictions need to be explained, and b the black box model used for prediction. Given a user-specified threshold $K \in \mathbb{N}$ representing the maximum number of explanations, find a subgroup set $S = \{s_1, s_2, ...\}$ with their fitted white box models $W = \{w_{s_1}, w_{s_2}, ...\}$ such that (1) $|S| \leqslant K$, (2) the subgroup set covers all the objects to explain: $\bigcup_{s \in S} s = O_E$, and (3) the global loss for the subgroup set is minimized: $S = argmin_{S' \subseteq \mathcal{S}} L(S', b)$.

4.3.3 Search Algorithm

The problem of summarizing explanations with SD is NP-Hard. This can be proven by reducing the NP-Complete problem of weighted set cover in a polynomial time to Problem 2: each set corresponds to a subgroup, and the set weight is represented by the loss $L(s, w_s, b)$ of the corresponding subgroup. Thus, providing a scalable approach that finds the best solution to Problem 2 is not possible. We propose to use an efficient heuristic strategy detailed in Algorithm 1 (`SplitSD-4XAI`) and empirically prove its performance. This algorithm starts by generating the neighborhoods $N(o)$ used to train local models for each object $o \in O_E$,

using GenerateNeighbors explained in 4.3.3.1. Then, it constructs a non-overlapping set of subgroups using a split based strategy. It begins with the subgroup set $S = \{O_E\}$ that contains a subgroup covering all the objects of O_E. In each iteration, and given the current set of subgroups S, one of the subgroups of S is split into two subgroups that minimizes the overall loss. The split is applied for one of the attributes $a \in A$. This procedure is done iteratively until the number of subgroups K is reached, or, until there is no additional possible improvement of the loss, as detailed in 4.3.3.2.

Algorithm 1: `SplitSD-4XAI`

Input: O_E a set of objects, b a black box prediction model, K a threshold on the number of subgroups.

Output: $S \subseteq \mathcal{S}$ a subgroup set that covers all the objects O_E, W the set of white box models associated with the found subgroups.

for $o \in O_E$ **do**
  $N(o) \leftarrow$ GenerateNeighbors(o)
$S \leftarrow \{O_E\}$
$W \leftarrow dict(\{\})$ // W is a dictionary
$W[O_E] \leftarrow w_{O_E}$ // w_{O_E} is the white box fitted to the subgroup O_E
$improve \leftarrow True$, $splits \leftarrow \varnothing$, $newSubgroups \leftarrow \{O_E\}$
while $|S| \leqslant K$ *and* $improve$ **do**
  // Compute the best splits for the new subgroups:
  for $s \in newSubgroups$ **do**
    $(a, v) \leftarrow argmin_{a \in A, v \in R_a} L(s[a \leqslant v], w_{s[a \leqslant v]}, b) + L(s[a > v], w_{s[a > v]}, b)$
    $splits \leftarrow splits \cup \{(s, a, v)\}$
  // Choose the subgroup split that leads to the minimum loss:
  $(s, a, v) \leftarrow argmin_{(s,a,v) \in splits} L(S \backslash \{s\} \cup \{s[a \leqslant v], s[a > v]\}, b)$
  if $L(s[a \leqslant v], w_{s[a \leqslant v]}, b) + L(s[a > v], w_{s[a > v]}, b) < L(s, w_s, b)$ **then**
    $S \leftarrow S \backslash \{s\} \cup \{s[a \leqslant v], s[a > v]\}$
    remove $W[s]$
    $W[s[a \leqslant v]] \leftarrow w_{s[a \leqslant v]}$
    $W[s[a > v]] \leftarrow w_{s[a > v]}$
    $newSubgroups \leftarrow \{s[a \leqslant v], s[a > v]\}$
    $splits \leftarrow splits \backslash \{(s, a, v)\}$
  else
    $improve \leftarrow False$
return (S, W)

4.3.3.1 Neighborhood generation

The goal of this step is to sample a set of neighbors $N(o)$ for each object $o \in O_E$, using a locality-aware sampling strategy. Many approaches have been proposed to address this problem [254, 114, 116]. As this part of the process is not the main concern of our study, any of these approaches can be directly used in GenerateNeighbors.

However, the application of the LIME explainer procedure, for example, does not appear to be effective due to its high sensitivity to the parameters of the exponential smoothing function. Consequently, it can lead to significantly different local models, resulting in varying interpretations. More formally, the function used is defined as $\pi(o, o') = \exp\left(\frac{-D(o,o')}{\sigma^2}\right)$, where D represents the Euclidean distance and σ represents the kernel width, indicating the size of the neighborhood. A smaller kernel width implies that an instance must be very close to influence the local model. Referring to the LIME implementation[2], this parameter is arbitrarily set to $0.75 \cdot |A|$, which fails to generalize well to many other cases. Additionally, there is no systematic mechanism provided to determine the optimal kernel width based on the specific use case. An interesting example presented in [217] (Figure 4.4) demonstrates this issue. They attempt to explain a prediction of an instance $x = 1.6$. The black box model's predictions based on a single feature are represented by a thick line, while the distribution of the data is indicated by rugs. Three local models with different kernel widths are computed, resulting in highly disparate regression models. Consequently, it becomes challenging to determine whether the X feature has a negative, positive, or negligible effect on the given data instance. This example focuses on a single feature, but the challenge becomes even more complex in high-dimensional feature spaces.

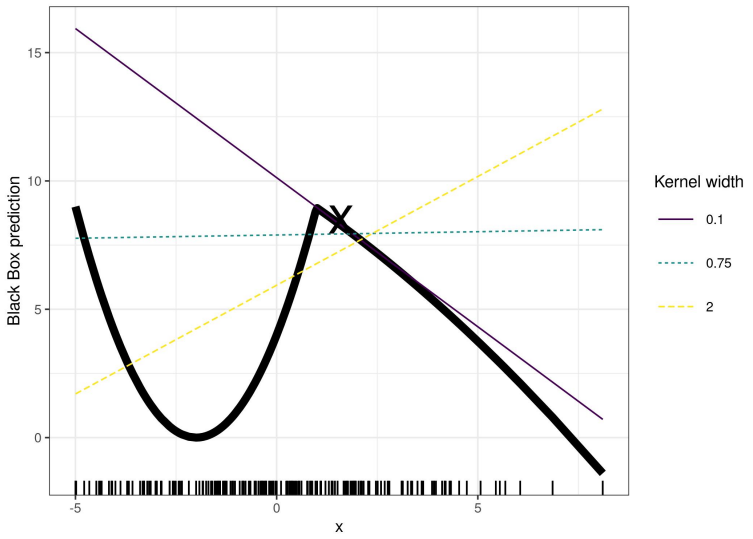


Figure 4.4: Parameter sensitivity problem in LIME neighborhood sampling procedure [217].

In order to limit the bias due to this step, we use a simple yet efficient sampling approach considering two main conditions (1) the closer a point o' is to o, the higher the chance to sample it in $N(o)$, (2) the correlation between the different attributes and the variance of each attribute need to be taken into account in order to sample more realistic objects. To provide further justification for these conditions, particularly the second one, we present in Figure 4.5 a visual representation of different sampling results. These outcomes arise due to variations in correlation and variance, directly influencing the importance of features in the resulting local models. One effective approach to incorporate both variance and correlation in the sampling process is by utilizing the covariance function between two attributes:

[2] `https://github.com/marcotcr/lime/blob/master/lime/lime_tabular.py`

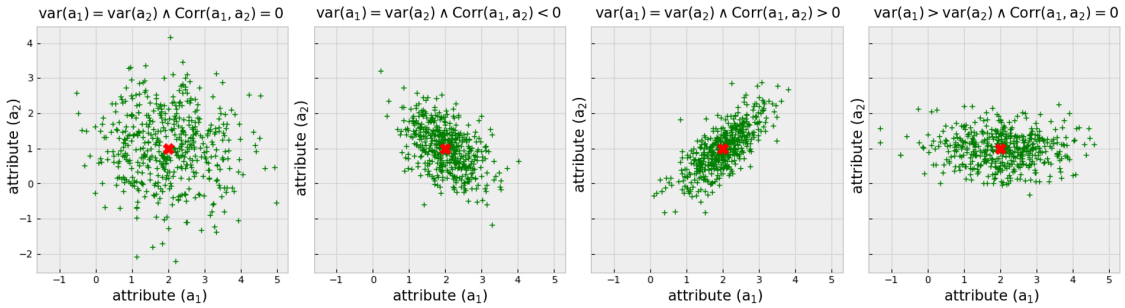


Figure 4.5: Different sampling results with different values of variance and correlation.

$$Cov(a_i, a_j) = \frac{1}{|O| - 1} \sum_{k=1}^{|O|} (a_i(o_k) - \bar{a}_i)(a_j(o_k) - \bar{a}_j)$$

In order to generate an object $o' \in N(o)$, the attribute values $A(o)$ are drawn from a multivariate normal distribution $\mathcal{N}(A, \frac{\Sigma}{z})$ centered in $A(o)$ with a covariance $\frac{\Sigma}{z}$, where Σ is the covariance matrix of (O, A) and $z \in \mathbb{N}$ is a parameter that shrinks the original covariance Σ to the locality of o. Since the multivariate Gaussian distribution generates values in $\mathbb{R}$ for all attributes, categorical features need first to be converted into numerical values. In ordinal data (e.g., Memory usage alert), while encoding, we should retain the information regarding the order in which the category is provided. Nominal features (e.g., Soft. type) are encoded so that each category is mapped with a binary variable containing either 0 or 1 using one hot encoding. Then, after the sampling process, these values are discretized. Particularly, for nominal attributes, the category having the closer value to 1 among other categories of the same nominal attribute is set to 1, otherwise 0.

4.3.3.2 Optimizing L with a split-based strategy

Let us now detail the approach used to identify a subgroup set $S = \{s_1, s_2, ...\}$ that optimizes the loss, while satisfying the constraints of coverage ($\cup_{s \in S} s = O$) and maximum size ($|S| \leqslant K$). In what follows, for a given subgroup s, we use the notation $s[a_i \leqslant v]$ and $s[a_i > v]$ to split s into two subgroups with respect to the values of attribute a_i. By considering that $\lhd$ corresponds either to $\leqslant$ or $>$, we define $s[a_i \lhd v] = \{o \in s \mid a_i(o) \lhd v\}$. Notice that if we split a subgroup s with a Boolean attribute $a \in A$, there is only one possible split, that is $s[a_i \leqslant 0]$ and $s[a_i > 0]$. Nominal attributes are transformed into a one hot representation, and are then treated exactly as Boolean attributes.

Algorithm 1 (`SplitSD-4XAI`) describes the different steps of this approach. The subgroup set is stored in S, and the corresponding white box models are kept in a dictionary W. S is initialized with a subgroup O_E that covers all the objects to explain. The variable *splits* stores the best split for each subgroup $s \in S$. This variable is updated in each iteration by computing the best splits of the newly added subgroups kept in *newSubgroups* (Line 9 to Line 11). Then, the subgroup s whose split reduces the loss the most is selected. It is removed from S and replaced by the subgroups resulted from this split, i.e. $s[a \leqslant v]$ and $s[a > v]$. This loop is repeated until $|S| = K$, or until there is no further split that reduces the loss. Particularly, since the used loss function is the SSE whose optimization is convex for a linear

model, a new split will either reduce $L(S)$ or let it unchanged, but it will never increase it. In fact, this is guaranteed for any model whose optimization is global, such as models with a convex loss function (linear regression, ridge regression, LASSO, etc.), as proven by the following property.

Property 1. Let s_0, s_1, s_2 s.t. $s_0 = s_1 \cup s_2$ and $s_1 \cap s_2 = \varnothing$, then we have: $L(s_1, w_{s_1}, b) + L(s_2, w_{s_2}, b) \leqslant L(s_0, w_{s_0}, b)$.

Proof. This can be proven by contradiction. Let us consider that the inequality does not hold. Then, $L(s_1, w_{s_1}, b) + L(s_2, w_{s_2}, b) > L(s_0, w_{s_0}, b)$. As $L(s_0, w_{s_0}, b) = L(s_1, w_{s_0}, b) + L(s_2, w_{s_0}, b)$, we have $L(s_1, w_{s_1}, b) + L(s_2, w_{s_2}, b) > L(s_1, w_{s_0}, b) + L(s_2, w_{s_0}, b)$. Two cases are then possible:

- $L(s_1, w_{s_1}, b) > L(s_1, w_{s_0}, b)$, which means that w_{s_1} is not the best model that fits s_1, which is absurd because w_{s_1} is a global optimal solution of L on s_1.
- $L(s_1, w_{s_1}, b) \leqslant L(s_1, w_{s_0}, b)$, thus we have necessarily $L(s_2, w_{s_2}, b) > L(s_2, w_{s_0}, b)$. Following the same logic, this implies that w_{s_2} is not the best fit for s_2, which is also absurd.

□

4.4 Experiments

An experimental study was carried out to assess the efficacy of `SplitSD-4XAI` in summarizing explanations for black box decisions within the domain of incident triage. The primary objective of these experiments was to address the following research questions:

- ✘ **RQ1**: Do subgroup models provide good explanations, in other words, are explanations of subgroup models *faithful* to the black box model predictions?
- ✘ **RQ2**: Are subgroup models *human interpretable* and do they help practitioners understand the black box results?
- ✘ **RQ3**: Are subgroup models *different* from each other?

4.4.1 Experimental Setup and Baselines

We have collected 170k incident reports involving more than 1k servers over the last 7 years. Although most of the data types introduced in Section 4.2 have been used in these experiments, metrics, and alerts have been as they cover only incidents of the last few months. Once the data is processed and encoded, we split it randomly into training (65%), validation (10%), and test set (25%). The results of the accurate black box model used for triaging are provided in Table 4.2. The distribution of incident reports across different services is extremely imbalanced, with the most popular services having thousands of incidents, while other minority services were rarely called upon. We randomly select from the test set 2000 incidents to be summarized in no more than 200 subgroups with their explanations. For that, we apply `SplitSD-4XAI` with a neighborhood size of 250 for each object ($|N(o)| = 250$). The

complete process requires about 3 hours to execute when the number of subgroups $K = 200$. Throughout these experiments, we compare `SplitSD-4XAI` with two baselines:

1. **Global white box** (`global-wb`): This method consists in training a white box model on the set of data that we need to explain to globally approximate the decisions of the black box model. The aim of this comparison is to evaluate the effectiveness of a global white box model in approximating a black box model and determine the extent to which we can enhance its performance using `SplitSD-4XAI`.

2. **Local white box** (`local-wb`): In this approach, we adopt the training of local white box models to explain each data object individually. Following a methodology similar to LIME [254], we make use of our proposed local neighborhood generation method to ensure a fair comparison between `SplitSD-4XAI` and the `local-wb` model. As a result, we obtain a separate model for each object that requires an explanation. The purpose of this comparison is to evaluate whether it is possible to achieve comparable explanatory quality results with a small number of explanations, as opposed to using a larger number.

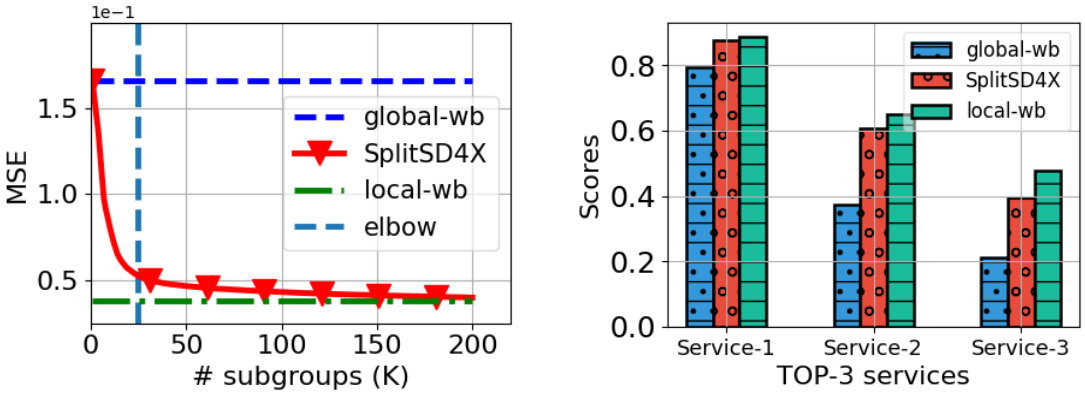
4.4.2 Experimental Results

In what follows, we present the results obtained from the evaluation of the scenarios related to the criteria defined previously.

RQ1: We assess whether `SplitSD-4XAI` effectively identifies subgroups whose associated models accurately mimic the decisions of the black box model. Our objective is to validate whether the subgroup models provided by `SplitSD-4XAI` explain the behavior of the black box model while maintaining its performance. Initially, we analyze this by calculating the Mean Squared Error ($\texttt{MSE} = \frac{\texttt{SSE}}{|O_E|}$) between the predictions made by `SplitSD-4XAI` and the black box model b, considering different numbers of computed subgroups K. The results, depicted in Figure 4.6a, also include the `MSE` values obtained from the `global-wb` and `local-wb` methods as reference. Notably, as the number of subgroups increases, the `MSE` decreases significantly. Interestingly, the largest improvement occurs even with a small number of subgroups. To determine the optimal number of subgroups $K^\star$, where the fidelity closely matches that of the `local-wb` method, and further increasing K does not significantly enhance fidelity, we employ the elbow technique using the Kneed package[3]. We demonstrate that with just 25 subgroup models, we can substantially enhance the fidelity compared to the `global-wb` method and achieve a score that closely approximates that of the `local-wb` method, which employs 2000 models.

In a second analysis, we compare `SplitSD-4XAI` with two baselines using the `F1-score` metric. This score evaluates the extent to which each approach is capable of replicating the service predictions made by the black box model. Specifically, we assess the `F1-score` for the three most probable services predicted by each model and compare them to the black box model's predictions. For example, when evaluating **Service-2**, we consider the services with the second-highest probabilities and compare them to the second-most probable services predicted by the black box model. Our evaluation focuses on only 25 subgroup models, and the results are presented in Figure 4.6b. The `F1-scores` achieved by `SplitSD-4XAI` consistently

[3] `https://github.com/arvkevi/kneed`



(a) `MSE` of `global-wb`, `local-wb`, and `SplitSD-4XAI` (with different K).

(b) `F1-score` of `global-wb`, `local-wb` and `SplitSD-4XAI` with $K^\star$ found by `elbow` technique.

Figure 4.6: Quality of explanations of black box outcomes.

outperform those of the `global-wb` baseline. Moreover, with just 25 subgroups, we achieve nearly comparable `F1-score` results as the `local-wb` method (0.87 for `SplitSD-4XAI` with 25 models compared to 0.88 for `local-wb` in relation to **Service-1**).

These results demonstrate that `SplitSD-4XAI` is able to significantly reduce the number of explanations while keeping them faithful to the black box decisions.

RQ2: Our approach aims to group predicted objects into subgroups with the following objectives: (1) Each subgroup should possess a distinct description that sets it apart from the rest of the data, and (2) objects within the same subgroup should support a same consistent explanation. To achieve this, we provide a description (pattern) for each subgroup, along with its corresponding model. From this model, we derive human-interpretable explanations that assist practitioners in comprehending the rationale behind predicting one service over another. We accomplish this by identifying the most significant features of the model based on the ridge model coefficients. The contribution of each feature in predicting the analyzed class is computed as the ratio between the absolute value of the feature coefficient and the sum of the absolute values of all model coefficients. It is important to note that the relevance of feature importance is contingent upon the fidelity of the subgroup models to the black box. The more accurately the subgroup model emulates the behavior of the black box, the greater our confidence in the explanation based on the coefficient values.

Figure 4.7 presents the descriptions and explanations for four distinct subgroups, each corresponding to the most predicted service within the subgroup. We specifically chose the most popular and frequently requested services for analysis. As an example, consider the subgroup (s_3 : summary_stats $= 0 \wedge$ summary_stock > 0), which comprises incident reports characterized by a `tf-idf` value for the term *stock* greater than 0, while excluding the term *stats* in their descriptions. This subgroup is predominantly dominated by the service ST, representing Stock-related incidents (approximately 41% of the incident reports). In the first subgroup, our focus lies in explaining predictions for the sales service concerning incidents reported between 12 p.m. and 11 p.m. that do not contain terms related to statistics or stock in their summaries. The feature importance plot highlights terms that exhibit a strong

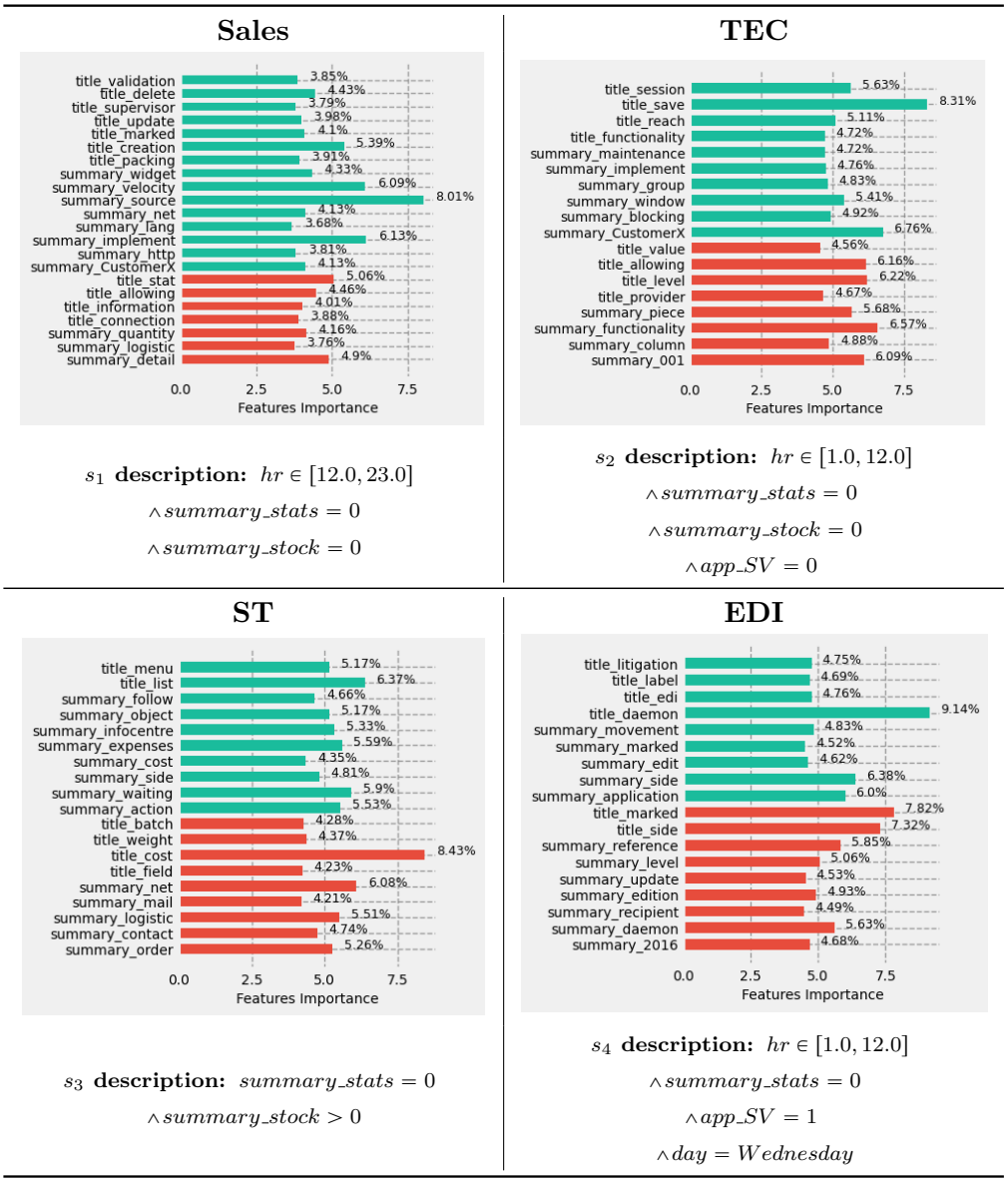


Figure 4.7: Subgroup examples: Patterns and the most important features of the subgroup models for specific services. Green color (resp. red) corresponds to features that contribute positively (resp. negatively) to predicting the analyzed service.

positive correlation with the sales context, such as *creation*, *update*, *validation*, and *packing* of orders. On the other hand, terms like *velocity* increase the likelihood of the sales service being requested. However, terms such as *logistic* and *connection* decrease this probability in favor of other services. Similarly, in the case of s_2, we observe that incidents assigned to the Technical team (TEC) exhibit discriminative terms such as *save*, *session*, and *blocking*. For the subgroup model of s_3, we find that terms related to stock (e.g., *expenses*, *cost*, and *menu*) are included. The last example is particularly intriguing as it pertains to the subgroup description and the quality of the associated explanation when predicting the EDI (Electronic

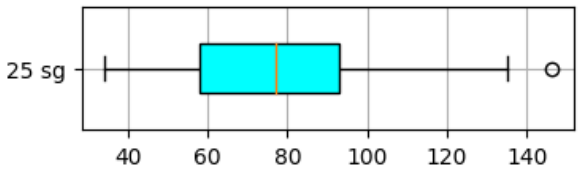


Figure 4.8: Distribution of incidents on the subgroups.

Data Interchange) service. Specifically, when considering supervision servers, we confirm that numerous issues reported to this service are related to the *daemon*, as all EDI operations are performed by daemons. Furthermore, explanations not only aid in comprehending and interpreting the results of the black box model, but they can also shed light on its limitations. In the second example, we observe that the term *functionality* exhibits a positive contribution in the incident title, but a negative contribution in the summary. Such an explanation allows us to gain a deeper understanding of the behavior of the black box model and offers insights for potential improvements.

RQ3: Another important question to address is whether the explanations provided by `SplitSD-4XAI` for the identified subgroups are diverse. If most subgroups have similar explanations, there may still be room for further summarization, indicating that `SplitSD-4XAI` might not have efficiently captured the necessary distinctions. To investigate this, we examine the similarity between the linear models associated with the subgroups identified by `SplitSD-4XAI` when the number of subgroups is set to 25. To measure the similarity between the explanations of two subgroups, denoted as s_1 and s_2, we employ the cosine function, which operates on the coefficient vectors of the corresponding linear models, w_{s_1} and w_{s_2}. Our findings reveal a significant level of dissimilarity among a large proportion of subgroup pairs. Specifically, approximately 93% of the subgroup pairs exhibit a similarity score (sim) of less than or equal to 0.4. Furthermore, we analyze the distribution of the 2000 incidents among the 25 identified subgroups. As depicted in Figure 4.8, our proposed solution demonstrates a balanced distribution without major outliers on the lower end, referring to subgroups with the smallest sizes. Conversely, the presence of numerous outliers in the distribution indicates the existence of numerous similar incidents that should be collectively explained rather than treating them independently, as done in the `local-wb` approach.

4.5 Discussion

In this chapter, we tackled one of the most demanding tasks in incident management, which is incident assignment during the triage phase. Our focus was on automatically routing incident reports to the appropriate teams using sophisticated deep learning algorithms known for their efficient analysis of textual data. However, we encountered a significant challenge related to interpretability. The issue arises when service teams seek explanations for why a specific incident has been assigned to them, and practitioners express concerns about using models whose decision-making mechanisms they cannot understand. This calls for a solution that can provide interpretable explanations for the model's decisions. While local models can offer accurate explanations while maintaining some fidelity to the black box models, they face limitations in terms of complexity due to the number of required local models.

Table 4.4: Mapping the contribution `SplitSD-4XAI` to our proposed taxonomy.

Context	Focus Area	Incident Triage: Assignement
	Maintenance Layer	N/A
	Scoop	ERP Software System
Data	Source and Type	Incident Reports ⟶ Tab data Topology ⟶ Tab data Standard Events ⟶ Tab data Performance Metrics ⟶ Tab data
	Feature Eng	Vectorization of textual data with `tf-idf`
Model	Approach	LSTM-Attention Model ⟶ Incident Assignement Subgroup Discovery (EMM) ⟶ Summarizing local b.b explanations
	Paradigm	Supervised
	Metrics	F1-Score ⟶ Incident Assignement Fidelity ⟶ Summarizing local b.b explanations
	Availability	Code (`https://github.com/RemilYoucef/split-sd4x`)
Particularities		Interpretability, In context-Evaluation, Scalabity
Contribution		First to apply SD in the context of XAI to contextualise local explanations

In our work presented in this chapter, we addressed this problem through the use of subgroup discovery and most likely exceptional model mining. We devised a framework that incorporates three key building blocks to effectively tackle our target problem. Instead of mapping a single incident prediction to a human-interpretable explanation, our framework maps a contextualized set of incidents described by an interpretable pattern to an interpretable subgroup model. Notably, our framework provides users with the flexibility to determine the number of subgroups and, consequently, the number of explanations they desire. We conducted experiments using incident reports, and the results showcased the effectiveness of our approach, which we refer to as `SplitSD-4XAI`. Our framework successfully provides a concise set of high-fidelity explanations for a black box model. Even with a small number of explanations, the results are comparable to individual explanations for each incident. For instance, we were able to explain 2,000 decisions using only 25 subgroups, without a significant loss in fidelity.

We believe that this work opens up new avenues for future research. One interesting direction is to expand our approach to compare the behaviors of different models and uncover what each model captures in various situations. This could involve using more sophisticated explainers such as SHAP [198] instead of relying solely on the LIME methodology [254]. Another improvement could be to incorporate a constraint based on the distribution of classes per subgroup. For example, one could prioritize grouping together incidents that are more likely to belong to a single class in order to analyze fewer explanations within the subgroup model.

Looking ahead to the next chapter, we will delve into another problem within the incident diagnosis phase, specifically focusing on the analysis of Java out-of-memory exceptions using the subgroup discovery approach. In this particular use case, the available datasets consist of suspect leaky classes with identified sizes, organized hierarchically. This unexplored aspect of subgroup discovery adds to the novelty of our research in the field. In Table 4.4, we provide the positioning of our contribution based on the taxonomy introduced in Section 2.4, demonstrating its alignment with the existing landscape.

Chapter 5

Mining Java Memory Heap Dumps using Subjective Interesting Subgroups with Hierarchical Targets Concepts

Java out-of-memory incidents are a common and challenging problem in software systems. Diagnosing these incidents is difficult due to the complexity of analyzing large histograms that contain information about suspect leaky classes occupying memory at the point of saturation. Additionally, the hierarchical relationship between packages and classes in these histograms adds to the complexity. It is also important to highlight that Java out-of-memory incidents are not always caused by memory leaks, but can also be the result of coding mistakes in the source code. To address this issue, we propose using Subgroup Discovery, which has been successfully employed in diagnosing performance issues in SQL query executions. However, applying Subgroup Discovery to this problem presents several challenges. The targets now involve real-valued attributes organized hierarchically, which is a novel problem in Subgroup Discovery that has not been explored before. This motivates us to introduce a novel and generic Subgroup Discovery setting with hierarchical target concepts. We define an adapted pattern syntax and a quality measure that identify relevant, non-redundant, and noise-resistant subgroups. Leveraging the developer's prior knowledge about similar historical problems and interesting patterns, we employ the framework of subjective interestingness. This framework incorporates prior knowledge about the data and emphasizes patterns that are surprising and informative when contrasted with these priors. We propose an approach that mines subsets of incidents to pinpoint, for each characterized subgroup, the potential root cause behind the memory issue. To demonstrate the effectiveness of our approach and the quality of the identified patterns, we provide an empirical study.

5.1 Introduction

The analysis of Java memory problems has consistently piqued the interest of both the software engineering and data mining communities. Their shared objective is to provide practitioners with efficient and effective tools for detecting, diagnosing, and mitigating these incidents. Traditional manual approaches often fall short in tackling these complex issues, especially when confronted with a high influx of incidents from diverse sources. This limitation becomes even more apparent in large-scale systems like ERP software-connected factories or cloud systems, where the sheer volume of incidents poses a significant challenge for manual analysis. One of these common incidents is related to `OutOfMemory` Errors, i.e., when the memory allocated to the software is saturated. This is often due to a memory leak [313, 140, 299, 272] caused by some specific bug. Engineers usually exploit tools that give statistics about the memory content at the moment of the error, which helps to find the root-cause. For instance, the `jmap` command depicts a histogram showing the memory consumed by each class in a Java Virtual Machine, as demonstrated in Figure 5.1. Moreover, these classes are *hierarchically* organized in packages. For example, the class `LinkedHashMap` in Line 9 is part of the package `java.util`, which is itself included in the package `java`. Using this histogram, the analyst may identify some classes that consume much more memory than expected, which may be the cause of the memory saturation.

Analyzing such histograms can be overwhelming and time-consuming for multiple reasons: **(i)** the analyst may not know what is the *normal* size consumption corresponding to each class. She needs to have either a significant experience or some *reference* histograms of healthy servers to compare with. In Figure 5.1, even if `[C` is the most consuming class, it does not mean that it is the cause of the incident as this value may be its usual size, **(ii)** some memory incidents are not only related to a single class but to a software feature which impacts several classes belonging to some packages. Among many hierarchy levels, how to concisely identify those suspicious packages and/or classes? **(iii)** the analyst often inspects a large dataset of histograms related to many incidents that happen in different servers and multiple situations. Each incident is described by its context (e.g., software version and type) as well as its `jmap` histogram describing memory content. In this case, she seeks to find contexts that significantly co-occur with specific kinds of memory errors, which would help to pinpoint the root cause for several incidents at once and also limits the efforts of diagnosing each incident separately. In fact, existing methods [65, 140, 299, 272] aim to diagnose memory problems separately, but not to analyze large sets of incidents simultaneously to discover common patterns. Also, they address particularly the problem of memory leak detection, whereas there are many other possible factors that cause `OutOfMemory` incidents.

Subgroup Discovery happens to be an effective and generic tool to address these challenges encountered in analyzing Java memory problems. It excels at extracting valuable information from datasets by grouping data within an interpretable context that exhibits abnormal behavior compared to the rest of the data. However, applying Subgroup Discovery to this specific problem context is not a trivial task. The distinctive aspect of this problem lies in the representation of target concepts as hierarchically organized sets of numerical values that are closely interconnected. Remarkably, existing Subgroup Discovery approaches have not previously considered such a case, thereby introducing a novel research area within the field. Furthermore, when investigating memory-related problems, developers often accumulate background knowledge through iterative analyses. Consequently, it becomes crucial to

```
> jmap -histo
  num   #instances      #bytes  class name
----------------------------------------------
   1:      121908    11267384  [C
   2:       19054     2895288  [I
   3:      118284     2838816  [Ljava.lang.Object
   4:       26595     2474152  [Ljava.lang.Class
   5:       20056     2236120  [B
   6:        9627     1618344  [J
   7:        1027     1599864  java.util.HashMap$Node
   8:       46906     1138200  [Ljava.util.HashMap$Node
   9:       12942      766720  java.util.LinkedHashMap
  10:       19168      528346  org.hibernate.metamodel.internal.BasicTypeImpl
```

Figure 5.1: Example of a `jmap` histogram.

incorporate subjective measures when identifying data subgroups. These measures should take developers' prior knowledge into account and facilitate the inclusion of newly obtained information that also needs to be concise and non-redundant. Hence, we introduce a novel and generic SD setting that elegantly addresses this case. We use Subjective Interestingness framework [80] to model the information dependency between hierarchical attributes, and to assess the informativeness of patterns. Its success resides in its ability to incorporate prior knowledge that the user may have about the data, which allows retrieving patterns that are surprising w.r.t. these priors. But also, it makes it possible to iteratively update the interestingness model to account for information already transmitted to the user during the mining task, and continuously bring informative subgroups. We characterize these subgroups with specific subsets of target attributes called antichains (a set of hierarchically incomparable elements). In fact, this antichain constraint allows us to avoid redundancy inside the same pattern, as hierarchically comparable attributes often transmit the same information. This work is built upon the approach proposed in [33] which seeks to extract contrastive attributes from only a single hierarchy. In other terms, this existing method is not able to mine a dataset of many hierarchies described by additional contextual attributes. The contribution we propose in this Chapter is exploited to analyze a dataset of `jmap` histograms and contextualize memory errors.

Roadmap. The next Section 5.2 introduces the background notations used to define our framework by presenting the raw data as well as an informal description of the studied problem. Section 5.3 formally defines the problem setting by introducing the employed pattern, the subjective interestingness, the iterative updating procedure of the background model, and the mining algorithm. In Section 5.4 we report an empirical study to evaluate the proposed approach. We conclude in Section 5.5 with a discussion of possible improvements and other future perspectives.

5.2 Background and Methodology

We conduct our analysis on a set of Java memory snapshots, a.k.a. `Java memory heap dumps`. Each generated memory snapshot denotes the instantiated objects that are stored in the heap of the Java virtual machine of a specific server at a particular moment. These snapshots can be generated using the `jmap` tool. Each snapshot is mapped with a hierarchy that recursively groups classes and/or sub-packages of the same package, along with the size of their instantiated objects. Furthermore, we incorporate additional descriptive features pertaining to memory snapshots, such as environment variables and performance metrics, resulting in many properties that allow contextualizing the Subgroup Discovery.

5.2.1 Raw Data

Java Memory Heap Dumps. Java memory analysis plays a crucial role in monitoring the performance of a Java application, allowing developers to effectively manage memory consumption and maintain application consistency. In a Java virtual machine (JVM), all application-created objects, such as strings, integers, and arrays, are stored in the heap. The heap memory utilizes dynamic allocation, as there is no fixed pattern for allocating and deallocating memory blocks. To efficiently manage heap memory, the JVM incorporates a garbage collector. This component collects unused Java objects from memory by releasing resources as long as they no longer have references. Essentially, the JVM automatically reclaims memory that is no longer in use. However, there are instances where the garbage collector is unable to free up memory due to certain objects still retaining references. This situation can lead to memory saturation. When the application attempts to create new objects but encounters insufficient remaining memory, it throws the `java.lang.OutOfMemoryError` exception. This exception can occur for three main reasons: (1) the currently allocated heap size is insufficient to accommodate the objects generated during runtime, (2) there is a coding error that retains references to unnecessary objects, causing a memory leak, (3) a large number of objects are loaded into memory simultaneously, overwhelming the available memory space.

To obtain specific memory-related statistics, we make use of the command-line utility `jmap` with the `-histo` option. This allows us to generate a class-wise histogram, providing information such as the number of instantiated objects per class in the heap and the total memory used by each of these objects. The histogram includes the fully qualified class names, as shown in Figure 5.1. For each memory snapshot, we utilize this histogram to construct a hierarchy that organizes classes and sub-packages recursively under their respective parent packages. This hierarchy helps us understand the relationships between different classes and packages. For instance, as demonstrated in the sub-hierarchy presented in Figure 5.2, the `java.lang` package contains the sub-package `java.lang.reflect` and the class `java.lang.String`, among others. As a result, the size of a package is determined by the combined size of its sub-packages and classes.

Additional features for contextualization. Each memory snapshot is incorporated with additional descriptive characteristics. Particularly, these features describe the execution environment and the software component in which the exception `OutOfMemoryError` is triggered (i.e., its topology). For instance, the Java virtual machine can be parameterized by the flag `Xmx` which specifies the maximum memory allocation pool for the `JVM`, the `Xms` to indicate the

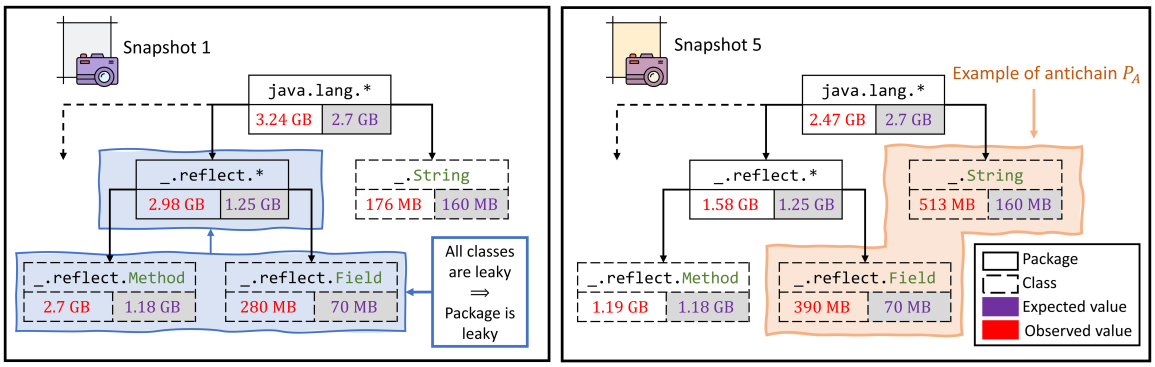


Figure 5.2: Example of a `JMAP` sub-hierarchy retrieved from two different memory snapshots with some class/-package sizes.

initial memory allocation pool, the size of swap memory. Information on when the memory collapsed is also provided (e.g., whether it is a working day), etc.

We create a dataset $\mathcal{D} = (\mathcal{O}, \mathcal{A}, \mathcal{H})$ that unifies the different data sources mentioned above. $\mathcal{O} = \{o_i\}_{1 \leqslant i \leqslant n}$ is a set of objects that correspond to memory snapshots indexed by the pairs `(server;timestamp)`. $\mathcal{A} = \{a_j\}_{1 \leqslant j \leqslant m}$ is the set of descriptive attributes used to contextualize the snapshots, and $\mathcal{H} = \{H_i\}_{1 \leqslant i \leqslant n}$ is a set of hierarchies constructed from `jmap` histograms. These hierarchies group classes and/or sub-packages that belong to the same package and include the size of their instantiated Java objects. Table 5.1 provides a dataset with 10 objects $\mathcal{O} = \{o_1, ..., o_{10}\}$ corresponding to 10 memory snapshots generated during `outOfMemoryError` exceptions. Each object is described by 4 attributes and referenced by a hierarchy $H \in \{H_1, ..., H_{10}\}$. Figure 5.2 illustrates a typical example of sub-hierarchical elements that have varying values of both H_1 and H_5, which highlight the integer-valued attributes associated with class sizes and their corresponding packages. For instance, the package `java.lang.reflect` in H_1 has a size of 2.98 GB and contains only two classes, namely, `java.lang.reflect.Method` (size of 2.7 GB) and `java.lang.reflect.Field` (size of 280 MB).

Table 5.1: Toy Example of a dataset $(\mathcal{O}, \mathcal{A}, \mathcal{H})$. Gray cells indicate that the size values are larger than what was expected.

$\mathcal{O}$	**Descriptive attributes $\mathcal{A}$**				**Size of instantiated objects w.r.t. packages in MB** represented with hierarchies $\mathcal{H}$ (see example for snapshots o_1 and o_5 in Fig. 5.1)				
	softType	softVersion	Xmx	weekDay	J.L.*	J.L.reflect.*	J.L.reflect.Field	J.L.reflect.Method	J.L.String
o_1	Sales	V_3	$4.2e+09$	True	3242	2980	280	2700	176
o_2	Sales	V_3	$2.3e+09$	False	3296	3003	322	2678	355
o_3	EDI	V_1	$6.4e+09$	True	2305	1474	264	1210	163
o_4	Factory	V_1	$1.8e+09$	False	2217	1481	386	1095	480
o_5	Factory	V_2	$2.4e+09$	True	2475	1582	390	1192	513
o_6	Manager	V_2	$5.3e+09$	True	2016	1258	56	1202	140
o_7	Sales	V_3	$2.4e+09$	True	3398	2814	320	2494	402
o_8	Factory	V_3	$8.2e+09$	False	2715	1326	84	1200	147
o_9	Sales	V_3	$6.4e+09$	True	2430	1577	412	1165	120
o_{10}	Sales	V_1	$4.5e+09$	True	2570	1283	68	1215	422

5.2.2 Hierarchical Target Concepts

We consider the scenario where the concepts of interest are defined as a set of positive integer-based attributes that are structured hierarchically. These hierarchically organized concepts are generally referred to as *counters* based on their observed values. These counters represent either the size of the classes (which are located at the leaves of the hierarchy) or packages (which are located at internal nodes). The root node represents the size of the heap at the time of the memory crash. We formally define a hierarchy $H \in \mathcal{H}$ as follows.

Definition 6 (Hierarchy)**.** A hierarchy $H_i \in \mathcal{H}$ is defined as a tuple $H_i = (E^{(i)}, \leqslant, \langle e_1, x_1^{(i)} \rangle)$ for $i \in [\![1, n]\!]$ where:

- $E^{(i)} = \{\langle e_1, x_1^{(i)} \rangle, ..., \langle e_k, x_k^{(i)} \rangle\}$ is a set of k items (nodes or concepts) with their counters. For convenience, We sometimes use $E = \{e_1, ..., e_k\}$ to refer to the set of items without their counters.
- $\leqslant$ is a partial order relation defined over this set E, indicating the relationship of predecessors between hierarchically linked concepts,
- $\forall e \in E : e_1 \leqslant e$ (the item e_1 is called the root of $\mathcal{H}$)
- there is only one path from the root e_1 to any other item:

$$\forall e_j, e_k, e_l \in E : e_j \leqslant e_l \wedge e_k \leqslant e_l \Longrightarrow e_j \leqslant e_k \vee e_k \leqslant e_j.$$

In Figure 5.2, the following relations hold: (`java.lang.*`) $\leqslant$ (`java.lang.reflect.*`) and (`java.lang.*`) $\leqslant$ (`java.lang.reflect.Field`). In other terms, if $e_j \leqslant e_k$, then e_j is a predecessor of e_k but not necessarily the direct parent of e_k. Moreover, we assume that the counter value x_l of a concept e_l is always larger or equal to the value of its successors. We introduce the following operations used throughout the remainder of this Chapter.

Definition 7 (Hierarchy operations)**.** Given $E' \subseteq E$:

- Precedessors operator $\Uparrow$, and successors operator $\Downarrow$ as:

$$\Uparrow E' = \{e \in E \mid \exists e' \in E' : e \leqslant e'\},$$
$$\Downarrow E' = \{e \in E \mid \exists e' \in E' : e' \leqslant e\},$$

- Strict predecessor relation: $e_j \prec e_k \Leftrightarrow e_j \leqslant e_k \wedge e_j \neq e_k$,
- The direct successor relation $\prec$ as: $e_j \prec e_k \Longleftrightarrow \Downarrow \{e_j\} \cap \Uparrow \{e_k\} = \{e_j, e_k\}$. Also, if $e_j \prec e_k$, we use the notation $\pi_k = j$ to refer to the index of the only direct parent of e_k ($e_j = e_{\pi_k}$),

The counter value of each concept e_l for an object $o_i \in \mathcal{O}$ is denoted using the discrete random variable $X_l^{(i)}$ defined in $\mathbb{N}$. If a particular value of a concept $e_l \in E$ is empirically observed for the object $o_i \in \mathcal{O}$, it will be denoted in the hierarchy H_i by $\hat{x}_l^{(i)}$. Obviously, we

have, $\forall i \in [\![1, n]\!], \forall l \in [\![2, k]\!] : X_l^{(i)} \leqslant X_{\pi_l}^{(i)}$ and hence, $\hat{x}_l^{(i)} \leqslant \hat{x}_{\pi_l}^{(i)}$ (e.g., $x^5_{java.lang.String} = 513 \leqslant x^5_{java.lang.*} = 2475$).

5.2.3 Contrastive Antichains as patterns

We first introduce the concept of contrastive antichains as interesting patterns based on a single hierarchy. These patterns are comprised of a subset of hierarchically disjoint concepts that are informative and non-redundant. In our case study, we aim to concisely inform developers about suspicious classes and packages. To evaluate the interestingness of a pattern, we rely on prior knowledge about counters, such as developers' rough estimates of the space occupied by classes in the heap. For example, in Figure 5.2, developers expect the size of the `java.lang.string` class to be around 160 MB based on the analysis conducted on healthy servers. Thus, discovering that this class takes up 513 MB in snapshot o_5 is surprising.

Providing the user with such concepts can be interesting. However, because one counter's information affects the expectation of other hierarchically related concepts, one intuition is to recursively aggregate the interesting concepts at the same level into a higher level of the hierarchy. This approach is shown in Figure 5.2, where, for example, instead of listing both the leaking classes `java.lang.reflect.Method` and `java.lang.reflect.Field`, we only provide their parent package `java.lang.reflect`. Another intuition is to only include non-comparable concepts in the pattern (i.e., no concept is a predecessor or successor of any other concept in the same pattern). We refer to this set of concepts by an *antichain*. Thus, an antichain denoted as P_A ensures that the information provided is not redundant. For each identified concept $e_l \in P_A$ for a specific object $o_i \in \mathcal{O}$, a contrastive antichain pattern informs the user about the value $\hat{x}_l^{(i)}$. Yet, users generally tend to memorize only the order of magnitude indication instead of precise values. Hence, we refer to the counters in patterns with the scale $\lfloor log_2(\hat{x}_l^{(i)}) \rfloor$.

Definition 8 (Contrastive Antichains)**.** Given a hierarchy $H_i = (E^{(i)}, \leqslant, \langle e_1, x_1^{(i)} \rangle)$ with pairs of concepts and their values $\langle e_l, x_l^{(i)} \rangle \in E^{(i)}$, a contrastive antichain pattern $P_A \subseteq E$ is a subset of concepts that form an antichain w.r.t. $\leqslant$, i.e., $\forall e_j, e_k \in P_A$: $e_j \leqslant e_k \implies e_j = e_k$, with the integers $\lfloor \log(\hat{x}_l^{(i)}) \rfloor$ describing the scale of the values of their counters.

5.2.4 Need to Characterize Subgroups with a Common Antichain

Developers often analyze a large set of objects (memory snapshots) at once. Providing contrastive antichains for each individual object can be overwhelming, and many objects may share hierarchical concepts that have similar properties. We need to simultaneously characterize a subset of objects that are together associated to a contrastive antichain pinpointing suspicious classes or packages. An interesting example in Table 5.1 is the subgroup $\{o_2, o_4, o_5, o_7\}$ containing memory snapshots from virtual machines with a `Xmx` flag value not exceeding $2.5e{+}09$. All these snapshots exhibit unexpectedly high memory consumption of classes forming the following antichain: {`java.lang.reflect.Field, java.lang.String`}. Subgroup Discovery can hence prove invaluable by exploring the set of all candidate hypotheses and using a quality function to subgroups along with their retrieved antichain and identify the best of

them. We exploit the subjective interestingness framework (SI). This framework makes it possible to iteratively incorporate the new information provided to the user when communicating a subgroup to her, to avoid communicating subgroups with redundant information. For instance, suppose the user is presented with the subgroup $\{o_1, o_2, o_7, o_9\}$ defined by Sales servers of Version 3 associated with an antichain consisting of only the package `java.lang.reflect`. Subsequently, the user is presented with another subgroup containing objects $\{o_2, o_4, o_5, o_7\}$, associated with the antichain {`java.lang.reflect.Field, java.lang.String`}. Although the second pattern is interesting, it becomes less surprising when the user is aware of the first pattern, because $\{o_2, o_7\}$ are already associated with the concept `java.lang.reflect`, which is a package that already includes `java.lang.reflect.field`. Hence, we should ignore this pattern and suggest a more restrictive one, such as the subgroup that covers only $\{o_4, o_5\}$ with the description (`Xmx` $< 2.5e + 09 \wedge$ softType = Factory).

5.3 Mining Interesting Subgroups with Hierarchical Targets

5.3.1 Pattern Language

We have chosen to incorporate both the descriptive pattern language and the target (or antichain) pattern language into a unified framework. This decision is based on the fact that our search space involves exploring both simultaneously. In other words, given a descriptive pattern obtained from the descriptive search space, our goal is to find the most optimal contrasting antichain in the target search space and evaluate their quality within a unified pattern. To formally define this unified pattern language, we introduce the pair $\mathcal{L} = (\mathcal{L}_S, \mathcal{L}_A)$, where $\mathcal{L}_S$ represents the subgroup pattern language defined over descriptive attributes $\mathcal{A}$, and $\mathcal{L}_A$ represents the antichain pattern language defined over concepts E from $\mathcal{H}$. A pattern $P \in \mathcal{L}$ is denoted as $P = (P_s, P_A)$, where $P_s \in \mathcal{L}_S$ is a constrained selector that identifies a subset of objects based on their descriptive attribute values (using the same notations as in Chapter 3), and $P_A \in \mathcal{L}_A$ is a retrieved antichain from E. The descriptive pattern language, $\mathcal{L}_S$, is defined as $\mathcal{L}_S = \times_{i=1}^{m} sel_i$, where sel_i represents the set of all possible selectors for an attribute a_i. On the other hand, the target or antichain pattern language, $\mathcal{L}_A$, is defined as the set of all possible antichains that can be derived from E, denoted as $\mathcal{L}_A = \{P_A \subseteq E \mid \forall e_l, e_k \in P_A\colon\ e_l \leqslant e_k \Longrightarrow e_l = e_k\}$.

5.3.2 Subjective Interestingness Measure

We design a subjective interestingness measure to assess the quality of each pattern $P = (P_s, P_A)$. This function measures its surprisingness when contrasted with some background distribution that represents user priors about the data. Therefore, we need to formally model the prior beliefs about each counter $\langle e_l, x_l^{(i)} \rangle \in E^{(i)}$. We will present these counters through the probability distributions $Pr(X_l^{(i)} = x_l^{(i)})$.

5.3.2.1 Background Distribution

For each concept in the hierarchy, we assume that we have a reference that derives either an approximation of its value or its proportion to other concepts that are hierarchically dependent on it. In our case, we compute `jmap` histograms on normally behaving servers to

derive expected values of the size (or the proportion of size to the parent) that each class and/or package can hold in the heap. In particular, we aim to represent expectations about the x_l value of each concept $e_l \in E$ and expectations conditioned on the parent except for the root, i.e., $x_l \mid x_{\pi_l}$. For example, the expected average size occupied by a `java.lang.String` class is $\sim$ 160MB. We formalize these constraints as follows:

- **Expectations**: initially, the expectation of each random variable $X_l^{(i)}$ is independent of o_i and given as $\bar{x}_l$:

$$\forall i \in [\![1, n]\!], \forall l \in [\![1, k]\!] : \sum_{x_l} \Pr(X_l^{(i)} = x_l) \cdot x_l = \bar{x}_l, \tag{5.1}$$

- **Conditional expectations**: the initial expectation of the ratio of the value of a concept over its parent value, *conditional on* that parent value, is equal to:

$$\forall i \in [\![1, n]\!], \forall l \in [\![2, k]\!] : \sum_{x_l} \Pr(X_l^{(i)} = x_l \mid X_{\pi_l}^{(i)} = x_{\pi_l}) \cdot \frac{x_l}{x_{\pi_l}} = \frac{\bar{x}_l}{\bar{x}_{\pi_l}}, \tag{5.2}$$

. It is proven that for $l > 1$, the second constraint is already sufficient as it necessarily implies the first one [33].

This means it suffices for the constraint (5.1) to consider only the root node, in additional to conditional expectation constraints in (5.2) for the other nodes. More formally, we have:

Property 2. If the two following sets of conditions hold:

1. **Expectations for only the root**:
   $\forall i \in [\![1, n]\!], \sum_{x_1} \Pr(X_1^{(i)} = x_1) \cdot x_1 = \bar{x}_1,$
2. **Conditional expectations for other nodes**:
   $\forall i \in [\![1, n]\!], \forall l \in [\![2, k]\!] : \sum_{x_l} \Pr(X_l^{(i)} = x_l | X_{\pi_l}^{(i)} = x_{\pi_l}) \cdot \frac{x_l}{x_{\pi_l}} = \frac{\bar{x}_l}{\bar{x}_{\pi_l}},$

It follows $\forall i \in [\![1, n]\!], \forall l > 1, \sum_{x_l} \Pr(X_l^{(i)} = x_l) \cdot x_l = \bar{x}_l$.

These constraints admit an infinity of solutions to define the probability distributions for all concepts. Hence, similar to [80], we use only the distributions that maximizes the entropy i.e., the distributions that do not introduce any further assumptions that reduce the entropy, but only and explicitly the specified constraints (i.e. the expectations). This results in two types of probability distribution; (1) an initially *geometric* distribution for the root nodes $X_1^{(i)}$ having an expectation equal to $\bar{x}_1$ [80]:

$$\Pr(X_1^{(i)} = x_1) = \left(1 - \frac{1}{1 + \bar{x}_1}\right)^{x_1} \cdot \frac{1}{1 + \bar{x}_1}$$

(2) *binomial* distributions for conditional random variables with and average $\frac{\bar{x}_l}{\bar{x}_{\pi_l}} \cdot x_{\pi_l}$ [119] :

$$\Pr(X_l^{(i)} = x_l \mid X_{\pi_l}^{(i)} = x_{\pi_l}) = \binom{x_{\pi_l}}{x_l} \cdot \left(\frac{\bar{x}_l}{\bar{x}_{\pi_l}}\right)^{x_l} \cdot \left(1 - \frac{\bar{x}_l}{\bar{x}_{\pi_l}}\right)^{x_{\pi_l} - x_l}$$

Given these two probability distributions, and as proven in [33], we can derive geometric probability distributions for each random variables $X_l^{(i)}$:

Property 3. The marginal distribution for each random variable $X_l^{(i)}$ is geometric, and it is given as:

$$\Pr(X_l^{(i)} = x_l) = \left(1 - \frac{1}{1+\bar{x}_l}\right)^{x_l} \cdot \frac{1}{1+\bar{x}_l}.$$

5.3.2.2 Interestingness of a Pattern

Now that a probability distribution has been defined to model the user's background knowledge, we need to evaluate the antichain P_A on each object of the subgroup $o \in ext(P_s)$ with respect to its probability distribution. This is necessary to efficiently determine the most interesting and surprising patterns that contradict the background beliefs or the previously discovered findings (i.e., after iteratively updating its primary knowledge). To assess the score of a given pattern $P = (P_s, P_A) \in \mathcal{L}$, we propose a new quality measure rooted in the framework of subjective interestingness SI [80]. $SI(P)$ is defined as the ratio between the information content of the pattern P and its description length: $SI(P) = \frac{IC(P)}{DL(P)}$.

Information Content (IC). Also known as self-information or surprisal [71] measures the amount of information communicated to the user. It is defined as the negative log probability under the background distribution: $IC(P) = -\log(\Pr(P))$. However, calculating the information content of such complex pattern can be challenging, especially when dealing with multiple probability distributions of the same concept associated with different objects in the subgroup. To address this, we aim to quantify the information gain provided by an effective aggregation of counters $\langle e_l, \hat{x}_l^{(i)} \rangle$ in P_A, by evaluating the objects of the subgroup under their respective probability distributions (which might have been updated in previous iterations). A straightforward solution would be to use the mean value of each concept in the subgroup. However, the mean is too sensitive to outliers and does not provide a complete overview of the subgroup values w.r.t. a concept, as no assumptions are made about the subgroup variance.

To overcome this issue, we came out with the idea of calculating the cumulative distribution function from the quantile q_α of order α that indicates the value below which a certain percentage of the subgroup values fall, for each concept. The hyperparameter $(0 < \alpha < 1)$ is selected based on the quality of the subgroups returned. For instance, if the quantile of order 0.25 is considered, then our solution informs the user that 75% of the subgroup values are greater than this quantile. To better understand this approach, let's consider the subgroup $s = \{o_1, o_2, o_7, o_9\}$ with corresponding values $\{2980, 3003, 2814, 1577\}$ for the package `java.lang.reflect`. Assuming that all objects have the same geometric probability distribution with a mean of 1250MB (in the first iteration), we can use the cumulative distribution function under the quantile of order 0.25 to calculate the probability $\Pr(X^{(i)}_{\texttt{java.lang.reflect.}} \geqslant 2814) = 0.08$, which is interesting. In other words, for the subgroup defined as (softType = Sales $\wedge$ softVersion = V_3), 75% of its objects have a size that is greater than or equal to 2814MB for the `java.lang.reflect.` package. This information is valuable and surprisings since it deviates from what we would expect based on its probability distribution.

As previously stated in 5.2.3, the information content is communicated to the user by transmitting the scales of the values, instead of the exact values. As a consequence, we define

the new random variables $Y_l^{(i)} = \lfloor \log_2(\hat{X}_l^{(i)}) \rfloor$. Concretely, the IC of a pattern $P \in \mathcal{L}$ is given as follows:

$$\begin{aligned} IC(P) = IC\left((P_s, P_A)\right) &= -\log\left(\prod_{o_i \in s} \prod_{e_l \in P_A} \Pr(Y_l^{(i)} \geqslant {q_\alpha}_l^s)\right), \\ &= -\sum_{o_i \in s} \sum_{e_l \in P_A} \log\left(\Pr(Y_l^{(i)} \geqslant {q_\alpha}_l^s)\right), \\ &= -\sum_{o_i \in s} \sum_{e_l \in P_A} \log\left((1-p_l)^{2^{{q_\alpha}_l^s}} - (1-p_l)^{2^{{q_\alpha}_l^s+1}}\right). \end{aligned}$$

where ${q_\alpha}_l^s$ is the quantile of order α of the values $\{y_l^{(i)}\}_{o_i \in s}$ for the concept $e_l \in P_A$

Description Length (DL). It measures the complexity involved in communicating a pattern P to a user. In our case, we propose to compute it based on both the subgroup pattern P_s and the antichain P_A. When communicating the antichain, items closer to the root e_1 are more likely to be familiar to the user and easier to interpret. For instance, it is simpler to communicate the package `java.lang` (2^{nd} level) than the class `java.lang.reflect.Method` (4^{th} level). On the other hand, a subgroup with only a few selectors is easier to interpret, as it helps to quickly pinpoint the root cause. In order to characterize as many objects as possible in a subgroup that is distinguished by an interesting antichain, we avoid linear penalization of the subgroup size. Hence, the DL is given as:

$$\begin{aligned} DL(P) &= DL_s(P_s) \cdot DL_A(P_A) \\ &= (\beta \cdot \log(|s|) + \gamma \cdot \|P_s\|) \cdot \left(\eta \cdot (\sum_{e_l \in P_A} 1 + \log(|\Uparrow \{e_l\}|))\right) \end{aligned}$$

with β, γ and η are hyperparameters to weight each part of the DL according to the user preferences.

5.3.3 Updating the Background Knowledge

When conveying a pattern to the user, her background knowledge model must be updated to take into consideration the new piece of information. The communicated pattern values are likely to become the new expected values, and therefore, the probability distributions must also be updated accordingly. In the following, let $\mathcal{R} \subseteq \mathcal{L}$ be the set of patterns that have already been observed up to the i^{th} iteration, that is, $\mathcal{R} = (P_s^1, P_A^1), ..., (P_s^{(i)}, P_A^{(i)})$. We refer to the quality of the pattern P assuming the user has knowledge of $\mathcal{R}$ as:

$$SI(P \mid \mathcal{R}) = \frac{IC(P \mid \mathcal{R})}{DL(P)} = \frac{-\log(\Pr(P \mid \mathcal{R}))}{DL(P)}$$

The probability $\Pr(P \mid \mathcal{R})$ represents the likelihood of pattern P appearing in the data given that the user is aware of the quantile of order α for the subgroup values of each concept for all previously communicated patterns $P' = (P'_s, P'_A) \in \mathcal{R}$. In other words, instead of

Algorithm 2: `SCA-Miner`

Input: the dataset: $\mathcal{D}_{crash} = (\mathcal{O}, \mathcal{A}, \mathcal{H})$, *width*: the number of most promising subgroups per level, *depth*: the maximum depth to explore in the lattice, *threshold*: a threshold on the number of patterns.

Output: $\mathcal{P}$: An ordered collection of patterns $P \in \mathcal{L}$ sorted based on iteratively updated SI.

$\mathcal{P} \leftarrow \langle\rangle$
$\mathcal{R} \leftarrow \varnothing$
repeat
  // Update the model with the sum-product algorithm
  // and derive the probabilities $\Pr(Y_l^{(i)} = \hat{y}_l^{(i)} \mid \mathcal{R})$:
  Sum-product($\mathcal{H}$, $\mathcal{R}$)
  // Get the best subgroup along with its associated contrastive antichain:
  $(P_s, P_A) \leftarrow$ `BeamSearch`($\mathcal{D}$, *width*, *depth*, $\mathcal{R}$)
  if $P_s \neq \varnothing$ *and* $P_A \neq \varnothing$ **then**
    $\mathcal{P}.append((P_s, P_A))$
    $\mathcal{R} \leftarrow \mathcal{R} \cup (P_s, P_A)$
until $P_s = \varnothing$ *or* $P_A = \varnothing$ *or* $|\mathcal{P}| = threshold$;

knowing the exact value of each object, the user only knows that its probability being higher than ${q_\alpha}_l^{s'}$ is $(1-\alpha)$. Thus, we update the probability distribution $\Pr(Y_l^{(i)} = y_l)$ as follows:

$$\Pr(Y_l^{(i)} = y_l) = \begin{cases} \Pr(Y_l^{(i)} = y_l) \cdot \frac{1-\alpha}{\Pr(Y_l^{(i)} \geqslant {q_\alpha}_l^{s'})}, & \text{if } Y_l^{(i)} \geqslant {q_\alpha}_l^{s'}. \\ \Pr(Y_l^{(i)} = y_l) \cdot \frac{\alpha}{1-\Pr(Y_l^{(i)} \geqslant {q_\alpha}_l^{s'})}, & \text{otherwise.} \end{cases}$$

For example, assuming that the user has been given the subgroup $s' = \{o_1, o_2, o_7, o_9\}$, whose antichain contains the package `java.lang.reflect` and that the first quartile (i.e., $\alpha = 0.25$) has been used to retrieve useful patterns, then the new probability distribution for each of the objects in s' must verify: $\Pr(Y^{(i)}_{java.lang.reflect} \geqslant \lfloor \log_2(2814) \rfloor) = 0.75$.

The hierarchical organization of the concepts implies that updating the probability distribution of a particular concept will have a direct and recursive impact on its predecessors and successors. For instance, if the user becomes certain (with a probability of 75%) that the size of the `java.lang.reflect.` package is larger than 2814, then it follows that the size of its parent, `java.lang.`, must also be larger than 2814 with a probability greater than 75%, since $X_{\pi_l} > X_l$. These dependencies between the random variables can be represented using a Bayesian tree, which is a graphical model. To propagate the impact of updating some random variables to all nodes in the hierarchy, we use the sum-product inference algorithm [39].

5.3.4 Mining Interesting Patterns

The process of finding the most interesting patterns $P \in \mathcal{L}$ described by an interpretable subgroup description that is associated with a contrastive antichain, is very costly, since the computational complexity of a subgroup discovery task is known to be prohibitive. Besides,

Algorithm 3: `BeamSearch`

Input: the dataset: $\mathcal{D}_{crash} = (\mathcal{O}, \mathcal{A}, \mathcal{H})$, *width*: the number of most promising subgroups per level, *depth*: the maximum depth to explore in the lattice, $\mathcal{R}$: the set of already provided patterns

Output: the best pattern $P^\star = (P_s^\star, P_A^\star)$.

```
beam ← {⟨∅, 0, {}⟩}
continue ← True
current_depth ← 0
while continue = True do
    last_beam ← beam.copy()
    for elt in last_beam do
        for Sel in ⋃_{j=1}^{m} sel_j do
            P_s ← elt[0] ∪ Sel
            (SI(P_s, P_A), P_A) ← GreedySearch(P_s, H, R)
            if SI(P_s, P_A) > worst_SI in beam then
                replace worst pattern in beam with {⟨P_s, SI(P_s, P_A), P_A⟩}
    current_depth ← current_depth + 1
    if last_beam = beam or current_depth > depth then
        continue ← False
return the best pattern P* in beam
```

each generated subgroup pattern $P_s \in \mathcal{L}_\int$ has to be evaluated with the set of all possible antichains L_A that tend to be at most $2^{k-1}+1$ in a rooted tree [151]. Moreover, the interestingness measure used in our framework SI is not monotonic (i.e., it is not straightforward to derive non-trivial bounds on its values to prune some uninteresting patterns), which implies that exhaustive search is not a feasible strategy to be adopted. We employ optimization procedures that are commonly used in both scenarios (i.e., enumeration of subgroup patterns and the search for contrastive antichains). We derive `SCA-Miner`, a heuristic approach that uses beam search to generate at each level of the lattice the most $k-$interesting subgroups with their associated antichain which is retrieved with a greedy search algorithm w.r.t. the subjective interestingness measure.

The proposed approach is outlined in Algorithm 2 (`SCA-MINER`), which is an iterative approach, that aims at each iteration to provide the user with an interesting pattern P. The algorithm starts by updating the model using the sum-product method to incorporate the previously acquired knowledge for the user $\mathcal{R}$. Then, based on the beam search strategy, which enumerates the subgroup patterns and uses a greedy search to derive the associated antichain that maximizes the SI, the algorithm expands its best patterns collection $\mathcal{P}$. `SCA-MINER` continues until the beam search yields an empty pattern or the maximum size of $\mathcal{P}$ is reached.

We use Beam search as highlighted in Algorithm 3 that systematically explores the conjunctions of selectors by expanding at each iteration a limited set of patterns that have the largest SI so far. Beam search evaluates the subgroup patterns on their set of hierarchies to extract the best associated contrastive antichain. In this phase, we use a greedy search method to greedily build a contrastive antichain for a specific subgroup pattern. This process is repeated on each level in the Beam search, where only the most promising patterns are

Algorithm 4: `GreedySearch`

Input: P_s: a subgroup pattern, $\mathcal{H}$: the set of hierarchies, $\mathcal{R}$ the set of already provided patterns.

Output: $(SI(P_s, P_A), P_A)$: The best greedily constructed antichain associated to its SI

$quality \leftarrow 0$
$P_A \leftarrow \{\}$
$continue \leftarrow$ True
$C \leftarrow E$
while $continue = True$ **do**
  $e^\star \leftarrow \arg\max_{e_l \in C} \frac{IC((P_s, P_A \cup \{e_l\})|\mathcal{R})}{DL_A(P_A \cup \{e_l\})}$
  if $\frac{IC((P_s, P_A \cup \{e^\star\})|\mathcal{R})}{DL_A(P_A \cup \{e^\star\})} \geqslant \frac{IC((P_s, P_A)|\mathcal{R})}{DL_A(P_A)}$ **then**
    $P_A^\star \leftarrow P_A \cup \{e^\star\}$
    $C \leftarrow \{e_l \in C \mid e_l \not\leqslant e^\star \wedge e^\star \not\leqslant e_l\}$
    $quality \leftarrow \frac{IC((P_s, P_A^\star\})|\mathcal{R})}{DL_A(P_A^\star\})}$
  else
    $continue \leftarrow$ False
  if $C = \{\}$ **then**
    $continue \leftarrow$ False
return $(\frac{quality}{DL_s(P_s)}, P_A^\star)$

maintained. The mining process stops when all possible selectors are explored or a chosen stopping criterion is met (i.e., the search depth). The best pattern found throughout the search is provided as an output.

To greedily build a contrastive antichain for a specific subgroup pattern, we use the algorithm detailed in 4. `GreedySearch` algorithm starts from the empty set of items $P_A = \{\}$, and a set of candidates $C = E$, and in each iteration tries to find the best item $e^\star$ that maximizes the ratio $\frac{IC(P_s, P_A)}{DL_A(P_A)}$. In this case, all the hierarchically related concepts to $e^\star$ (i.e., its successors and predecessors) are removed from C. When C becomes empty or when no possibility to improve the pattern quality remains, the best pattern $P_A^\star$ with its quality is returned. This process is repeated on each level in the Beam search, where only the most promising patterns are maintained. The mining process stops when all possible selectors are explored or a chosen stopping criterion is met (i.e., the search depth). The best pattern found throughout the search is provided as an output.

5.4 Experiments

We present in the following the experimental study we conducted to evaluate the performance and quality of results provided by `SCA-Miner` for analyzing Java memory heap dumps reported by our ERP software. We aim to assess whether the approach is capable of identifying interesting patterns based on Subjective Interestingness and whether the update of the background model is effective. Additionally, we review the descriptions of the identified patterns for both subgroups and antichains and assess their interpretability and relevance to our case study (i.e., memory crashes).

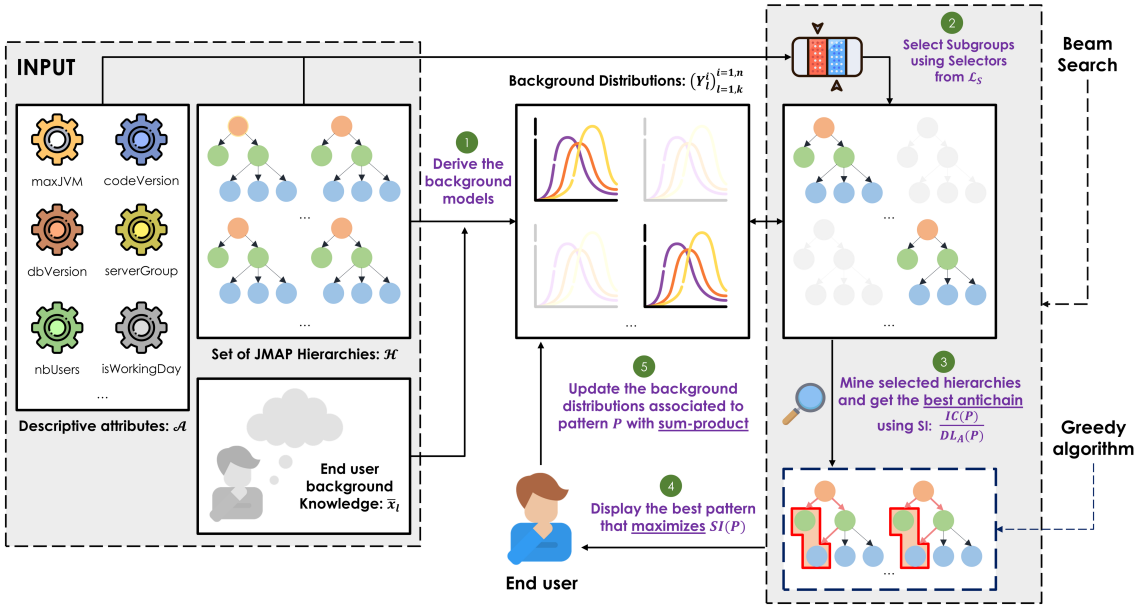


Figure 5.3: Overview of the subjective Subgroup Discovery framework with hierarchical target concepts.

Figure 5.3 is a recapitulation of the whole process of the proposed approach. The input dataset describes a set of incidents by `jmap` histograms, as well as other descriptive attributes contextualizing each incident. Moreover, the approach is provided by some priors about the data to be analyzed. These priors are the `jmap` histograms computed in healthy servers, which give the expected sizes of classes. The approach uses these priors to derive background distributions that represent healthy memory usage. Then, it mines the dataset of incidents to find the most informative subgroup, along with suspicious classes and packages. Once this subgroup is communicated to the analyst, the background model is then updated to incorporate this piece of information already known by her, and then identify the next most informative subgroup. This iterative process can be repeated as much as needed.

5.4.1 Experimental Setup and Methodology

Datasets and hyperparameters. Our experimental study involved analyzing more than 4,000 Java memory heap dumps collected over a 3-month period from approximately 350 servers. To establish reference values for the average heap space usage of each class or package, we generated a separate dataset of heap dumps from healthy servers at the beginning of each week. We only considered the top 200 classes in each histogram associated with a heap dump, as these are often the ones that retain the most heap space. The resulting dataset comprised 3,320 memory snapshots, each described by 14 descriptive attributes, and mapped into hierarchies to contextualize each subgroup and antichain. We used the readily available data mining tool `Pysubgroup` [169] to extend the beam search algorithm to fit our pattern language and measure of interest. We set the beam width to 50 and the number of selective selectors to 4, and displayed the top 20 patterns based on Subjective Interestingness (SI). We set the quantile of order 20 to inform users about 80% of the subgroup data values for a

specific concept. The hyperparameters associated with the DL function were set empirically as $\beta = 0.8, \gamma = 0.2$, and $\eta = 1$.

Baselines. While there are no approaches in the literature that specifically support hierarchical Subgroup Discovery with interesting target concepts, we consider some baselines in our study to highlight the benefits of `SCA-MINER` novel features, including the hierarchical structure and the new interestingness measure, as well as its capability to iteratively update the user background knowledge to avoid redundancy. First, we compare with the `SI` approach, which returns the best results according to our interestingness measure, but does not iteratively update the background model. Next, we compare with `Customized WRAcc (CWRAcc)`, which adapts the widely-used WRAcc measure [163] to our problem, measuring the deviation of the subgroup mean value from the mean value of the entire dataset regarding a target concept. Specifically, $CWRAcc(P_s, P_A, \theta) = \frac{1}{|P_A|^\theta} \sum_{e_l \in P_A} (\frac{1}{|s|} \sum_{o_i \in s} \hat{x}_l^{(i)} - \bar{x}_l)$. We run the Beam Search algorithm under the same hyperparameters as `SCA-MINER`. We also compare against the `KL` divergence, as performed in [286], to measure the difference between the observed probability distributions for the contextualized memory snapshots and the expected probability distribution, at each level of the hierarchy. Finally, we perform a post-processing step `PP` for each of these baselines to eliminate redundant patterns according to the Jaccard coefficient.

5.4.2 Comparative Evaluation

Figure 5.4 displays the comparative measures used to evaluate `SCA-MINER` against the baseline techniques discussed earlier. The SI measure produces larger contrast values in general (with or without update) compared to the CWRAcc and KL-Divergence measures. This is due to the fact that CWRAcc typically retrieves antichains containing nodes with higher counters, which do not necessarily result in contrastive patterns, as their average contrast is remarkably low when compared to that of the SI measure. Nonetheless, the contrast values of the non-updated SI tend to be higher than those of the updated SI because the algorithm often focuses on the same top patterns, generating a high level of redundancy. This redundancy is demonstrated in the bar graphs, where more than 60% of the patterns are redundant, which makes the process less surprising for the end user. Similarly, when using the CWRAcc and KL-Divergence measures, the resulting patterns often contain redundancy. Even after applying post-processing to the final pattern set, there is still a much higher level of redundancy when compared to using the updated SI measure (only about 4% redundancy).

5.4.3 Illustrative Results.

We present the top 4 patterns discovered by `SCA-MINER` from our dataset in Figure 5.5. Each pattern is accompanied by a description of its corresponding subgroup and the number of memory snapshots it covers. The red color in the figure represents the expected value of the concept in the antichain, while the green color indicates the average observed values of the subgroup with respect to this concept. We chose to report the mean value in the charts since it is more interpretable, intuitive, and comparable to the expected value. However, in Table 5.2, we also provide all the statistics related to the top 4 patterns, including the minimum and maximum value, and $0.2-$order quantile, which further reveals the properties of the subgroup distributions. Overall, our approach has successfully identified interesting

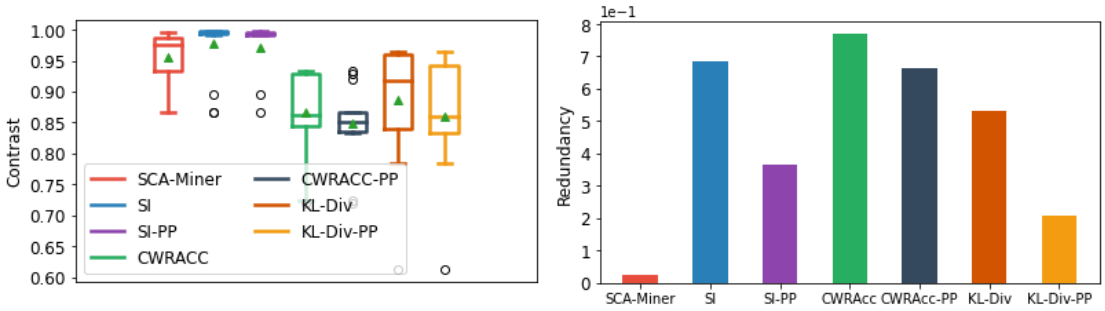


Figure 5.4: Comparison of contrast and redundancy between top patterns of `SCA-MINER` against the baselines.

Table 5.2: Statistics related to antichains that belong to the top retrieved patterns.

Top k	**Antichains**	$\bar{x}$	**Min**	$q_{0.2}$	**Avg**	**Max**
Top 1	`company.outil.persistance`	13	568	579	606	659
	`java.sql`	7	267	271	284	305
Top 2	`company.stock.gestion.` `modele.LotContQualVal`	3	1	344	458	749
Top3	`java.util.HashMap$KeySet`	2	1	104	104	106
Top 4	`company.core.services.droits`	21	104	482	541	730
	`company.outils.persistance.IK`	8	80	82	88	118

and surprising over-expressed patterns for all the extracted patterns. These patterns are diverse, non-redundant, and cover large subgroups (e.g., 85 and 183 memory snapshots in the second and third pattern). Additionally, our approach is capable of outputting generic packages as well as single classes when relevant.

The first discovered pattern discovered highlights that the server `PAM-p-03012` experienced a consistent increase in heap usage for two packages: `company.outils.persistance` and `java.sql`. The latter package includes several classes such as `sql.Timestamp` and `sql.BigDecimal`, indicating that the memory saturation is due to improper usage of the Direct SQL API in the source code. This API allows direct access to the database from the source code, bypassing the Hibernate (Object-Relational-Mapping) layer. Although it avoids loading Java objects mapped to the database in memory, it can cause memory saturation with SQL objects if not properly handled. The second component of the antichain is the package `company.outils.persistance`, which covers classes used to identify objects with primary keys, further strengthening the hypothesis of excessive object loading with the Direct SQL API. This hypothesis was confirmed by reviewing past maintenance resolution tickets, some of which occurred shortly after memory saturation. It's important to note that although this incident highlights a memory crash problem, it is not a memory leak, as it occurs rapidly due to an unhandled use case. This confirms that our method is not limited to specific issues and can diagnose any memory-related problem.

The antichain associated with the second pattern is noteworthy because it highlights the class `LotContQualVal`, which is highly contrastive for all Factory servers, with more than 97 weekly users. This pattern reveals that the frequent use of this particular class in Factory servers tends to cause memory saturation when the number of users exceeds a

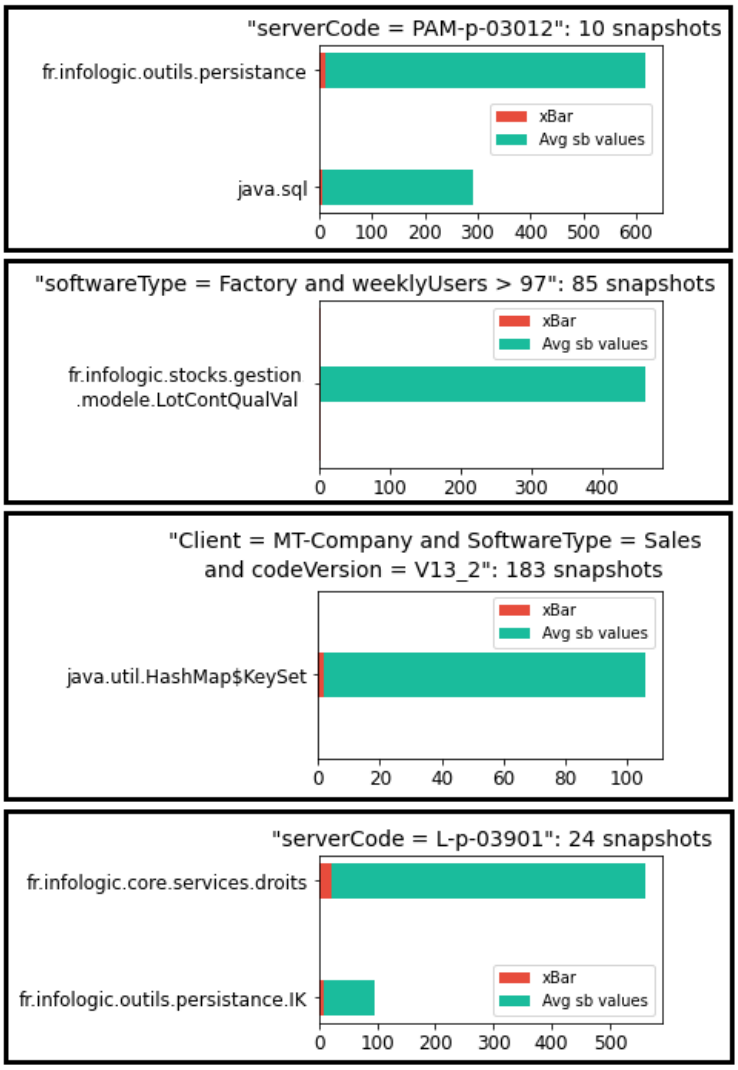
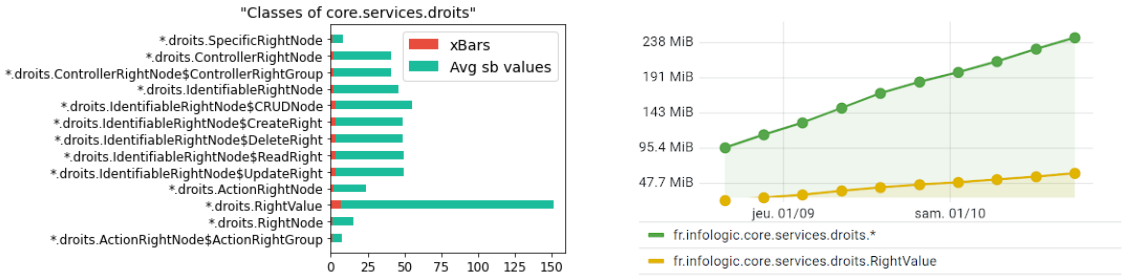


Figure 5.5: Top patterns returned by `SCA-MINER`

certain threshold, which is considerably high in the context of an ERP for industries. Despite having the best Information Content (IC), this pattern is ranked second because it has a lower generality and a relatively high depth compared to other patterns (i.e., $DL_A$is larger). On the other hand, Despite having a smaller depth, pattern 3 has been ranked lower than pattern 2 because it has more selectors (3), making it less interpretable than pattern 2. The Factory servers are crucial because they manage the core of the factory and our clients' production. The `LotContQualVal` class is instrumental in identifying functionality with an unusual problem, particularly those related to product quality features. This problem leads to a quick saturation event, and many maintenance tickets raised by our clients have identified this class as one of the crucial elements in root-cause analysis.

Pattern 4 highlights a prevalent issue in our ERP, which is memory leaks caused by the `RightValue` class. Therefore, in this pattern, we provide our experts with a more general solution that identifies memory leak problems across all classes in the `services.droits`



(a) Size of the classes belonging to `services.droits` package in pattern 4.

(b) Heap consumption of `services.droits` during 50 days in a leaky server.

Figure 5.6: Explanation of an interesting pattern found by `SCA-MINER`.

package, rather than just a specific class. This is shown in Figure 5.6a, where we compare the expected value with the subgroup average value for all relevant classes. Through further analysis with our experts, we discovered a growing memory leak that had gone unnoticed for several weeks (Figure 5.6b), which was not detected by other supervision tools. The antichain package helped pinpoint the source of the bug in the source code so that it could be fixed as a temporary solution. However, to prevent this problem from recurring, we established a systematic memory snapshot control to monitor the trend of the package size.

5.5 Discussion

In this chapter, we present our contribution, `SCA-MINER`, the first algorithm that enables the discovery of subgroups from data with labels/targets anchored in a hierarchy. This algorithm outputs the appropriate level of label abstraction in the hierarchy to the user, along with a descriptive pattern. It takes into account non-redundancy and interestingness using the subjective interestingness framework. We illustrate this contribution with an effective use case involving the analysis of a large batch of Java memory heap dumps. The goal is to identify potential issues that lead to memory crashes, including memory leaks. Our approach allows simultaneous analysis of multiple incidents and incorporates the user's background iteratively into the model. It identifies various problems related to Java memory crashes, not limited to memory leaks. Importantly, our model does not require access to the source code or any additional information beyond the memory heap dumps and contextualizing information. We start by presenting the formal definition and the necessary preliminaries for implementing our approach. Next, we introduce the `SCA-Miner` algorithm, which incorporates a new pattern syntax that combines descriptive patterns and target patterns. We also propose a new subjective interestingness measure to assess the interestingness of subgroups. This measure can vary iteratively based on the patterns transmitted to the user. Finally, using a real-life dataset provided by our ERP supervision team, we highlight several actionable patterns. Some of these patterns are directly integrated into our rule-based maintenance engine. Moving forward, our plan is to utilize the algorithm regularly to discover new interesting rules, bugs, and memory leaks. While runtime was not our primary concern, we recognize that there is room for improvement in the algorithm implementation by incorporating other heuristic approaches such as MCTS (Monte Carlo Tree Search) and genetic algorithms. Additionally,

Table 5.3: Mapping the contribution `SCA-Miner` to our proposed taxonomy.

Context	Focus Area	Root Cause Analysis
	Maintenance Layer	Application
	Scoop	ERP Software System
Data	Source and Type	Java Heap Dumps ⟶ Hierarchies Topology ⟶ Tab data
	Feature Eng	Parsing of Heap Dumps
Model	Approach	Subgroup Discovery with Subjective Interestingness Framework
	Paradigm	Supervised
	Metrics	Contrast and Redundancy (new metrics)
	Availability	Data and Code (`https://github.com/RemilYoucef/sca-miner`)
Particularities		Interpretability, In context-Evaluation, Human in the loop
Contribution		First to consider hierarchical target concepts in Subjective SD framework First to apply this framework to analyze Java memory crashes.

it would be beneficial to consider time-related pattern constraints for characterizing continuous increases. Furthermore, we believe that our approach can make a significant impact on mining patterns and descriptive rules in massively multi-labeled data when a hierarchy is available, (e.g., folksonomy and web data). Similarly to the previous two chapters, we provide the positioning of our contribution according to our taxonomy in Table 5.3.

In the upcoming chapter, we address another problem that cannot be addressed through subgroup discovery. Our goal is to summarize similar crash reports by grouping them into buckets based on shared similarities according to a given similarity measure. However, this problem focuses more on localizing similar incident reports based on a newly reported one, rather than contextualizing incidents that exhibit a specific property of interest. Specifically, the property of interest, in this case, can be a single bug induced by multiple bugs, which is currently not available to us as information. Therefore, our objective is to improve the search process, both in terms of efficiency and speed, to identify duplicate crash reports using any given similarity measure.

Chapter 6

Enhancing Near-Duplicate Crash Report Retrieval with Deep Locality Sensitive Hash Learning

Automatic near-duplicate crash report detection, also known as automatic crash bucketing, is a crucial phase in the software development process for efficiently triaging bug reports. A common methodology involves grouping similar reports using clustering techniques that rely on customized similarity measures aimed at comparing stack traces. However, with real-time streaming bug collection, there is a need for systems to quickly answer the question: What are the most similar bugs to a new one? This entails efficiently finding near-duplicates. Therefore, it is natural to consider nearest neighbors search to tackle this problem. Locality-sensitive hashing (LSH) is a well-known approach that can effectively address this problem and handle large datasets due to its sublinear performance and theoretical guarantees on similarity search accuracy. Surprisingly, LSH has not been considered in the crash bucketing literature. It is indeed nontrivial to derive hash functions that satisfy the so-called locality-sensitive property for the most advanced crash bucketing metrics. Consequently, in this chapter, we study how we leverage LSH for this task. To be able to consider the most relevant metrics used in the literature, we introduce `DeepLSH`, a Siamese DNN architecture with an original loss function that approximates the locality-sensitivity property of any given similarity measures, particularly those intended for comparing similarity measures and even for Jaccard and Cosine metrics, for which exact LSH solutions exist. Technically, we design an original loss function based on locality-sensitive property preservation while addressing the problem of optimizing non-smooth objective functions due to binarization. We support this claim with a series of experiments on an original dataset performed using state-of-the-art similarity measures proposed to handle the crash deduplication problem.

6.1 Introduction

Collecting and triaging runtime errors after a software release is a standard quality process (e.g., Windows Error Reporting [109], Mozilla Crash Reporter [222]). At Infologic, we receive a daily influx of over ten thousand of automatic and user-generated reports. Each report comprises a Java stack trace and relevant contextual information It is important to note that not all crash reports hold the same level of priority: Rare occurrences indicate unexpected shutdowns, more frequent ones involve GUI issues that obstruct end-users from completing their tasks, while the majority fall into the silent category. The latter typically represents bugs that either have workarounds found by users (thus limiting their impact) or background process failures that may take days to notice but have significant consequences. Hence, it is imperative to promptly address such bugs.

While extreme problems are rare and easy to identify and prioritize, it remains challenging to sort, rank and assign other reports to developers (or simply ignore them). Indeed, many different reports may actually imply a single root cause and a bug can produce slightly different stack traces known as *near-duplicates* [75]. Therefore, it is highly valuable to group similar crashes into buckets to accelerate the crash investigation process [87]. Reporting systems use a range of heuristics and manually developed rules to organize crash reports into categories, ideally each referring to the same bug [109]. However, in many cases, it may assign crash reports caused by the same bug to multiple buckets [109]. Thus, various alternatives address the *stack trace-based report deduplication problem* by designing custom and accurate similarity measures between stack traces, relying mostly on string and graph matching (e.g., edit distance, prefix match and LCSS) [87, 150, 48] and information retrieval (e.g., TF-IDF and N-grams) [171, 258]. Other metrics take into account specific characteristics of stack traces, such as the distance to the top frame or alignment between matched frames) [289, 75, 219]. In the aforementioned studies, similarity measures are generally embedded in clustering algorithms, but this comes with several drawbacks: First, it needs numerous similarity calculations when assigning a new stack trace to a cluster. Second, clusters are not stable over time and should be recalculated frequently without losing links to actual bug tickets that have been previously created. It can also be difficult to set its various parameters.

At Infologic, we aim to process reports in quasi-real time. An essential aspect of this objective is efficiently identifying the nearest neighbors for each crash report. We need to quickly determine if a corresponding ticket has already been created, allowing us to update its statistics, or if a new ticket needs to be generated. We must perform the same checks to ascertain whether similar issues have been encountered before and if any temporary or permanent solutions have been provided. However, the computational demands of searching for nearest neighbors are generally prohibitive. For instance, conducting linear scans, took approximately 10 hours to compare 1,000 stack traces against a pool of 100,000 stack traces using the similarity metric proposed by [48]. This has triggered the necessity to explore the concept of Approximate Nearest Neighbors Search (ANN) [200].

Hashing is a popular technique for ANN [200], and particularly LSH, allowing to search for approximate nearest neighbors in constant time. LSH satisfies the locality-sensitive property, that is, similar items are expected to have a higher probability to be mapped into the same hash code (or hash bucket) than dissimilar items [292]. Most importantly, LSH provides guarantees on the search accuracy that is, the probability for two stack traces having randomly the same hash function is need to be equal to their similarity and the collision probability

of being hashed into at least one bucket can be simply and fully controlled with two key parameters representing the user's desired search precision and recall. The more accurate LSH is, the larger its hash tables are. Fortunately, LSH is known for its computational and storage efficiency, as well as its sublinear search performance [130]. LSH remains surprisingly unexplored in the crash bucketing literature, despite its potential benefits. It is indeed not trivial to derive hash functions that guarantee the locality-sensitive property for the most advanced metrics of crash bucketing. Generating hash functions that meet this property is a non-trivial task. Although several LSH function families have been proposed, each for estimating one and only one conventional similarity/distance measure, e.g., Min-Hash function for Jaccard coefficient [47] and Sign-Random-Projection (Sim-Hash) function for angular distance [56], etc., a generalized procedure for applying LSH to various similarity measures remains ambiguous and theoretically complex. More specifically, there is currently no systematic procedure for deriving a family of LSH hash functions for any given similarity measure.

Exploring the application of LSH to custom similarities for crash deduplication is a novel area that we delve into. We propose to learn these hash functions in a supervised manner to mimic any given similarity measure while incorporating the locality-sensitive hashing component into the model learning process. We draw inspiration from the field of Learn to Hash techniques [189, 191, 176, 90, 158, 296], which effectively reduce the dimensionality of input data representations while preserving similarity. In fact, we aim to leverage the strengths of both LSH and Learn to Hash approaches. Our proposed model generates a hash code, with LSH guarantees, that is shared by the nearest neighbors of the input stack trace. This approach is the first similarity-agnostic method that utilizes hashing for the crash deduplication problem. To summarize, our contribution is three-fold:

- Aiming to overcome the problem of deriving LSH functions for stack trace similarity measures, we propose a generic approach dubbed `DeepLSH` that learns and provides a family of binary hash functions that perfectly approximate the locality-sensitive property to retrieve efficiently and rapidly near-duplicate stack traces.
- Technically, we design a deep Siamese neural network architecture to perform end-to-end hashing with an original objective loss function based on the locality-sensitive property preserving with appropriate regularization to cope with the binarization problem of optimizing non-smooth loss functions.
- We demonstrate through our experimental study the effectiveness and scalability of `DeepLSH` to yield near-duplicate crash reports under a dozen of similarity metrics.

Roadmap. The remainder of this chapter is structured as follows. In Section 6.2, we provide a comprehensive background that introduces the problem at hand and familiarizes readers with hashing for ANN, particularly focusing on the properties of LSH. Subsequently, in Section 6.3, we present the architecture of the `DeepLSH` model. We delve into the derivation of the objective loss function, which is based on the preservation of locality-sensitive properties. Furthermore, we address the challenge of optimizing non-smooth loss functions to directly obtain binary hash functions without compromising information integrity when relaxing vectors. Section 6.4 presents an overview of related work that explores hashing techniques in similar contexts, as well as an examination of stack trace-based similarity metrics that are

```
id: 16377610978254995717215-XXXXXXX-XX
sessionId: 2D7E2416131887D473F6CFD7B35769C
version: 13.7
@timestamp: 2022-12-26 11:13:40.657
typeError: ERROR
functionality: com.company.modules.factory.Factory
message: No CAB matches reading 'Invalid'
detail: class com.company.exceptions.MyException:
    at com.company.LancAdapter.do(LancAdapter.java:449)
    at com.company.CABWrapper.read(CABWrapper.java:191)
    ...
    at com.company.Main(Main.java:94)
user message : I got this error while I was trying to ...
```

Figure 6.1: An example of a crash report with a stack trace and its context. User message is only available for user-generated crash reports

subsequently utilized in the experimental studies. In Section 6.5, we present the results of our experimental study, where we assess the efficiency and scalability of `DeepLSH` using a large dataset of historical crash reports. We evaluate its performance under a variety of stack trace-based similarity metrics.

6.2 Background and Problem definition

Software often contains bugs that cause crashes and errors. In what follows, we use both terms interchangeably to denote application crash (the system is down) and both errors from background tasks or error pop-ups presented to the end-users. All of them are provided with a Java stack trace and a run-time context (software/OS/database version, timestamp, etc.) [264]. A stack trace is a detailed report of the executed methods associated with their packages during a crash. Stack traces can be retrieved through system calls in many programming languages. In Java, the stack trace lists methods in descending order: the most-inner call corresponds to the top of the stack trace. This is illustrated with a crash report from a software product in Figure 6.1. A stack trace dataset is a defined as set of N stack traces $\mathcal{D} = \{s_1, s_2, ..., s_N\}$.

6.2.1 Approximate Nearest Neighbors Search

A similarity measure between two stack traces is a function denoted as $sim : \mathcal{D} \times \mathcal{D} \longrightarrow [0, 1]$. It can be any conventional similarity metric (Jaccard coefficient) or a specialized stack-trace similarity measure (e.g., PDM [75], TraceSim [289], etc.). The distance function is naturally given by $dist : \mathcal{D} \times \mathcal{D} \longrightarrow [0, 1]$ where $dist = 1 - sim$. Given a dataset $\mathcal{D}$ of N stack traces, the problem of nearest neighbor search under a user-defined similarity measure sim consists in finding, for a specific stack trace $s \in \mathcal{D}$, another stack trace denoted as $nn(s) \in \mathcal{D}\backslash\{s\}$ such that: $nn(s) = \arg\max_{s' \in \mathcal{D}\backslash\{s\}} sim(s, s')$.

An alternative to nearest neighbor search is the fixed-radius nearest neighbor ($R-$near neighbor) problem, which seeks to find a set of stack traces $\mathcal{S}_R$ that are within the distance R of s ($0 < R < 1$), such that: $\mathcal{S}_R = \{s' \in \mathcal{D}\backslash\{s\} \mid dist(s, s') \leqslant R\}$. There exist simple tree-based algorithms for approximate nearest neighbor search problems, notably KD trees [35] and SR-tree [141]. However, for large-scale high-dimensional cases, these techniques suffer

from the well-known problem of *curse of dimensionality* [37] where the performance is often surpassed by a linear scan. Consequently, significant research efforts have been dedicated to exploring highly efficient and scalable methods for approximating nearest neighbor search problems in large-scale datasets, including Locality-Sensitive Hashing (LSH).

Locality-Sensitive Hashing (LSH) has been particularly proposed to tackle the problem of randomized or probabilistic approximate nearest neighbors search [292], that is, targeting the ANN problem with guarantees aiming to find approximate nearest neighbors with probability rather than a deterministic way (which is not tractable). This choice is driven by the purpose of ensuring guarantees on the search accuracy with respect to the exact nearest neighbor search while giving the user the ability to balance precision and recall to her desired level. Formally, we define our problem as follows:

Problem 3 (Randomized approximate nearest neighbors search (RANN))**.** Given a new reported stack trace s, a dataset $\mathcal{D}$ of historical stack traces, the goal is to report some of the $R-$nearest neighbors $\mathcal{R}$ of s such that: $\mathcal{R} = \{s' \in \mathcal{D} \backslash \{s\} \mid Pr[s' \in \mathcal{S}_R] \geqslant 1 - \delta\}$ with $(0 < \delta < 1)$. The lower the parameter δ, the lower the chance of finding elements in the radius (i.e., more restrictive)

6.2.2 Hashing approach for the RANN problem

Hashing-based approaches attempt to map data features from the input space into a lower-dimensional space using hash functions so that the approximate nearest neighbors search on the resulting hash vectors can be performed efficiently. The compact hash codes generally belong to the Hamming space i.e., binary codes. We define the hash function for a stack trace s as $y = h(s)$ where y is the hash code and $h : \mathcal{D} \longrightarrow \{0,1\}^b$ where $b \geqslant 1$ is the number of bits in the hash code. In approximate nearest neighbors search settings, we usually opt for multiple hash functions to compute the final meta-hash code: $Y = H(s)$, where $H(s) = [h_1(s), h_2(s), ..., h_K(s)]^T$ and K is the number of hash functions. Hashing-based nearest neighbors search includes hash table lookup strategy, [292] which seeks to design an efficient search scheme rooted in hash tables. The hash table is a data structure made of buckets, each of which is indexed by a meta-hash code such that the probability of collision of near-duplicates under a given similarity measure is maximized.

Given a stack trace s, the stack traces $\{s' \in \mathcal{D} \backslash \{s\} \mid H(s) = H(s')\}$ are retrieved as near-duplicates of s. In order to improve the recall, we generally construct L hash tables containing hash buckets, each corresponding to a hash code $\{H_1, H_2, ..., H_L\}$. The near-duplicate stack traces are then defined as $\{s' \in \mathcal{D} \backslash \{s\} \mid \exists j \in [\![1, L]\!], H_j(s) = H_j(s')\}$. To ensure good precision, the meta-code length of the hash functions needs to be increased.

6.2.3 LSH for RANN problem

To address the problem 3 of randomized nearest neighbors search, Locality-Sensitive Hashing (LSH) [130] maps high dimensional data to lower dimensional representations by using a family $\mathcal{H}$ of random hash functions that satisfy the locality-sensitive property. Thus, similar data items in the high-dimensional input space are expected to have more chances to be mapped to the same hash buckets than dissimilar items. These similar data items are said to collide. Starting with a formal definition of an LSH family $\mathcal{H}$ to address our problem, we

consider a metric space such that, $\mathcal{M} = (\mathcal{D}, dist)$, a threshold $0 < R < 1$, an approximation factor $c > 1$, and two probabilities p_1 and p_2. The hash family $\mathcal{H}$ is a set of M hash functions $\{h_1, h_2, ..., h_M\}$ where each $h \in \mathcal{H}$ is defined as $h : \mathcal{D} \longrightarrow \{0,1\}^b$. An LSH family must satisfy the following conditions for any two stack traces $s, s' \in \mathcal{D}$ and any random hash function $h \in \mathcal{H}$:

- if $dist(s, s') \leqslant R$, then $Pr[h(s) = h(s')] \geqslant p_1$,
- if $dist(s, s') \geqslant cR$, then $Pr[h(s) = h(s')] \leqslant p_2$.

A family $\mathcal{H}$ is said to be $(R, cR, p_1, p_2)-$sensitive if $p_1 > p_2$. Alternatively [56], a sufficient condition for $\mathcal{H}$ to be an LSH family is that the *collision probability* should be monotonically increasing with the similarity, i.e.,

$$Pr[h(s) = h(s')] = g(sim(s, s')), \tag{6.1}$$

where g is a monotonically increasing function. Indeed, most of popular known LSH families such as Minhash [47] for Jaccard similarity, satisfy this strong property.

The LSH scheme indexes all stack traces in hash tables and searches for near-duplicates via a hash table lookup strategy. The LSH algorithm uses two key hyperparameters L and K to be tuned. Given the LSH family $\mathcal{H}$, the LSH algorithm amplifies the gap between the high probability p_1 and the low probability p_2 by concatenating K hash functions chosen independently and uniformly at random from $\mathcal{H}$, to form a meta-hash function $H(s) = [h_1(s), h_2(s), ..., h_K(s)]^T$. The meta-hash function is associated with a bucket ID in a hash table. Intuitively, it reduces the chances of collision between similar stack traces, since this requires them to have the same value for each of the K hash functions (i.e., high precision over the recall). To improve the recall, L meta-hash functions $H_1, H_2, ..., H_L$ are sampled independently, each of which corresponds to a hash table. These meta-hash functions are used to map each stack trace into L hash codes, and L hash tables are constructed to index the corresponding buckets, each using K random hash functions. The LSH algorithm is conducted in two phases as illustrated in Figure 6.2 (considering, $L = 4$ hash tables, for each we have $K = 3$ hash functions of $b = 4$ bits.)

1. **Pre-processing phase:** The L hash tables are built from the N stack traces. Each hash table is indexed by K hash functions constituting its meta-hash function. Note that we store pointers to stack traces in hash tables, since storing them in the original format is memory intensive.

2. **Querying phase:** Given a stack trace s, the algorithm iterates over the L meta-hash functions in order to retrieve all stack traces that are hashed into the same bucket as s, then reports the union from all these buckets $\bigcup_{j=1}^{L}\{s' \in \mathcal{D} \mid H_j(s) = H_j(s')\}$. A $(L, K)-$parameterized LSH algorithm succeeds in finding candidate near-duplicates for a stack trace s with a sampling probability at least $1 - (1 - p^K)^L$, where p is the collision probability of LSH function. This means that $\delta = (1 - p^K)^L$ as defined in the problem 3 of randomized ANN. If property (6.1) holds, in particular for the identity function, i.e., $g(x) = I_x$, we can rely on the so-called *probability-similarity* relation between two different stack traces $s, s' \in \mathcal{D}$ such that:

$$P_{K,L}(s, s') = 1 - (1 - sim(s, s')^K)^L \tag{6.2}$$

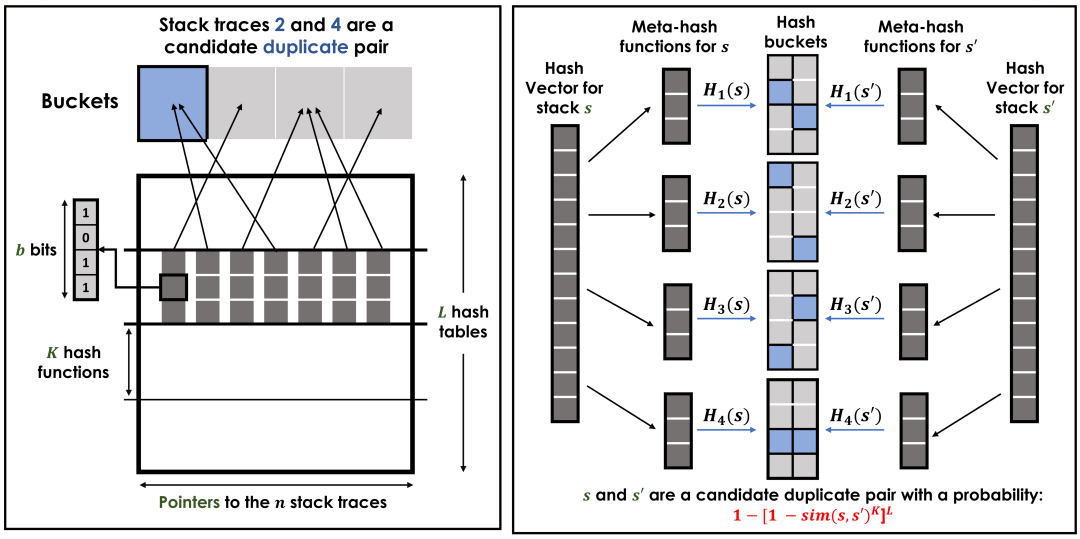


Figure 6.2: $(L, K)-$ parameterized LSH algorithm for retrieving near-duplicate stack traces with guarantees.

6.3 DeepLSH Design Methodology

To address Problem 3 in the context of crash deduplication in order to retrieve efficiently and with approximate guarantees the near-duplicates, we need to define appropriate LSH families for a given stack trace-based similarity measure. We recognize many suitable LSH families that have been already proposed for several similarity measures [77, 56, 47]. However, to the best of our knowledge, no generic mechanism exists to generate a family of hash functions that satisfies the locality-sensitive property for any user-defined similarity measure. Particularly, this presents a major limitation towards using LSH for crash deduplication problem, since the most efficient measures are non-linear functions and often rely on human expertise. Therefore, we propose `DeepLSH`, a generic approach that takes as input only the stack trace dataset and any user-defined similarity measure (measure-agnostic), to provide a family of hash functions that converges to the locality-sensitive property.

6.3.1 Learning a family of LSH functions

We exploit a deep supervised Siamese neural network with an original objective loss function to learn hash functions that converge to the locality-sensitive property for a given similarity measure. Figure 6.3 shows the structure of the proposed model that combines two identical neural networks sharing the same structure and the same parameters Θ. As input, we provide the model with the set $\mathcal{G}$ of all possible distinct pairs of stack traces. The model output is provided with the similarity values for each pair of stack traces. The model F, with its corresponding parameters Θ, consists in encoding a stack trace into a compact vector that represents a family of M concatenated binary hash codes, each of which is encoded in b bits in the Hamming space, denoted as $F_\Theta(s)$. The model consists of a concatenation of stacked convolution layers with different kernel sizes. We depict three kernel region sizes: $2, 3$, and 4, each of which has 256, 512, or 1024 filters. These filters perform convolutions on the one-hot encoded stack frames to generate feature maps. Then, 1-Max pooling is performed

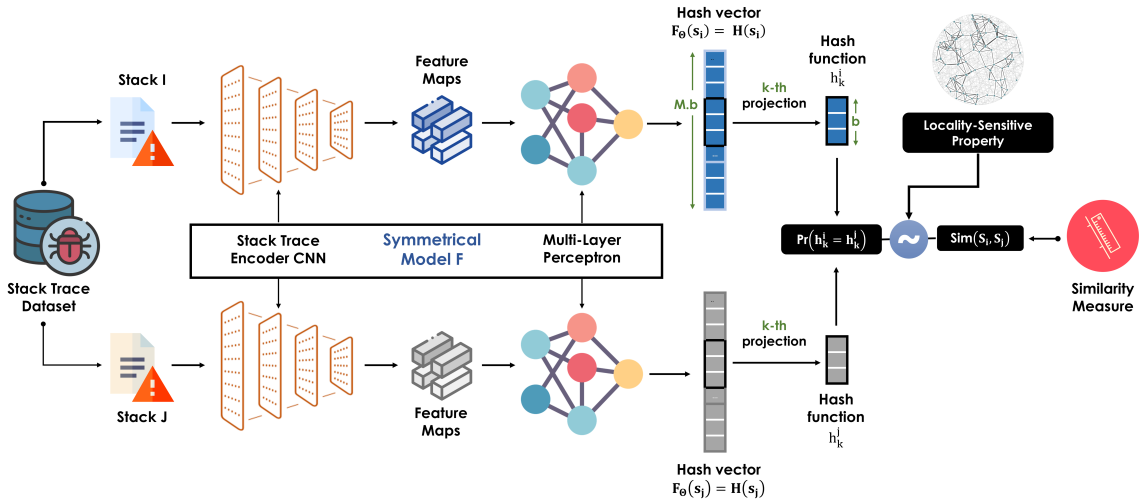


Figure 6.3: `DeepLSH`: Deep Siamese hash learning neural network overview.

over each map to record the largest number from each feature. Finally, the resulting features are concatenated to form a feature encoding vector for the penultimate layer that is fully connected to the hash model. It is noteworthy that our deep hash approach is model agnostic and, therefore, any feature encoder structure (e.g., CNN, CNN-LSTM, AE, etc.) can be also used as a stack trace encoder instead of our proposed network architecture.

Given the two hash vectors of the Siamese neural network, our contribution consists in designing an objective loss function that efficiently conducts F_Θ to learn a family of binary hash functions that aim to converge to the locality-sensitive property for the given similarity function sim. We propose to leverage Property (6.1) which is sufficient to imply the two required conditions of an LSH family. Assuming that the function g is the identity function: $g(x) = I_x$ since the similarity values are within the closed interval $[0, 1]$, the set of parameters Θ are optimized such that the probability of two random projected hash functions of order k from the resulting hash vectors, h^i_k and h^j_k being equal, converges to the similarity value between the two stack traces s_i and s_j i.e.,

$$Pr[h^i_k = h^j_k] = Pr[H(s_i)_k = H(s_j)_k] = sim(s_i, s_j) \tag{6.3}$$

More formally, we seek to minimize the Mean Squared Error (MSE) between the probability of collision of two randomly projected hash functions of order k, i.e. h^i_k resp. h^j_k of $H(s_i)$ resp. $H(s_j)$, and the similarity value $sim(s_i, s_j)$, that is:

$$\arg\min_{\Theta} \sum_{(s_i,s_j)\in\mathcal{G}} \frac{1}{|\mathcal{G}|}[Pr[F_\Theta(s_i)_k = F_\Theta(s_j)_k] - sim(s_i, s_j)]^2 \tag{6.4}$$

At this point, the challenge is to formalize the probability of collision in the loss function. In other words, we attempt to quantify the probability $Pr[F_\Theta(s_i)_k = F_\Theta(s_j)_k]$ during the learning phase. In what follows, we present the complete procedure for designing a computed loss function for the model F.

6.3.2 Objective loss function

Given that the hash vectors are in a Hamming space, i.e., the vectors are restricted to binary values, $\{0, 1\}$ or $\{-1, 1\}$, it can be demonstrated that calculating the collision probability between two randomly projected hash functions of the same order h_k^i and h_k^j is equivalent to computing the Hamming similarity between the two hash vectors $H(s_i)$ and $H(s_j)$. This equivalence is satisfied since, for the Hamming similarity when $b = 1$, it has been proven in [130], that the projection function (i.e., a single bit drawn randomly) verifies the locality-sensitive property. In other terms, for two binary vectors x and x' of length d with a Hamming distance r, the collision probability by randomly pulling a hash function from the set $\{h : [-1, 1]^d \rightarrow \{-1, 1\} \mid h(x) = x_i, i \in \{1, ..., d\}\}$ verifies:

$$Pr[h(x) = h(x')] = g(r) = 1 - \frac{r}{d} \tag{6.5}$$

Intending to leverage property (6.5) and given that our hash functions are rather b-bit encoded ($b \geqslant 1$), i.e., not restricted to a single projection but a succession of b coordinates, we need to generalize this property for $b \geqslant 1$. Consequently, we define a generalized Hamming distance between two hash vectors $H(s_i)$ and $H(s_j)$ as the number of different projected hash functions of order k: $|\{k \in [\![1, M]\!] \mid h_k^i \neq h_k^j\}|$. As a result, each hash function that belongs to $\{h' : [-1, 1]^{M \times b} \rightarrow \{-1, 1\}^b \mid h'(s) = H(s)_k\}$ satisfies the locality-sensitive property. This leads to conclude that for two different stack traces s_i and s_j, the collision probability between two projected hash functions of a specific order k' referring to (6.5):

$$Pr[h_{k'}^i = h_{k'}^j] = 1 - \frac{|\{k \in [\![1, M]\!] \mid h_k^i \neq h_k^j\}|}{M} \tag{6.6}$$

Correspondingly, referring to the property (6.6), the objective function as described in (6.4) can be formalized as follows:

$$\sum_{(s_i, s_j) \in \mathcal{G}} \frac{1}{|\mathcal{G}|} [1 - \frac{|\{k \mid h_k^i \neq h_k^j, k \in [\![1, M]\!]\}|}{M} - sim(s_i, s_j)]^2 \tag{6.7}$$

A challenging problem in hashing, on the other hand, consists in dealing with the binary constraint on hash vectors. This binary constraint leads to NP-hard mixed integer optimization problem [90]. In particular, the challenge in neural network parameter optimization is the *vanishing gradient descent* from the `Sign` function used to obtain binary values. Specifically, the gradient of the `Sign` function is zero for all non-zero input values, which is limiting for neural networks that rely on gradient descent for training. In order to handle this challenge, most deep hashing techniques relax the constraint during the learning of hash functions using `Sigmoid` or `Hyperbolic Tangent` functions [191, 121, 158, 127]. With this relaxation, the continuous hash codes are learned first. Then, the codes are binarized with thresholding. Continuous relaxation is a simple approach to address the original binary constraint problem. However, with binary hash codes that result from thresholding in the test phase, the solution may be suboptimal, compared to including the binary constraint in the learning phase.

To this extent, we propose a simple yet efficient solution to cope with the binary constraint

in the training phase. The solution lies in using approximate Hamming similarity. It requires having continuous values that are extremely close to binary values $\{-1, 1\}$. We propose to use the `Hyperbolic Tangent` activation on the hash layer while including the following condition in the loss function to drive the absolute hash values to be exceedingly close to 1:

$$\frac{1}{M \cdot b} H(s)^T . H(s) - 1 = 0. \tag{6.8}$$

Under this regularization term incorporated into the loss function, we define the approximate generalized Hamming similarity as follows:

$$gHam\left(H(s_i), H(s_j)\right) = 1 - \frac{\sum_{k=1}^{M} D_{\text{Chebyshev}}(h_k^i, h_k^j)}{2 \cdot M}, \tag{6.9}$$

where $D_{\text{Chebyshev}} = \max_{l \in \{1,...,b\}}(|h_{k,l}^i - h_{k,l}^j|)$

The Chebyshev distance between $h_{k,l}^i$ and $h_{k,l}^j$ is then given as the maximum absolute distance in one of the b dimensions. This implies that two hash codes are assumed to be similar if all bits of the hash code are matched for a specific projection. In other words, if $\exists$ $l \in \{1, ..., b\}$ for a specific k such that $|h_{k,l}^i - h_{k,l}^j| \approx 2$, then h_k^i and h_k^j are considered as two different hash codes.

Finally, to ensure independence between the hash code bits along with the load-balanced locality-sensitive hashing, and inspired by the work of [90], we have introduced the following regularization term that pushes the model to diversify the hash codes:

$$\frac{1}{M \cdot b} H(s)^T . \mathbb{1}_{M \cdot b} = 0. \tag{6.10}$$

Putting all together. Having all the necessary elements to design an appropriate objective loss function to be optimized for `DeepLSH` model, we define for convenience the following notations. Let $S = \{sim(s_i, s_j)\}_{i,j \in [\![1,N]\!]} \in [0, 1]^{N \times N}$ be the matrix representation of the similarities between all the stack trace pairs, and $\mathcal{H} = [H(s_1), H(s_2), ..., H(s_N)] \in [-1, 1]^{M \cdot b \times N}$ be the approximate binary hash vectors generated by the model F_Θ, such that, $H(s_i) = [h_1^i, h_2^i, ..., h_M^i]^T \in [-1, 1]^{M \cdot b}$. We refer to $\mathcal{W} = \{gHam(H(s_i), H(s_j))\}_{i,j \in [\![1,N]\!]} \in [0, 1]^{N \times N}$ as the matrix representation of the generalized Hamming similarity between all pairs of hash vectors produced from the model F_Θ. We formulate the following optimization problem to learn the parameters of our `DeepLSH` model using gradient descent:

$$\begin{aligned} \min_{\Theta} \mathcal{L}_{\texttt{DeepLSH}} = & \frac{1}{|\mathcal{G}|} \|\mathcal{W} - S\|^2 \\ & + \frac{\lambda_1}{2} \|\frac{1}{M \cdot b} \mathcal{H}^T \mathcal{H} - \mathbf{I}_N\|^2 \\ & + \frac{\lambda_2}{|\mathcal{G}|} \|\frac{1}{M \cdot b} \mathcal{H}^T \mathbb{1}_{M \cdot b}\|^2 \\ & + \lambda_3 \|\Theta\|_F^2, \end{aligned} \tag{6.11}$$

λ_1, λ_2, and λ_3 are regularization parameters to assess the importance of the different parts of the objective function.

6.4 Related Work

In order to provide a comprehensive understanding of the positioning of our contribution relative to existing approaches, as well as to introduce the baselines and metrics utilized in the experimental study, we discuss the two research areas covered by our work: (1) the approximate nearest neighbor search through hashing techniques, and (2) custom similarity measures for the stack trace deduplication problem.

Hashing for ANN search. Locality-sensitive hashing has been widely studied by the theoretical computer science community. Its main aspect focuses on the generation of a family of random hash functions that meet the locality-sensitive property for conventional similarity measures [259, 77, 56, 130, 47]. Particularly, Min-Hash (or min-wise independent permutations) [47] is an LSH function designed specifically for Jaccard similarity. Sim-Hash [56, 259] is another popular technique whose aim is to estimate angular similarities such as Cosine. Sim-Hash has been adopted by Google [259] and it is often used in text processing applications to compare between documents using their representations as a set of features (e.g., bag-of-words or n-grams). Both techniques cannot, however, be applied to estimate other similarity metrics besides Jaccard or Cosine. Designing LSH functions for any given similarity metric remains ambiguous and theoretically challenging, as there is no established method for deriving a set of LSH hash functions for a specific similarity measure. On the other hand, the concept of learn to hash has become the focus of many learning-based hashing methods, especially for the computer vision community [189, 191, 127, 121, 158, 296, 186, 331]. These methods are generally intended for the search of image similarity and rely on the semantic label information instead of continuous values. Moreover, these techniques are highly effective in reducing the dimensionality of the input data representations while preserving their similarities. However, they do not meet our main objective, which consists in an end-to-end procedure to retrieve near-duplicate data objects with guarantees. They do not reveal a systematic way to construct hash tables from the resulting hash codes, and neither do they control the trade-off between recall and precision using key parameters, as does LSH. Alternatively, we take benefit from both worlds, i.e., LSH and Learn to Hash, by proposing an end-to-end procedure that incorporates the LSH component in a learning process. We have identified a similar baseline approach in [127] that proposes a different methodology compared to ours, consisting of a deep hash coding neural network combined with Hamming LSH fitted on the resulting compact vectors to retrieve near-duplicate images in a large database. The authors first proposed a constrained loss function without incorporating the locality-sensitive property, and then performs discretization on continuous hash vectors to carry out the Hamming LSH separately from the model. We show through our experiments in Section 6.5 by adapting this two-step approach to stack traces, that it leads to a considerable search performance degradation compared to `DeepLSH`.

Stack trace similarities. We report research studies that tackle the crash report deduplication problem using stack trace similarity functions. [171] employed the TF-IDF-based scoring function from Lucene library [196]. [258] proposed DURFEX system which uses the package name of the subroutines and then segment the resulting stack traces into N-grams to

compare them using the Cosine similarity on their vector representations. Some alternative techniques have been proposed to compute the similarity using derivatives of the Needleman-Wunsch algorithm [228]. In [48], the authors suggested adjusting the similarity based on the frequency and the position of the matched subroutines. [75] proposed a new similarity measure called PDM in their framework Rebucket. This method computes the similarity based on the offset distance between the matched frames and the distance to the top frame. More recently, TraceSim [289] has been proposed to take into consideration both the frame position and its global inverse frequency. [219] present an approach that combines TF-IDF coefficient with PDM. Finally, we outline some earlier approaches that used edit distance as it is equivalent to optimal global alignment [26, 216]. It is noteworthy that our approach `DeepLSH` does not propose a new similarity measure and does not question the effectiveness or compete against these existing measures, but it is complementary to them. We demonstrate in what follows that `DeepLSH` model is able to estimate all these measures with the purpose of providing a scalable way to yield approximate near-duplicate stack traces w.r.t these custom similarity functions.

6.5 Experiments

We report our experimental study to assess the effectiveness of our approach `DeepLSH` in performing efficient, fast, and scalable approximate nearest neighbors search, by providing appropriate hash functions that can approximate the stack trace similarity measures.

6.5.1 Experimental Setup and Baselines

Stack trace dataset and training methodology. Our experiments are conducted on a comprehensive real-world dataset comprising stack traces automatically reported by our ERP software. To establish a robust training dataset, we selectively choose the most frequent stack traces from our historical incident database, creating distinct pairs of stack traces. These pairs are then utilized to train our `DeepLSH` model. Each pair is assigned a similarity value calculated using diverse similarity functions. We evaluate the performance of the `DeepLSH` model using twelve different similarity measures: Jaccard (Bag of Words and bi-grams), Cosine (Bag of Words, bi-grams, and TF-IDF), Edit distance [26], PDM [75], Brodie [48], DURFEX [258], Lerch [171], Moroo [219], and TraceSim [289].

It is important to note that these similarity metrics serve as a reference point and act as the ground truth in our current setup. Our objective, therefore, is not to evaluate the effectiveness of these similarity measures or compare them with each other. This is because each measure is applied to a vast dataset of stack traces, where labeled information is not always available, or manual labeling is impractical, especially in cases of frequent background process failures that may occur in the thousands per day. Therefore, our goal is to ensure that regardless of the used similarity measure employed for stack traces, our `DeepLSH` approach can effectively replicate this measure of similarity. Additionally, it should be capable of scaling up and being utilized within large-scale systems.

Regarding the training methodology, the training set consists of 499500 pairs of stack traces, while the validation and test set are constituted of 99900 pairs. The number of hash functions M and the size of each hash code b can be parameterized by the user. By default, these values are respectively set to 64 hash functions of 8 bits. The max iteration is fixed at

20 epochs with a batch size of 256 or 512. The parameter optimization process is achieved with the readily available `Adam` optimizer of `TensorFlow` with an adaptive learning rate and a weight decay of $1e^{-4}$.

Baselines. As there is no explicit competing approach for `DeepLSH` in the state-of-the-art, we chose to initially compare with **(1) Standard LSH methods**, namely Min-Hash and Sim-Hash. This comparison should only be performed with the Jaccard and Cosine metrics respectively, since they are not generalizable to other measures compared to `DeepLSH`. As mentioned in Section 6.4, we compare against the closest method to our work, referred to as **(2)** (`CNNH+LSH`) [127]. This approach uses the concept of Learn to Hash and then performs in a post-processing step the Hamming LSH. The methodology followed in this work is significantly different to ours. `DeepLSH` unlike the latter incorporates the LSH component into the model learning phase, resulting in a new loss function with related regularization to meet the locality-sensitive property. Regarding the application of [127] on stack traces, since it was primarily designed for images, we only needed to provide a list of one-hot encoded stack-frames to the convolutional feature extractor instead of pixels. Finally, to evaluate the scalability of our approach, we compare against **(3) Native k-NN (k-Nearest neighbors)** approach of linear complexity, using the exact computation of similarity functions between stack traces. It is noteworthy that clustering techniques have not been considered as baselines (as discussed in the introduction), since the addressed problem is an ANN search.

Evaluation protocol. Through this experimental study, we address the following research questions by proposing an evaluation protocol to assess each point claimed in this work:

- ✘ **RQ1** [**Model Evaluation**]: Does `DeepLSH` model manage to converge to the locality-sensitive property to mimic a diverse set of stack-trace-based similarity metrics? We first highlight, by means of the Kendall τ ranking coefficient [145], whether the model succeeds in preserving the original order between the predicted pairwise similarities. In addition, we study how accurately the generalized Hamming similarity approximates the true Hamming similarity between the discretized hash vectors in the test phase.

- ✘ **RQ2** [**`DeepLSH` for ANN search**]: Does `DeepLSH` model achieve satisfactory performance in finding near-duplicate crash reports user a given similarity measure? By querying the model to obtain the hash vectors for unseen stack traces, we study the search performance to retrieve approximately near-duplicate stack traces based on two metrics that are widely used in the context of crash deduplication: Recall Rate at the first k positions (RR@k) [262] and the Mean Reciprocal Rank (MRR) [72].

- ✘ **RQ3** [**Preserving LSH guarantees**]: To what extent does `DeepLSH` succeed in preserving the guarantees of LSH compared to Standard LSH methods and the baseline (`CNNH+LSH`) [127]. For this purpose, we provide recall, precision, and F-score measures adjusted to quantify the extent to which the probability-similarity constraint (6.2) has been satisfied (more details are provided hereafter).

- ✘ **RQ4** [**Runtime Analysis**]: We report the execution time required for `DeepLSH` to find near-duplicates as shown, compared to Native k-NN approach of linear time complexity.

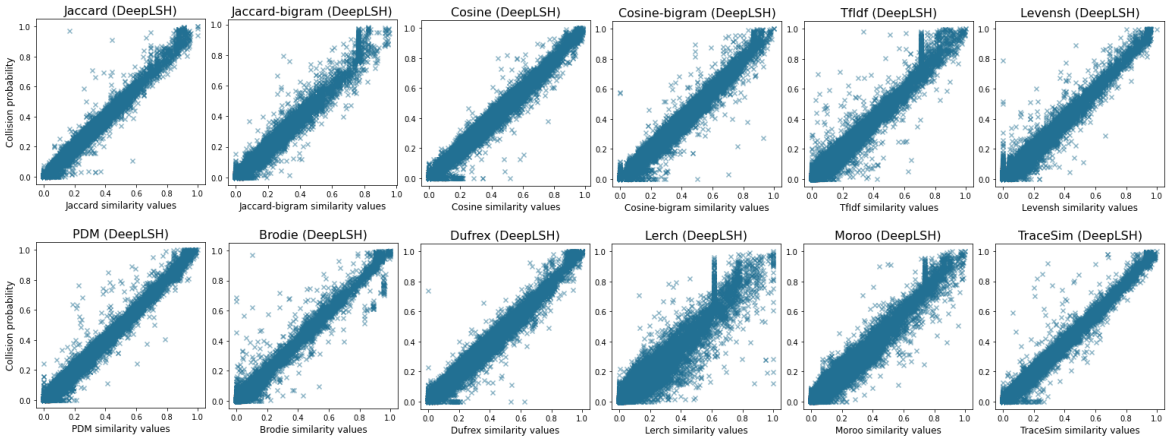


Figure 6.4: Locality-sensitive hash preserving. Correlation between the probability of hash collision and the similarity value.

6.5.2 Model Evaluation

In Figure 6.4, we highlight the strong linear correlation between the probability of hash collision and the similarity values for almost all similarity measures. It is also important to assess the capability of `DeepLSH` model to maintain the order between pairwise similarity values. For instance, for a triplet of stack traces s, p and q if $sim(s,p) > sim(s,q)$, we aim to evaluate whether the model is likely to provide hash functions s.t. $Pr[h_k(s) = h_k(p)] > Pr[h_k(s) = h_k(q)]$ for $k \in M$. For this purpose, we measure the Kendall rank correlation coefficient, between the set of similarity values, and the set of generalized Hamming similarities between the resulting hash vectors as shown in Figure 6.5. Remarkably, we obtained satisfactory results compared to our baseline (on average, 0.82 for `DeepLSH`, against 0.60 for (`CNNH+LSH`), that is, 0.22 of improvement) which permitted to achieve better and accurate results on the ANN search. Finally, thanks to the regularization conditions (6.8) and (6.10) incorporated in the objective loss function, the model yields approximate binary hash values extremely close to $\{-1, 1\}$ that are binarized/relaxed in the test phase. We seek to evaluate whether our proposed solution to deal with the binarization problem using the generalized Hamming similarity (6.9) performed in the training phase, is optimal and captures the true Hamming similarity between the discretized hash values in the test phase. Considering the TraceSim measure as an example in Figure 6.6 (on the left), we notice a strong linear correlation between the true hamming similarity calculated on the binary vectors and the approximate generalized Hamming similarity used in the loss function during the learning phase. This means that our loss optimization process is as identical as the optimization of any loss function with strictly binary values in the training phase. In the same Figure (on the right), we show the impact of not incorporating the LSH component into the model, as has been done in Hu et al. [127]. Performing LSH on the discretized vectors in a post-processing step results in a sub-optimal optimization, since the correlation between the similarity calculated using basic embedding and the true Hamming similarity is not even monotonic. Consequently, (`CNNH+LSH`) has failed to capture TraceSim similarity.

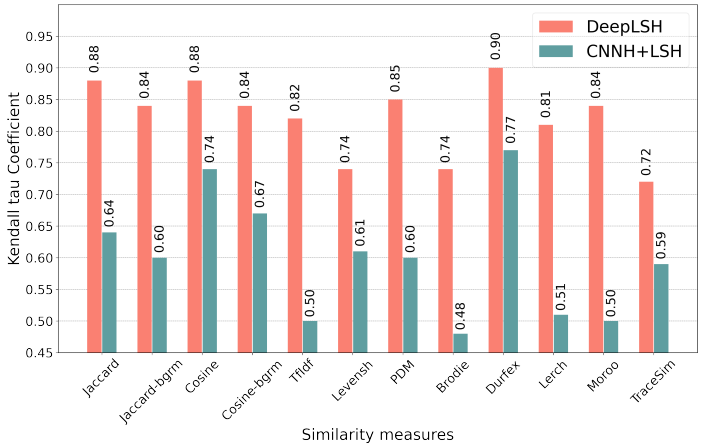


Figure 6.5: Kendall's τ coefficient between the real and predicted pairwise similarities.

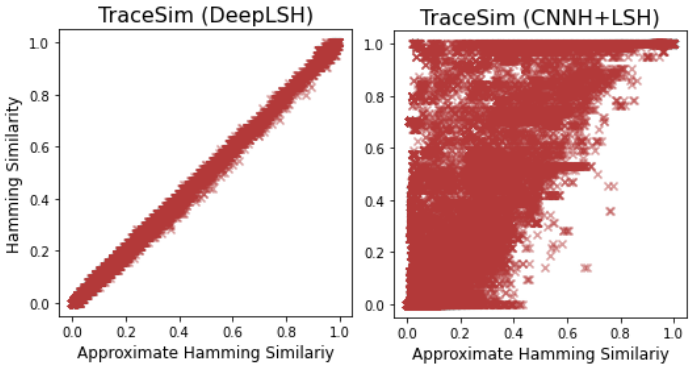


Figure 6.6: Comparison between `DeepLSH` and (`CNNH+LSH`) [127] in preserving the Hamming similarity between hash vectors.

6.5.3 Evaluation of `DeepLSH` for ANN

The objective of `DeepLSH` approach, given a similarity measure, is to generate, for a new stack trace s reported by our monitoring system, an appropriate hash vector to query a $(L, K)-$parameterized LSH for quickly and efficiently locate in a sub-linear time complexity its near-duplicates as illustrated in Figure 6.7. The hash vector contains M hash functions, partitioned across L hash tables each consisting of a concatenation of K hash functions called a meta-hash function of size $K \cdot b$. An existing stack trace $q \in \mathcal{R}_s$ is identified as a near-duplicate stack of s if it matches the stack trace s at least in one meta-hash function. The set of near-duplicate stack traces of s is denoted by $\mathcal{R}_s$. It is worth noting that the set $\mathcal{R}_s$ is sorted in the original order with respect to the similarity measure value.

LSH offers the possibility to control the trade-off between precision and recall (w.r.t LSH guarantees) by setting the hyperparameters values K and L. We thus choose to consider the tuple of hyperparameters that maximizes the F-score, as shown in Figure 6.8 (e.g., $(L, K) = (16, 4)$ for PDM). We ignore extreme cases (i.e., cases where the threshold is very small or very large, which refers to a very large value of L or K). In a first analysis, we were interested in the Recall Rate of order k. For each stack trace s that belongs to a query set $\mathcal{Q}$, we yield a set

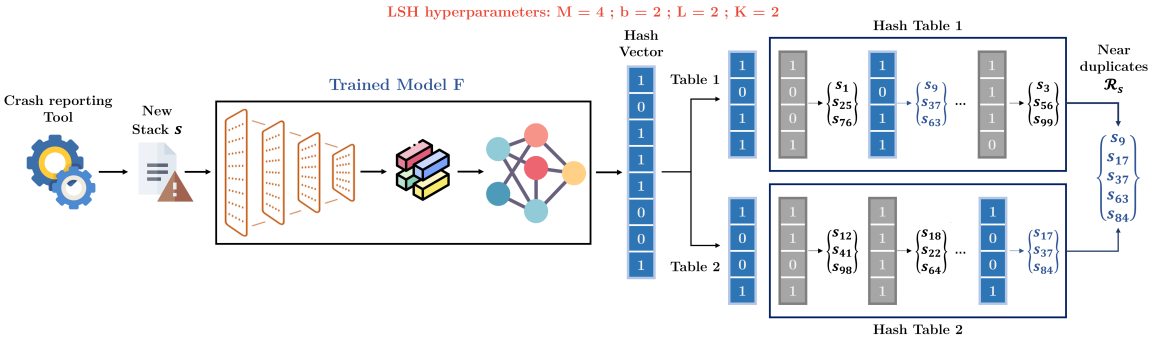


Figure 6.7: Near-duplicate stack trace retrieving using `DeepLSH` model and $(L, K)-$ parameterized LSH.

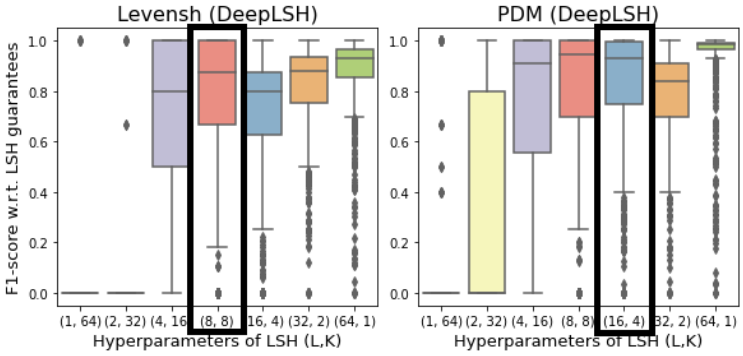


Figure 6.8: F-score boxplots w.r.t. different values of (L, K).

of its approximate nearest neighbors $\mathcal{R}_s$ (i.e., potential near-duplicates) such that $|\mathcal{R}_s| \geqslant k$ and hence,

$$\textbf{RR@k} = \frac{1}{k \cdot |\mathcal{Q}|} \sum_{s \in \mathcal{Q}} \sum_{i=1}^{k} \mathbb{1}_{[nn_i(s, *\text{args}) \in \mathcal{R}_s]},$$

where $nn_i(s, *\text{args})$ is a function that returns the real nearest neighbor of order i for the stack trace s given a set of historical stack traces and the LSH hyperparameters L and K.

In order to evaluate the ranking quality of a set of near-duplicates $\mathcal{R}_s$ for a stack trace s according to a (L, K) combination, and relative to the set of true nearest neighbors $\mathcal{T}_s$, we use the Mean reciprocal rank (MRR) [72]. This measure seeks to compute the reciprocal rank of a retrieved near-duplicate $s' \in \mathcal{R}_s$ relative to its actual position in the set of true nearest neighbors. More concretely:

$$\textbf{MRR} = \frac{1}{|\mathcal{Q}|} \sum_{s \in \mathcal{Q}} \frac{1}{|\mathcal{R}_s|} \sum_{s' \in \mathcal{R}_s} \frac{rank(s', \mathcal{R}_s)}{rank(s', \mathcal{T}_s)}$$

E.g., let's consider a given stack trace q, where we retrieve the set of its approximate nearest neighbors and subsequently sort them $\mathcal{R}_q = \{s_2, s_3, s_5\}$ and the set of its true nearest neighbor is given as $\mathcal{T}_q = \{s_1, s_2, s_3, s_4, s_5\}$. The MRR is then: $\frac{1}{3}(\frac{1}{2} + \frac{2}{3} + \frac{3}{5}) \approx 0.63$. The MRR, in this case, is over-penalized since we failed to find the true nearest neighbor s_1 in $\mathcal{R}_q$. The set $\mathcal{R}_q$ is sorted according to the original order w.r.t to the similarity measure.

Table 6.1: Comparison between the search performances of DeepLSH against the standard LSH approaches w.r.t. their addressed similarity measures and CNNH+LSH [127] in terms of Recall Rate (RR@k) and Mean Reciprocal Rank (MRR).

Similarity Measure	RR@1				RR@5				MRR			
	CNNH+LSH	DeepLSH	MinHash	SimHash	CNNH+LSH	DeepLSH	MinHash	SimHash	CNNH+LSH	DeepLSH	MinHash	SimHash
Jaccard	0.71	0.87	**0.90**	–	0.79	**0.92**	**0.92**	–	0.85	**0.96**	0.95	–
Jaccard-bigram	0.67	0.87	**0.90**	–	0.76	0.91	**0.92**	–	0.82	0.93	**0.94**	–
Cosine	**0.84**	0.81	–	0.61	0.83	**0.90**	–	0.62	**0.88**	0.87	–	0.80
Cosine-bigram	0.76	**0.84**	–	0.58	0.79	**0.93**	–	0.58	0.89	**0.91**	–	0.80
TF-IDF	0.73	**0.76**	–	0.55	0.75	**0.88**	–	0.55	0.85	**0.90**	–	0.73
Edit Distance	0.81	**0.88**	–	–	0.75	**0.94**	–	–	0.88	**0.95**	–	–
PDM	0.80	**0.84**	–	–	0.76	**0.90**	–	–	0.82	**0.93**	–	–
Brodie	0.79	**0.84**	–	–	0.76	**0.90**	–	–	0.82	**0.93**	–	–
DURFEX	0.72	**0.83**	–	–	0.79	**0.91**	–	–	0.82	**0.91**	–	–
Lerch	0.70	**0.78**	–	–	0.70	**0.85**	–	–	0.80	**0.88**	–	–
Moroo	0.75	**0.80**	–	–	0.68	**0.90**	–	–	0.80	**0.93**	–	–
TraceSim	**0.81**	0.79	–	–	0.75	**0.90**	–	–	0.84	**0.92**	–	–

The results are presented in detail in Table 6.1, according to the identified similarity measures that have been proposed to address the crash-deduplication problem. We choose 2 different values of $k = \{1, 5\}$ for the recall rate. We compare DeepLSH against MinHash, SimHash, and (CNNH+LSH). Ideally, standard LSH techniques should guarantee optimal search accuracy compared to DeepLSH, since they are proven to converge to the locality-sensitive property w.r.t. their similarity measures. Interestingly, we observe that DeepLSH almost matches the search performances of MinHash on Jaccard similarity, and outperforms SimHash. Since MinHash is a robust probabilistic model that has been specifically designed to estimate Jaccard similarity, but however can't be generalized to other similarity metrics specifically those intended to compare between stack traces. This demonstrates also that DeepLSH is not only generalizable to other complex measures but can even be used for measures where an existing LSH is already known. We also show that DeepLSH outperforms (CNNH+LSH) with a large margin on 3 different comparison metrics and for almost all similarity measures. More specifically, we notice that DeepLSH search performance is enhanced with a larger value of k up to 0.94 for Edit distance with an improvement of ~ 0.2 over (CNNH+LSH).

6.5.4 LSH guarantees Preserving

In what follows, we aim to evaluate whether DeepLSH succeeds in preserving the guarantees of LSH regarding the probability-similarity relation in (6.2). To this end, for a specific stack trace s we look for the true near-duplicate stack traces $q \in \mathcal{R}_s^{\text{True}}$ that the model should return with a (K, L) setting, s.t. $P_{K,L}(s, q) = 1 - (1 - sim(s, q)^K)^L \geqslant 0.5$, i.e., the probability to belong to the set is equal or higher than 0.5. We then derive the precision, recall and F-score between the returned set of near-duplicates $\mathcal{R}_s$ and $\mathcal{R}_s^{\text{True}}$. More formally we derive this values, s.t:

$$\textbf{Recall} = \frac{1}{|\mathcal{Q}|} \sum_{s \in \mathcal{Q}} \frac{|\mathcal{R}_s \cap \mathcal{R}_s^{\text{True}}|}{|\mathcal{R}_s^{\text{True}}|} \text{ and } \textbf{Precision} = \frac{1}{|\mathcal{Q}|} \sum_{s \in \mathcal{Q}} \frac{|\mathcal{R}_s \cap \mathcal{R}_s^{\text{True}}|}{|\mathcal{R}_s|}$$

As detailed in Table 6.2, one can notice that DeepLSH is much better than all baselines in terms of F-score on almost all similarity measures, including the accurate Minhash for Jaccard. DeepLSH showed significantly better performance in terms of recall, i.e., its ability to capture all similarities that are beyond the threshold imposed by an optimal parameterization

Table 6.2: Comparison between the precision/recall and f-score of `DeepLSH` in preserving the probability-similarity relation (6.2) against the standard LSH approaches w.r.t. theirs addressed similarity measures and `CNNH+LSH` [127].

Similarity Measure	Precision				Recall				F-score			
	CNNH+LSH	DeepLSH	MinHash	SimHash	CNNH+LSH	DeepLSH	MinHash	SimHash	CNNH+LSH	DeepLSH	MinHash	SimHash
Jaccard	0.64	**0.78**	0.76	–	0.78	**0.85**	**0.85**	–	0.70	**0.81**	0.80	–
Jaccard-bigram	0.56	**0.76**	0.70	–	0.70	0.74	**0.83**	–	0.62	**0.75**	**0.75**	–
Cosine	**0.77**	0.72	–	0.74	0.74	**0.84**	–	0.6	0.75	**0.78**	–	0.66
Cosine-bigram	**0.74**	**0.74**	–	0.66	0.72	**0.82**	–	0.41	0.73	**0.78**	–	0.50
TF-IDF	**0.85**	0.76	–	0.49	0.61	**0.86**	–	0.62	0.71	**0.81**	–	0.55
Edit Distance	0.37	**0.78**	–	–	0.78	**0.88**	–	–	0.50	**0.83**	–	–
PDM	0.68	**0.85**	–	–	0.76	**0.86**	–	–	0.72	**0.85**	–	–
Brodie	0.36	**0.83**	–	–	0.81	**0.86**	–	–	0.50	**0.84**	–	–
DURFEX	0.73	**0.78**	–	–	0.70	**0.79**	–	–	0.71	**0.78**	–	–
Lerch	0.74	**0.76**	–	–	0.57	**0.76**	–	–	0.64	**0.76**	–	–
Moroo	0.66	**0.73**	–	–	**0.85**	0.82	–	–	0.74	**0.77**	–	–
TraceSim	0.31	**0.80**	–	–	0.84	**0.88**	–	–	0.45	**0.84**	–	–

of (K, L). On the other hand, the reported precision values, as opposed to (`CNNH+LSH`), show that `DeepLSH` does not generate false positives, so that false near-duplicates are not grouped in the same bucket. In particular, on similarity measures that use the Levenshtein distance (e.g. ED, Brodie, and TraceSim), we observe a rather low precision for (`CNNH+LSH`), which shows, on the one hand, the limitation of (`CNNH+LSH`) to generalize to such metrics, and on the other hand agrees with the explanation of Figure 6.6.

6.5.5 Runtime Analysis

We evaluate the scalability of `DeepLSH` and how quickly this approach identifies, for a batch of stack traces in a large historical crash database, the most similar stacks w.r.t. a given similarity measure. We report the execution time required to find the near-duplicate candidates for $1,000$ new stack traces when querying on more than $100,000$ historical crashes (100 million queries). We compare basically against the native k-NN based approach.

Table 6.3: Comparison between the runtime required to find near-duplicate stack traces for `DeepLSH`, k-NN based approach and standard LSH techniques. `Limit` indicates that the approach failed to get a result within 10 hours.

Similarity Measure	Runtime ($\sim$ Seconds)				
	k-NN	CNNH+LSH	DeepLSH	MinHash	SimHash
Jaccard	258	30	26	57	-
Cosine	8288	15	14	-	3
TF-IDF	8510	16	15	-	4
Edit Distance	4911	29	29	-	-
PDM	10047	16	16	-	-
Brodie	Limit	27	27	-	-
DURFEX	12160	26	24	-	-
Lerch	3118	24	24	-	-
Moroo	15253	25	25	-	-
TraceSim	13050	30	30	-	-

The reported results are depicted in Table 6.3. The execution time for a native k-NN approach depends on the batch size, the size of the database, and the computational complexity of the similarity measure. With the latter, most similarity measures under the conditions

Table 6.4: Mapping the contribution `DeepLSH` to our proposed taxonomy.

Context	Focus Area	Incident Deduplication
	Maintenance Layer	Functional
	Scoop	ERP Software System
Data	Source and Type	Stack traces ⟶ Sequences
	Feature Eng	Parsing of Stack traces
Model	Approach	Locality-Sensitive Hashing embedded in Siamese NN
	Paradigm	Supervised
	Metrics	Mean Reciprocal Rank (MRR) Recall Rate of order k.
	Availability	Data and Code (`https://github.com/RemilYoucef/deep-locality-sensitive-hashing`)
Particularities		Scalability, Maintainability, In context-Evaluation
Contribution		First similarity-agnostic method that utilizes hashing for the crash deduplication problem.

described above require more than an hour to return any results (more than 3 hours for DURFEX, Moroo, and TraceSim) and no results returned by Brodie within 10 hours. On the other hand, `DeepLSH` only depends on the number of hash tables which does not exceed 64 tables. We notice that the runtime is roughly constant and around $24 \sim 27$ seconds on average. Remarkably, `DeepLSH` is even faster than MinHash for Jaccard Similarity. SimHash, on the other hand, has proven to be faster, and (`CNNH+LSH`) has been similar to `DeepLSH` in runtime, but as seen above, both approaches show poor search performance and lower guarantees.

6.6 Discussion

In this chapter, we tackled the important task of fast and efficient automatic crash bucketing in software development. We investigated the potential of locality-sensitive hashing (LSH) for this purpose, leveraging its sublinear performance and theoretical guarantees in terms of accuracy for similarity search. This approach offers significant advantages when dealing with large datasets, yet surprisingly, LSH has not been explored in the crash bucketing literature. The main reason for the lack of consideration of LSH in crash bucketing research is the challenge of deriving hash functions that satisfy the locality-sensitive property for advanced and complex crash bucketing metrics. To address this gap, we first the concept of Locality-sensitive hashing in the context of approximate nearest neighbors search. Then, we propose a novel, parameterizable approach dubbed `DeepLSH`. We introduced an original objective loss function, complemented by appropriate regularizations, enabling convergence to the desired locality-sensitive property. By doing so, `DeepLSH` had the capability to mimic any given similarity metrics, thereby enhancing and improving the time and efficiency of near-duplicate crash report detection. Overall, our findings highlight the untapped potential of LSH in the crash-bucketing domain. Table 6.4 provides the positioning of our contribution according to our taxonomy.

Chapter 7

Conclusion

Back in 2020, the journey of this thesis began with a comprehensive evaluation of the current maintenance landscape at Infologic. This involved examining existing routines, maintenance protocols, incident management processes from reporting to mitigation, and the engagement of teams in taking responsibility for these incidents. The objective was to identify areas and opportunities for improvement, with a focus on automating processes that can be better managed through automated actions, while also leveraging human expertise to ensure efficiency, reliability, usability, and trust within the maintenance team. This thesis was motivated by a real industrial context, presenting concrete problems and studying existing solutions in both academia and industry, relying on software engineering, machine learning, and community practices. The aim was to bring innovative solutions that not only benefited the specific use case but also addressed identified gaps in the current state of the art.

Our objective was to establish a data-driven approach to effectively and efficiently manage incidents while optimizing available resources. This objective aligns well with the concept of AIOps, which emerged as a defined policy in 2017, and generalize them to common cases. However, this approach is not as straightforward, especially considering the challenges faced in this thesis. Designing a performant model for specific cases does not guarantee its utility across various scenarios in the current or future state. Additionally, findings from machine learning models do not necessarily apply to software analytics domains like AIOps due to factors such as data quality and the inherent complexity of software systems.

The first lesson learned was that designing, implementing, and deploying machine learning models in real-world software systems scenarios is not trivial. To transfer knowledge from AI models to the AIOps domain, a set of requirements needed to be investigated. Furthermore, to address our problems related to AIOps and pioneer research in this area, we encountered a diverse range of contributions from various specialized disciplines. For example, domains like software defect prediction, software aging, and anomaly detection methods provide independent taxonomies and different ways to handle data. They may focus on single or multiple use cases or address specific maintenance layers (e.g., functional, network, or hardware). This diversity makes it challenging to extend their work to cover other topics.

In fact, five main issues related to existing work in our studied domain can be identified. Firstly, the field of AIOps lacks unified terminology due to its novelty and contributions from various specialized disciplines. Additionally, the requirements for transferring knowledge from data-driven approaches to AIOps models are not well-defined or suited to real-world scenar-

ios. Secondly, obtaining high-quality labeled data is a challenge, making the construction of accurate AIOps solutions complex. This leads us to the third limitation, where existing methods primarily focus on developing predictive models for anomaly detection and failure prediction, overlooking the challenges posed by data quality and complexity. Descriptive models, which historically have been effective in dealing with these challenges by employing data mining techniques to extract informative and exceptional patterns within a contextualized topology, are less adopted. Fourthly, most predictive models rely on opaque models that are difficult to interpret, raising ethical and trust concerns among practitioners. Finally, in the maintenance context, the ultimate objective is to optimize and control maintenance costs, with time being a crucial factor. However, existing work often overlooks scalability and focuses to assess the accuracy of the model.

In this chapter, we concisely summarize the primary contributions made in this thesis to address these challenges. Additionally, we present an outlook highlighting promising research directions for the future in both academic and industrial domains.

7.1 Summary of Contributions

In Chapter 2, our objective was to establish a fundamental knowledge base for AIOps based on our perspective, which we believe can serve as a valuable resource for future research. In order to achieve this, we introduced a standardized terminology that encompasses all relevant concepts, along with a comprehensive taxonomy that effectively organizes the extensive knowledge surrounding AIOps. We offered a clear comparison of terms associated with incident management in the realm of AIOps, as well as a detailed explanation of the various data sources and evaluation metrics that should be utilized in this context. Additionally, we conducted a thorough examination of previous research and related efforts within the field of AIOps, and presented a list of essential requirements that should be taken into account during the development of AIOps solutions.

In Chapter 3, we proposed the utilization of Subgroup Discovery as a solution for addressing the limitations associated with data complexity, exceptional cases, and regularities within large datasets concerning a specific target problem, model implementations, and interpretability. We harnessed the power of this framework to extract valuable and easily understandable patterns from extensive datasets that possess abnormal distributions compared to the overall data. Furthermore, we took advantage of its efficiency in handling complex, diverse, and extensive datasets, all while providing flexibility in user interaction and incorporating domain-specific knowledge and subjective criteria. Following the formal definition of its fundamental components, such as pattern syntax, interestingness measure, target concept, and mining algorithm, we proceeded to explore a practical and direct application of these concepts in a real-world scenario. This application involved identifying SQL data patterns that demonstrate correlations with various performance degradation issues.

In Chapter 4, we showcased the effectiveness of Subgroup Discovery in assisting explainable artificial intelligence (XAI) approaches, enabling them to offer concise and contextualized explanations while maintaining a high level of accuracy comparable to black box models in the context of AIOps. Our investigation focused on an intriguing use case involving the assignment of incident reports to appropriate responsible teams. Typically, this task relies on sophisticated NLP techniques that require explanations to establish trust among

practitioners. To address this challenge, we proposed an algorithm based on the Subgroup Discovery framework. This algorithm identifies groups of incident data that share similar properties, providing a contextualized description for each group. We also developed a human-interpretable interface that highlights the importance of each feature within each subgroup for a given class. Notably, this use case presented a novel scenario in which the target concept was defined as an explanation model rather than a single attribute. As a result, we needed to introduce an appropriate interestingness measure and design effective yet scalable algorithms to generate meaningful results.

In Chapter 5, we addressed the challenging problem of diagnosing the underlying causes of Java out-of-memory incidents in an effective manner. The diagnosis of such incidents is typically difficult due to the complexity involved in analyzing large histograms that provide information about memory-occupying classes suspected of causing saturation. Moreover, the hierarchical relationship between packages and classes within these histograms adds another layer of intricacy. To tackle this specific problem, as well as related problems that involve hierarchical data organization, we proposed the introduction of a novel and versatile Subgroup Discovery framework with hierarchical target concepts. This framework enables us to define a pattern syntax tailored to the problem and a subjective quality measure that effectively identifies relevant, non-redundant, and noise-resistant subgroups. It also takes into account the user's prior knowledge regarding similar historical issues and interesting patterns. Additionally, we proposed a search mining algorithm designed to efficiently identify subsets of incidents. This approach was able to pinpoint the potential root causes behind memory issues for many of our relevant use-cases at Infologic.

In Chapter 6, we tackled the specific issue of enhancing performance and scalability in incident management. One significant application of this approach is to speed up the detection of duplicate incidents. Instead of clustering, we formulated the problem as an approximate nearest neighbor search and suggested employing locality-sensitive hashing (LSH) combined with a specific similarity measure to compare crash reports, particularly stack traces. Although LSH is effective in handling large datasets and offers sublinear performance and theoretical guarantees for accurate similarity search, it is challenging to derive hash functions that satisfy the locality-sensitive property for advanced crash deduplication metrics. To overcome this challenge, we propose a Siamese deep neural network architecture with an original loss function that approximates the locality-sensitive property of any given similarity measures, especially those designed for comparing similarity measures. We have designed this unique loss function to preserve the locality-sensitive property while dealing with the optimization of non-smooth objective functions caused by binarization.

7.2 Perspectives

Through the contributions made in this thesis, our primary objective was to address the open research questions mentioned earlier and simultaneously tackle real-world problems driven by our industrial requirements. We have accomplished several significant results, and some of them are already being integrated into our production system at Infologic. However, it is essential to acknowledge that there is always room for improvement, and we may encounter minor or major limitations during the implementation, testing, or deployment phases of our solutions. These limitations can arise in both applicative and theoretical aspects. In the

following, we will outline our perspectives on enhancing the robustness of our solutions and expanding their applicability to cover a broader range of use cases.

Firstly, as outlined in the introduction, we have proposed a comprehensive revision of the AIOps domain. This includes reevaluating its categorization, identifying the necessary requirements, exploring techniques for data cleaning and processing, and addressing the challenges associated with deployment, training, and updating effective data-driven approaches. This endeavor has allowed us to shift our maintenance efforts away from traditional routines, which have proven to be less effective in handling and qualifying in a timely manner some incidents and diagnosing issues. These routines also lack the high standards required to optimize the cost of maintenance actions. Consequently, we have initiated a new project aimed at automating the entire maintenance process. This project involves a redesign of the maintenance workflow at Infologic. To achieve this, we have developed our proposed automated iterative workflow that incorporates both human knowledge and insights gained from past experiences. This workflow guides maintenance tickets through clear phases, each utilizing the best approaches to address incidents based on their context, as proposed in our taxonomy. We also ensure that the nature of the data is considered and that the evaluation process is fair, while ensuring compliance with the AIOps requirements outlined in Chapter 2. In summary, our efforts focus on developing an end-to-end incident management procedure that covers all types of incidents. We are actively deploying its components to gradually replace traditional routines where applicable.

Our proposed methods, as described in this thesis, are currently in the process of being deployed. We are particularly concentrating on the incident triage model, where we aim to incorporate explanations of the predictions made for service teams. This additional step enhances trust and encourages practitioners to utilize this model. Furthermore, we are working on leveraging the Subgroup Discovery framework to analyze SQL queries. Hereafter, we to provide some theoretical and practical perspectives regarding the application of our proposed methods.

Regarding our contribution, `SD4-SQL`, there are certain limitations in its current implementation and deployment. Presently, it only supports `SELECT` queries and restricts users to choosing only one property at a time. To address these limitations, we have planned several improvements. Firstly, we intend to extend our SQL parsing query to support a broader range of query types, including `INSERT`, `UPDATE`, `DELETE`, and `CREATE` queries. This expansion will follow the same parsing scheme for consistency and improved functionality. Furthermore, we believe that enhancing the pattern syntax to accommodate tree structures, such as the JSON format, instead of the conventional tabular format, would be advantageous. This modification aims to prevent sparsity issues in data and significantly reduce the complexity of the mining algorithm when scanning the set of potential interesting patterns. Additionally, integrating new query types also necessitates considering new measures of interestingness. Different query types exhibit diverse behaviors, and as a result, the current measures may not effectively indicate the target problem. Therefore, we are committed to developing new interestingness measures that align with the characteristics of each query type. Moreover, the current framework solely incorporates objective measures. To provide a more comprehensive evaluation, we are determined to introduce subjective criteria iteratively. This addition will allow us to ignore patterns that may not be surprising to database administrators, or patterns that are expected based on their expertise and perspective. Lastly, we recognize the importance of enabling users to consider multiple target attributes simultaneously. To

enhance user experience, we are planning to incorporate a feature that allows users to select and analyze multiple attributes at once, according to their preferences and requirements.

The third contribution can benefit from several improvements, particularly in the incident assignment model and the explanation mechanism. Firstly, instead of relying on LSTM-attention units, we can leverage newer NLP models such as transformers. This shift is motivated by the presence of technical language in our incident reports, specific to our industrial settings. Additionally, the use of French language makes it challenging to apply existing stemming and lemmatization models to our use cases, often requiring manual preprocessing efforts before feeding the data into a prediction model. To address this, we plan to train a language model capable of learning the main technical language used by our customers and internal staff at Infologic. This approach will provide dynamic embeddings of incidents with respect to their context, effectively capturing the semantic aspects of the incidents, which are often similar as discussed in the introduction. Moreover, the incident triage model can be enhanced by incorporating not only human textual data, topology, and performance metrics, but also semi-structured data such as stack traces and image captures frequently submitted in incident reports. Extracting useful information from stack traces may require parsing techniques, while image captures often require optical character recognition (OCR) to extract relevant details. In terms of the explanation mechanism, one interesting direction is to extend our approach to compare the behaviors of different models and uncover what each model captures in various situations. This could involve using more sophisticated explainers such as SHAP, expanding beyond relying solely on the LIME methodology or local white box models. Another improvement could involve incorporating a constraint based on the distribution of classes per subgroup, especially in the case of multi-label classification. For instance, one possible approach is to prioritize grouping together incidents that are more likely to belong to a single class. This would result in analyzing fewer explanations within the subgroup model, necessitating a revision of the pattern syntax.

Regarding our `SCA-Miner` algorithm, we have already observed its ability to uncover surprising and actionable patterns that have successfully identified numerous Java memory incidents in the past. However, we acknowledge that its runtime is not optimal. This can be attributed to the requirement for mining algorithms that need to iterate through a large set of potentially interesting subgroup candidates and retrieve the best contrastive antichain for each subgroup. In reality, this runtime issue is not overly critical because our framework is not designed for daily use. Instead, it operates on a batch of Java memory incidents to generate a report on possible root causes. As a result, it can be sufficiently executed to analyze heap dumps collected over a one-week period. The entire process takes only a few minutes to produce five interesting patterns. Nonetheless, if we aim to improve this aspect, alternative approaches like MCTS and genetic algorithms can be explored. Furthermore, we must pay particular attention to the temporal dimension when maintaining prior knowledge. Additionally, incorporating time-related pattern constraints would be beneficial for characterizing continuous increases. It's worth noting that this framework is not limited to analyzing Java memory issues. It can also be applied to analyze execution traces within the source code. The data structure in this case is hierarchical, with the root representing the requested service. The hierarchy then describes the workflow the program follows to produce the final result. Each node represents a basic instruction or an internal node such as a function, module, or loop block. Each node is associated with the time it takes to complete its execution, providing valuable insights into the program's behavior

Our recent contribution, `DeepLSH`, introduces a novel hashing scheme that effectively preserves the locality-sensitive hashing property for any given similarity measure. Experimental evaluations have demonstrated the reliability of this framework, particularly in the context of incident deduplication detection. Specifically, we have extensively evaluated it on a large historical dataset of crash reports, utilizing stack trace-based similarity measures as our ground truth. This was necessary due to the limited availability of labeled information for each measure when applied to vast datasets of stack traces. To determine the most suitable similarity measure for our specific use case, one potential solution is to implement an AB testing mechanism. This would help identify the measure that best aligns with our requirements. Additionally, we could propose our own similarity measure or utilize a weighted combination of the most favored similarity measures as determined by our developers. Deploying such a model poses certain challenges, particularly in terms of updating the model with newly reported stack traces. We need to establish mechanisms to ensure the model remains up-to-date and reflects the latest information accurately. Furthermore, we aim to extend the capabilities of this framework beyond crash reports to include near-duplicate incident reports. This involves applying the hashing technique to textual data along with other relevant information contained in incident reports. In our preliminary tests, we employed a pre-trained fastText model to embed incident reports and used the word mover's distance [128] to compare the resulting vectors. This approach has shown promising results, indicating the potential of our model in retrieving near-duplicate incident reports.

Bibliography

1. Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. An evaluation of similarity coefficients for software fault localization. In *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*, pages 39–46. IEEE, 2006. 63

2. Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. Spectrum-based multiple fault localization. In *2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 88–99. IEEE, 2009. 29, 41, 63

3. Florian Adriaens, Jefrey Lijffijt, and Tijl De Bie. Subjectively interesting connecting trees and forests. *Data Min. Knowl. Discov.*, 33(4):1088–1124, 2019. 18

4. Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollár, Arunprasad P. Marathe, Vivek R. Narasayya, and Manoj Syamala. Database tuning advisor for microsoft SQL server 2005: demo. In *SIGMOD Conference*, pages 930–932. ACM, 2005. 83

5. Javad Akbarnejad, Gloria Chatzopoulou, Magdalini Eirinaki, Suju Koshy, Sarika Mittal, Duc On, Neoklis Polyzotis, and Jothi S Vindhiya Varman. SQL QueRIE recommendations. *Proceedings of the VLDB Endowment*, 3(1-2):1597–1600, 2010. 84

6. V Akila, G Zayaraz, and V Govindasamy. Bug triage in open source systems: a review. *International Journal of Collaborative Enterprise*, 4(4):299–319, 2014. 53, 54

7. Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. Similarity measures for OLAP sessions. *Knowledge and information systems*, 39(2): 463–489, 2014. 48, 84

8. Javier Alonso, Jordi Torres, Josep Ll Berral, and Ricard Gavalda. Adaptive on-line software aging prediction based on machine learning. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 507–516. IEEE, 2010. 34, 57

9. Turky N Alotaiby, Alanoud Alhakbani, Nujood Alwhibi, Gaseb Alotaibi, and Saleh A Alshebeili. Locality sensitive hashing for ecg-based subject identification. In *ICECTA*, pages 1–4. IEEE, 2019. 20

10. Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. Is it a bug or an enhancement? a text-based approach to classify change requests. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, pages 304–318, 2008. 35

11. John Anvik, Lyndon Hiew, and Gail C Murphy. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 35–39, 2005. 35

12. Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. Clustering-based materialized view selection in data warehouses. In *East European conference on advances in databases and information systems*, pages 81–95. Springer, 2006. 48, 84

13. Natalia Arzamasova, Martin Schäler, and Klemens Böhm. Cleaning antipatterns in an SQL query log. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):421–434, 2017. 82

14. Mona Attariyan, Michael Chow, and Jason Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 307–320, 2012. 36, 63

15. Martin Atzmueller. Subgroup discovery. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 5(1):35–49, 2015. 17, 68, 70

16. Martin Atzmueller, Stephan Doerfel, and Folke Mitzlaff. Description-oriented community detection using exhaustive subgroup discovery. *Inf. Sci.*, 329:965–984, 2016. 18, 74

17. Martin Atzmüller and Frank Puppe. Semi-automatic refinement and assessment of subgroup patterns. In *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference*, pages 323–328. AAAI Press, 2008. 81

18. Martin Atzmüller, Frank Puppe, and Hans-Peter Buscher. Exploiting background knowledge for knowledge-intensive subgroup discovery. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 647–652. Professional Book Center, 2005. 81

19. Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3395–3404, 2020. 55

20. M. Gethsiyal Augasta and T. Kathirvalavakumar. Reverse engineering the neural networks for rule extraction in classification problems. *Neural Processing Letters*, 35(2): 131–150, 2012. 19, 94

21. Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004. 29

22. Mozhgan Azimpourkivi, Umut Topkara, and Bogdan Carbunar. A secure mobile authentication alternative to biometrics. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 28–41, 2017. 20

23. Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A Maltz, and Ming Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. *ACM SIGCOMM Computer Communication Review*, 37(4): 13–24, 2007. 45, 62

24. Sean Banerjee, Bojan Cukic, and Donald Adjeroh. Automated duplicate bug report classification using subsequence matching. In *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*, pages 74–81. IEEE, 2012. 35, 61

25. Abdul Ali Bangash, Hareem Sahar, Abram Hindle, and Karim Ali. On the time-based conclusion stability of cross-project defect prediction models. *Empirical Software Engineering*, 25(6):5047–5083, 2020. 37

26. Kevin Bartz, Jack W. Stokes, John C. Platt, Ryan Kivett, David Grant, Silviu Calinoiu, and Gretchen Loihle. Finding similar failures using callstack similarity. In *SysML*, 2008. 61, 144

27. Mehdi Bateni and Ahmad Baraani. Time window management for alert correlation using context information and classification. *International Journal of Computer Network & Information Security*, 5(11), 2013. 64

28. Soeren Becker, Florian Schmidt, Anton Gulenko, Alexander Acker, and Odej Kao. Towards aiops in edge computing environments. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 3470–3475. IEEE, 2020. 38

29. Adnene Belfodil, Aimene Belfodil, Anes Bendimerad, Philippe Lamarre, Céline Robardet, Mehdi Kaytoue, and Marc Plantevit. FSSD - A fast and efficient algorithm for subgroup set discovery. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 91–99. IEEE, 2019. 79, 81

30. Aimene Belfodil, Adnene Belfodil, and Mehdi Kaytoue. Anytime subgroup discovery in numerical domains with guarantees. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part II*, volume 11052, pages 500–516. Springer, 2018. 79

31. Luca Bellani, Michele Compare, Piero Baraldi, and Enrico Zio. Towards developing a novel framework for practical phm: A sequential decision problem solved by reinforcement learning and artificial neural networks. *International Journal of Prognostics and Health Management*, 10(4), 2019. 58

32. Ahmed Anes Bendimerad. *Mining Useful Patterns in Attributed Graphs. (Fouille de motifs utiles dans les graphes attribués)*. PhD thesis, University of Lyon, France, 2019. 76, 82

33. Anes Bendimerad, Jefrey Lijffijt, Marc Plantevit, Céline Robardet, and Tijl De Bie. Contrastive antichains in hierarchies. In *Proceedings of the 25th ACM SIGKDD*, pages 294–304, 2019. 115, 121, 122

34. Anes Bendimerad, Youcef Remil, Romain Mathonat, and Mehdi Kaytoue. On-premise infrastructure for aiops in a software editor sme: An experience report. In *31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESECFSE (Accepted)*, 2023. 38

35. Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. 136

36. Tarek Berghout and Mohamed Benbouzid. A systematic guide for predicting remaining useful life with machine learning. *Electronics*, 11(7):1125, 2022. 54, 58

37. Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Database Theory - ICDT '99, 7th International Conference, 1999, Proceedings*, pages 217–235, 1999. 137

38. Pamela Bhattacharya and Iulian Neamtiu. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *2010 IEEE International Conference on Software Maintenance*, pages 1–10. IEEE, 2010. 46

39. Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2007. 124

40. Jasmin Bogatinovski, Sasho Nedelkoski, Alexander Acker, Florian Schmidt, Thorsten Wittkopp, Soeren Becker, Jorge Cardoso, and Odej Kao. Artificial intelligence for it operations (aiops) workshop white paper. *arXiv preprint arXiv:2101.06054*, 2021. 2, 13, 32, 54, 68

41. Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017. 46

42. Mario Boley, Claudio Lucchese, Daniel Paurat, and Thomas Gärtner. Direct local pattern sampling by efficient two-step random procedures. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 582–590. ACM, 2011. 79

43. Mario Boley, Sandy Moens, and Thomas Gärtner. Linear space direct pattern sampling using coupling from the past. In Qiang Yang, Deepak Agarwal, and Jian Pei, editors, *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 69–77. ACM, 2012. 79

44. Guillaume Bosc, Jean-François Boulicaut, Chedy Raïssi, and Mehdi Kaytoue. Anytime discovery of a diverse set of patterns with monte carlo tree search. *Data Min. Knowl. Discov.*, 32(3):604–650, 2018. 79

45. Marcello Braglia, Gionata Carmignani, Marco Frosolini, and Francesco Zammori. Data classification and mtbf prediction with a multivariate analysis approach. *Reliability Engineering & System Safety*, 97(1):27–35, 2012. 54

46. Lionel C. Briand, John W. Daly, and Jurgen K Wust. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on software Engineering*, 25(1):91–121, 1999. 13

47. Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Comput. Networks*, 29(8-13):1157–1166, 1997. 135, 138, 139, 143

48. Mark Brodie, Sheng Ma, Guy M. Lohman, Laurent Mignet, Natwar Modani, Mark Wilding, Jon Champlin, and Peter Sohn. Quickly finding known software problems via automated symptom matching. In *Second International Conference on Autonomic Computing (ICAC 2005)*, pages 101–110. IEEE Computer Society, 2005. 48, 61, 134, 144

49. Zaharah A Bukhsh, Hajo Molegraaf, and Nils Jansen. A maintenance planning framework using online and offline deep reinforcement learning. *Neural Computing and Applications*, pages 1–12, 2023. 38

50. Saul Carliner. An overview of online learning. 2004. 38

51. Vittorio Castelli, Richard E Harper, Philip Heidelberger, Steven W Hunter, Kishor S Trivedi, Kalyanaraman Vaidyanathan, and William P Zeggert. Proactive management of software aging. *IBM Journal of Research and Development*, 45(2):311–332, 2001. 57

52. Peggy Cellier, Mireille Ducassé, Sébastien Ferré, and Olivier Ridoux. Formal concept analysis enhances fault localization in software. In *Formal Concept Analysis: 6th International Conference, ICFCA 2008, Montreal, Canada, February 25-28, 2008. Proceedings 6*, pages 273–288. Springer, 2008. 41, 63, 68

53. Thanyalak Chalermarrewong, Tiranee Achalakul, and Simon Chong Wee See. Failure prediction of data centers using time series and fault tree analysis. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 794–799. IEEE, 2012. 58

54. Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009. 54

55. Yen-Yu Chang, Pan Li, Rok Sosic, MH Afifi, Marco Schweighauser, and Jure Leskovec. F-fade: Frequency factorization for anomaly detection in edge streams. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 589–597, 2021. 55

56. Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002. 20, 135, 138, 139, 143

57. Surajit Chaudhuri, Ashish Kumar Gupta, and Vivek Narasayya. Compressing SQL workloads. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 488–499, 2002. 48, 82

58. Surajit Chaudhuri, Vivek Narasayya, and Prasanna Ganesan. Primitives for workload summarization and implications for SQL. In *Proceedings 2003 VLDB Conference*, pages 730–741. Elsevier, 2003. 82

59. Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 364–375. IEEE, 2019. 12, 13, 35, 46, 60, 94, 98

60. Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, et al. How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 373–384, 2020. 59

61. Tse-Hsun Chen, Weiyi Shang, Zhen Ming Jiang, Ahmed E Hassan, Mohamed Nasser, and Parminder Flora. Detecting performance anti-patterns for applications developed using object-relational mapping. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1001–1012, 2014. 48, 82

62. Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, et al. Outage prediction and diagnosis for cloud service systems. In *The World Wide Web Conference*, pages 2659–2665, 2019. 29

63. Zhuangbin Chen, Yu Kang, Feng Gao, Li Yang, Jeffrey Sun, Zhangwei Xu, Pu Zhao, Bo Qiao, Liqun Li, Xu Zhang, et al. Aiops innovations of incident management for cloud services. 2020. 14

64. Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1487–1497, 2020. 9, 13, 16, 28, 33, 46, 54, 68

65. Sigmund Cherem, Lonnie Princehouse, and Radu Rugina. Practical memory leak detection using guarded value-flow analysis. In Jeanne Ferrante and Kathryn S. McKinley, editors, *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, 2007*, pages 480–491. ACM, 2007. 114

66. Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994. 13, 41

67. Michael Chow, David Meisner, Jason Flinn, Daniel Peek, and Thomas F Wenisch. The mystery machine: End-to-end performance analysis of large-scale internet services. In *11th {USENIX} symposium on operating systems design and implementation ({OSDI} 14)*, pages 217–231, 2014. 13, 29

68. Peter Clark and Tim Niblett. The CN2 induction algorithm. *Mach. Learn.*, 3:261–283, 1989. 80

69. Ira Cohen, Jeffrey S Chase, Moises Goldszmidt, Terence Kelly, and Julie Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, volume 4, pages 16–16, 2004. 58

70. Andrew A. Cook, Goksel Misirli, and Zhong Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet Things J.*, 7(7):6481–6494, 2020. 54

71. Thomas M Cover and Joy A Thomas. Entropy, relative entropy and mutual information. *Elements of information theory*, 2:1–55, 1991. 122

72. Nick Craswell. *Mean Reciprocal Rank*, pages 1703–1703. Springer US, 2009. 53, 145, 148

73. Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. Logram: Efficient log parsing using n n-gram dictionaries. *IEEE Transactions on Software Engineering*, 48(3):879–892, 2020. 43, 68

74. Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In *2010 7th IEEE working conference on mining software repositories (MSR 2010)*, pages 31–41. IEEE, 2010. 53

75. Yingnong Dang, Rongxin Wu, Hongyu Zhang, Dongmei Zhang, and Peter Nobel. Rebucket: A method for clustering duplicate crash reports based on call stack similarity. In *34th International Conference on Software Engineering, ICSE*, pages 1084–1093, 2012. 61, 134, 136, 144

76. Yingnong Dang, Qingwei Lin, and Peng Huang. Aiops: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5. IEEE, 2019. 2, 9, 13, 14, 28, 32, 36, 38, 68

77. Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry*, pages 253–262, 2004. 139, 143

78. Narjes Davari, Bruno Veloso, Gustavo de Assis Costa, Pedro Mota Pereira, Rita P Ribeiro, and João Gama. A survey on data-driven predictive maintenance for the railway industry. *Sensors*, 21(17):5739, 2021. 16, 54

79. Nickolas Allen Davis, Abdelmounaam Rezgui, Hamdy Soliman, Skyler Manzanares, and Milagre Coates. Failuresim: a system for predicting hardware failures in cloud data centers using neural networks. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 544–551. IEEE, 2017. 58

80. Tijl De Bie. Maximum entropy models and subjective interestingness. *Data Mining and Knowledge Discovery*, 23(3):407–446, 2011. 77, 82, 115, 121, 122

81. Cláudio Rebelo de Sá, Wouter Duivesteijn, Paulo J. Azevedo, Alípio Mário Jorge, Carlos Soares, and Arno J. Knobbe. Discovering a taste for the unusual: exceptional models for preference mining. *Mach. Learn.*, 107(11):1775–1807, 2018. 76

82. Shaleen Deep, Anja Gruenheid, Paraschos Koutris, Jeffrey F. Naughton, and Stratis Viglas. Comprehensive and efficient workload compression. *Proc. VLDB Endow.*, 14 (3):418–430, 2020. 48, 85

83. Karel Dejaeger, Thomas Verbraken, and Bart Baesens. Toward comprehensible software fault prediction models using bayesian network classifiers. *IEEE Transactions on Software Engineering*, 39(2):237–257, 2012. 34

84. Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4027–4035, 2021. 54

85. Houtao Deng. Interpreting tree ensembles with intrees. *International Journal of Data Science and Analytics*, 7(4):277–287, 2019. 19, 94

86. Junning Deng, Bo Kang, Jefrey Lijffijt, and Tijl De Bie. Mining explainable local and global subgraph patterns with surprising densities. *Data Min. Knowl. Discov.*, 35(1): 321–371, 2021. 18, 74

87. Tejinder Dhaliwal, Foutse Khomh, and Ying Zou. Classifying field crash reports for fixing bugs: A case study of mozilla firefox. In *IEEE 27th International Conference on Software Maintenance, ICSM*, pages 333–342, 2011. 48, 134

88. Josu Díaz-de Arcaya, Ana I Torre-Bastida, Raúl Miñón, and Aitor Almeida. Orfeon: An aiops framework for the goal-driven operationalization of distributed analytical pipelines. *Future Generation Computer Systems*, 140:18–35, 2023. 38

89. Rui Ding, Qiang Fu, Jian-Guang Lou, Qingwei Lin, Dongmei Zhang, Jiajun Shen, and Tao Xie. Healing online service systems via mining historical issue repositories. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 318–321, 2012. 65, 68

90. Thanh-Toan Do, Anh-Dzung Doan, and Ngai-Man Cheung. Learning to hash with binary deep neural network. In *Computer Vision - ECCV 2016 - 14th European Conference*, pages 219–234, 2016. 135, 141, 142

91. Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1285–1298, 2017. 43, 56

92. Wouter Duivesteijn and Julia Thaele. Understanding where your classifier does (not) work - the scape model class for EMM. In Ravi Kumar, Hannu Toivonen, Jian Pei, Joshua Zhexue Huang, and Xindong Wu, editors, *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, pages 809–814. IEEE Computer Society, 2014. 76

93. Wouter Duivesteijn, Arno J. Knobbe, Ad Feelders, and Matthijs van Leeuwen. Subgroup discovery meets bayesian networks – an exceptional model mining approach. In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, pages 158–167. IEEE Computer Society, 2010. 76

94. Wouter Duivesteijn, Ad Feelders, and Arno J. Knobbe. Different slopes for different folks: mining for exceptional regression models with cook's distance. In Qiang Yang, Deepak Agarwal, and Jian Pei, editors, *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 868–876. ACM, 2012. 76

95. Wouter Duivesteijn, Ad Feelders, and Arno J. Knobbe. Exceptional model mining - supervised descriptive local pattern mining with complex target concepts. *Data Min. Knowl. Discov.*, 30(1):47–98, 2016. 69, 76, 79

96. Marco D'Ambros, Michele Lanza, and Romain Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17:531–577, 2012. 16, 53, 54, 56

97. Erki Eessaar. On query-based search of possible design flaws of SQL databases. In *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*, pages 53–60. Springer, 2015. 82

98. Nosayba El-Sayed, Hongyu Zhu, and Bianca Schroeder. Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1333–1344. IEEE, 2017. 14

99. Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 1022–1029. Morgan Kaufmann, 1993. 75

100. Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI/MIT Press, 1996. 77

101. Olga Fink. Data-driven intelligent predictive maintenance of industrial assets. *Women in Industrial and Systems Engineering: Key Advances and Perspectives on Emerging Topics*, pages 589–605, 2020. 4

102. Damien Fourure, Muhammad Usama Javaid, Nicolas Posocco, and Simon Tihon. Anomaly detection: How to artificially increase your f1-score with a biased evaluation protocol. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part IV*, pages 3–18. Springer, 2021. 15

103. Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*, pages 149–158. IEEE, 2009. 43, 55

104. Jiaqi Gao, Nofel Yaseen, Robert MacDavid, Felipe Vieira Frujeri, Vincent Liu, Ricardo Bianchini, Ramaswamy Aditya, Xiaohang Wang, Henry Lee, David A. Maltz, Minlan Yu, and Behnaz Arzani. Scouts: Improving the diagnosis process through domain-customized incident routing. In *SIGCOMM 2020*, pages 253–269. ACM, 2020. 98

105. Salvador García, Julián Luengo, José Antonio Sáez, Victoria López, and Francisco Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Trans. Knowl. Data Eng.*, 25(4):734–750, 2013. 74

106. Sachin Garg, Aad Van Moorsel, Kalyanaraman Vaidyanathan, and Kishor S Trivedi. A methodology for detection and estimation of software aging. In *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No. 98TB100257)*, pages 283–292. IEEE, 1998. 57

107. Liqiang Geng and Howard J. Hamilton. Interestingness measures for data mining: A survey. *ACM Comput. Surv.*, 38(3):9, 2006. 77

108. Mohammad GhasemiGol and Abbas Ghaemi-Bafghi. A new alert correlation framework based on entropy. In *ICCKE 2013*, pages 184–189. IEEE, 2013. 64

109. Kirk Glerum, Kinshuman Kinshumann, Steve Greenberg, Gabriel Aul, Vince R. Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen C. Hunt. Debugging in the (very) large: ten years of implementation and experience. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles, SOSP*, pages 103–116, 2009. 134

110. Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *journal of Computational and Graphical Statistics*, 24(1):44–65, 2015. 19

111. Henrik Grosskreutz, Stefan Rüping, and Stefan Wrobel. Tight optimistic estimates for fast subgroup discovery. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, volume 5211, pages 440–456. Springer, 2008. 79

112. Henrik Grosskreutz, Bastian Lang, and Daniel Trabold. A relevance criterion for sequential patterns. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I*, pages 369–384, 2013. 18, 74

113. Michael Grottke, Rivalino Matias, and Kishor S Trivedi. The fundamentals of software aging. In *2008 IEEE International conference on software reliability engineering workshops (ISSRE Wksp)*, pages 1–6. Ieee, 2008. 30

114. Riccardo Guidotti, Anna Monreale, and Leonardo Cariaggi. Investigating neighborhood generation methods for explanations of obscure image classifiers. In *PAKDD*, pages 55–68, 2019. 103

115. Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019. 16, 94, 98

116. Riccardo Guidotti, Anna Monreale, Stan Matwin, and Dino Pedreschi. Black box explanation by learning image exemplars in the latent feature space. *ECMLPKDD*, 2020. 103

117. Longfei Han, Senlin Luo, Jianmin Yu, Limin Pan, and Songjing Chen. Rule extraction from support vector machines using ensemble learning approach. *IEEE J-BHI*, 19(2): 728–734, 2015. 94

118. Satoshi Hara and Kohei Hayashi. Making tree ensembles interpretable. In *AISTATS*, pages 77–85, 2018. 94

119. Peter Harremoës. Binomial and poisson distributions as maximum entropy distributions. *IEEE Trans. Information Theory*, 47(5):2039–2041, 2001. 121

120. Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *network and distributed systems security symposium*, 2019. 59

121. Kun He, Fatih Çakir, Sarah Adel Bargal, and Stan Sclaroff. Hashing as tie-aware learning to rank. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4023–4032, 2018. 141, 143

122. Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, pages 207–218. IEEE, 2016. 16, 43

123. Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)*, 54(6):1–37, 2021. 42, 43, 54

124. Zengyou He, Simeng Zhang, and Jun Wu. Significance-based discriminative sequential pattern mining. *Expert Syst. Appl.*, 122:54–64, 2019. 18

125. Francisco Herrera, Cristóbal J. Carmona, Pedro González, and María José del Jesus. An overview on subgroup discovery: foundations and applications. *Knowl. Inf. Syst.*, 29(3):495–525, 2011. 69

126. Lyndon Hiew. *Assisted detection of duplicate bug reports.* PhD thesis, University of British Columbia, 2006. 61

127. Weiming Hu, Yabo Fan, Junliang Xing, Liang Sun, Zhaoquan Cai, and Stephen J. Maybank. Deep constrained siamese hash coding network and load-balanced locality-sensitive hashing for near duplicate image detection. *IEEE Trans. Image Process.*, 27 (9):4452–4464, 2018. 141, 143, 145, 146, 147, 149, 150

128. Gao Huang, Chuan Guo, Matt J. Kusner, Yu Sun, Fei Sha, and Kilian Q. Weinberger. Supervised word mover's distance. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems*, pages 4862–4870, 2016. 158

129. Mark Ibrahim, Melissa Louie, Ceena Modarres, and John W. Paisley. Global explanations of neural networks: Mapping the landscape of predictions. In *AAAI/ACM AIES*, pages 279–287, 2019. 95

130. Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 604–613, 1998. 135, 137, 141, 143

131. Infologic. INFOLOGIC-COPILOTE. 1

132. Tariqul Islam and Dakshnamoorthy Manivannan. Predicting application failure in cloud: A machine learning approach. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pages 24–31. IEEE, 2017. 59

133. Shrainik Jain, Bill Howe, Jiaqi Yan, and Thierry Cruanes. Query2vec: An evaluation of NLP techniques for generalized workload analytics. *arXiv preprint arXiv:1801.05613*, 2018. 48, 82

134. Nicholas Jalbert and Westley Weimer. Automated duplicate detection for bug tracking systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 52–61. IEEE, 2008. 35

135. Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 111–120, 2009. 46

136. Vimalkumar Jeyakumar, Omid Madani, Ali Parandeh, Ashutosh Kulshreshtha, Weifei Zeng, and Navindra Yadav. Explainit!–a declarative root-cause analysis engine for time series data. In *Proceedings of the 2019 International Conference on Management of Data*, pages 333–348, 2019. 13, 44

137. James A Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 273–282, 2005. 53

138. James A Jones, Mary Jean Harrold, and John Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th international conference on Software engineering*, pages 467–477, 2002. 63

139. Xiaolin Ju, Shujuan Jiang, Xiang Chen, Xingya Wang, Yanmei Zhang, and Heling Cao. Hsfal: Effective fault localization using hybrid spectrum of full slices and execution slices. *Journal of Systems and Software*, 90:3–17, 2014. 53

140. Changhee Jung, Sangho Lee, Easwaran Raman, and Santosh Pande. Automated memory leak detection for production use. In Pankaj Jalote, Lionel C. Briand, and André van der Hoek, editors, *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 825–836. ACM, 2014. 48, 114

141. Norio Katayama and Shin'ichi Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 369–380, 1997. 136

142. Branko Kavsek and Nada Lavrac. APRIORI-SD: adapting association rule learning to subgroup discovery. *Appl. Artif. Intell.*, 20(7):543–583, 2006. 79

143. Mehdi Kaytoue, Sergei O. Kuznetsov, Amedeo Napoli, and Sébastien Duplessis. Mining gene expression data with pattern structures in formal concept analysis. *Inf. Sci.*, 181 (10):1989–2001, 2011. 74

144. Mehdi Kaytoue, Marc Plantevit, Albrecht Zimmermann, Ahmed Anes Bendimerad, and Céline Robardet. Exceptional contextual subgraph mining. *Mach. Learn.*, 106(8):1171–1211, 2017. 18, 74

145. Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. 145

146. Randy Kerber. Chimerge: Discretization of numeric attributes. In William R. Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992*, pages 123–128. AAAI Press / The MIT Press, 1992. 75

147. Taghi M Khoshgoftaar and David L Lanning. A neural network approach for early detection of program modules having high risk in the maintenance phase. *Journal of Systems and Software*, 29(1):85–91, 1995. 13

148. Aleksandr Khvorov, Roman Vasiliev, George A. Chernishev, Irving Muller Rodrigues, Dmitrij V. Koznov, and Nikita Povarov. S3M: siamese stack (trace) similarity measure. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR*, pages 266–270, 2021. 53

149. Myunghwan Kim, Roshan Sumbaly, and Sam Shah. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):93–104, 2013. 13

150. Sunghun Kim, Thomas Zimmermann, and Nachiappan Nagappan. Crash graphs: An aggregated view of multiple crashes to improve crash triage. In *Proceedings of the 2011 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN*, pages 486–493, 2011. 48, 134

151. M. Klazar. Twelve countings with rooted plane trees. *European J. Combin.*, 18(2): 195–210, 1997. 125

152. Willi Klösgen. Explora: A multipattern and multistrategy discovery assistant. In *Advances in Knowledge Discovery and Data Mining*, pages 249–271. AAAI/MIT Press, 1996. 69, 70, 77

153. Arno J. Knobbe, Marc de Haas, and Arno Siebes. Propositionalisation and aggregates. In Luc De Raedt and Arno Siebes, editors, *Principles of Data Mining and Knowledge Discovery, 5th European Conference, PKDD 2001, Freiburg, Germany, September 3-5, 2001, Proceedings*, volume 2168 of *Lecture Notes in Computer Science*, pages 277–288. Springer, 2001. 73

154. Josua Krause, Adam Perer, and Kenney Ng. Interacting with predictions: Visual inspection of black-box machine learning models. In *Proceedings of the 2016 CHI conference on human factors in computing systems*, pages 5686–5697, 2016. 19

155. R. Krishnan, G. Sivakumar, and P. Bhattacharya. Extracting decision trees from trained neural networks. *Pattern Recognit.*, 32(12):1999–2009, 1999. 94

156. Gokhan Kul, Duc Luong, Ting Xie, Patrick Coonan, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. Ettu: Analyzing query intents in corporate databases. In *Proceedings of the 25th international conference companion on world wide web*, pages 463–466, 2016. 48

157. Gokhan Kul, Duc Thanh Anh Luong, Ting Xie, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. Similarity metrics for SQL query clustering. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2408–2420, 2018. 82, 85

158. Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 3270–3278, 2015. 135, 141, 143

159. Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM computer communication review*, 34(4):219–230, 2004. 45, 54

160. Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *ACM SIGCOMM computer communication review*, 35(4):217–228, 2005. 45, 54

161. Ahmed Lamkanfi, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. Comparing mining algorithms for predicting the severity of a reported bug. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 249–258. IEEE, 2011. 35

162. Nada Lavrac, Bojan Cestnik, Dragan Gamberger, and Peter A. Flach. Decision support through subgroup discovery: Three case studies and the lessons learned. *Mach. Learn.*, 57(1-2):115–143, 2004. 68

163. Nada Lavrac, Branko Kavsek, Peter A. Flach, and Ljupco Todorovski. Subgroup discovery with CN2-SD. *J. Mach. Learn. Res.*, 5:153–188, 2004. 78, 79, 128

164. Sangho Lee, Changhee Jung, and Santosh Pande. Detecting memory leaks through introspective dynamic behavior modelling using machine learning. In *Proceedings of the 36th International Conference on Software Engineering*, pages 814–824, 2014. 48

165. Sun-Ro Lee, Min-Jae Heo, Chan-Gun Lee, Milhan Kim, and Gaeul Jeong. Applying deep learning based automatic bug triager to industrial projects. In *ESEC/FSE 2017*, pages 926–931. ACM, 2017. 46, 60, 94

166. Wan-Jui Lee. Anomaly detection and severity prediction of air leakage in train braking pipes. *International Journal of Prognostics and Health Management*, 8(3), 2017. 52

167. Dennis Leman, Ad Feelders, and Arno J. Knobbe. Exceptional model mining. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II*, volume 5212 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008. 75

168. Florian Lemmerich. *Novel Techniques for Efficient and Effective Subgroup Discovery.* PhD thesis, Julius Maximilians University Würzburg, Germany, 2014. 73, 78

169. Florian Lemmerich and Martin Becker. pysubgroup: Easy-to-use subgroup discovery in python. In *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, volume 11053, pages 658–662. Springer, 2018. 86, 127

170. Florian Lemmerich, Martin Atzmueller, and Frank Puppe. Fast exhaustive subgroup discovery with numerical target concepts. *Data Min. Knowl. Discov.*, 30(3):711–762, 2016. 78, 79, 80

171. Johannes Lerch and Mira Mezini. Finding duplicates of your yet unwritten bug report. In *17th European Conference on Software Maintenance and Reengineering, CSMR*, pages 69–78, 2013. 48, 61, 134, 143, 144

172. Anna Levin, Shelly Garion, Elliot K Kolodner, Dean H Lorenz, Katherine Barabash, Mike Kugler, and Niall McShane. Aiops for a cloud object storage service. In *2019 IEEE International Congress on Big Data (BigDataCongress)*, pages 165–169. IEEE, 2019. 13, 14

173. Jian Li, Pinjia He, Jieming Zhu, and Michael R Lyu. Software defect prediction via convolutional neural network. In *2017 IEEE international conference on software quality, reliability and security (QRS)*, pages 318–328. IEEE, 2017. 34, 57

174. Jing Li, Rebecca J Stones, Gang Wang, Zhongwei Li, Xiaoguang Liu, and Kang Xiao. Being accurate is not enough: New metrics for disk failure prediction. In *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pages 71–80. IEEE, 2016. 13, 28, 44

175. Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. Causal inference-based root cause analysis for online service systems with intervention recognition. *arXiv preprint arXiv:2206.05871*, 2022. 13

176. Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. Deep supervised discrete hashing. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 2482–2491, 2017. 135

177. Yangguang Li, Zhen Ming Jiang, Heng Li, Ahmed E Hassan, Cheng He, Ruirui Huang, Zhengda Zeng, Mian Wang, and Pinan Chen. Predicting node failures in an ultra-large-scale cloud computing platform: an aiops solution. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(2):1–24, 2020. 2, 13, 36, 42

178. Ze Li and Yingnong Dang. Aiops: Challenges and experiences in azure. *USENIX Association, Santa Clara*, pages 51–53, 2019. 13

179. Zeyan Li, Chengyang Luo, Yiwei Zhao, Yongqian Sun, Kaixin Sui, Xiping Wang, Dapeng Liu, Xing Jin, Qi Wang, and Dan Pei. Generic and robust localization of multi-dimensional root causes. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 47–57. IEEE, 2019. 62, 68

180. Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. Actionable and interpretable fault localization for recurring failures in online service systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 996–1008, 2022. 29

181. Zhiguo Li and Qing He. Prediction of railcar remaining useful life by multiple data source fusion. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):2226–2235, 2015. 52

182. Zhihan Li, Youjian Zhao, Rong Liu, and Dan Pei. Robust and rapid clustering of kpis for large-scale anomaly detection. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2018. 44

183. Meng-Hui Lim, Jian-Guang Lou, Hongyu Zhang, Qiang Fu, Andrew Beng Jin Teoh, Qingwei Lin, Rui Ding, and Dongmei Zhang. Identifying recurrent and unknown performance issues. In *2014 IEEE International Conference on Data Mining*, pages 320–329. IEEE, 2014. 44, 60

184. Fan Lin, Matt Beadon, Harish Dattatraya Dixit, Gautham Vunnam, Amol Desai, and Sriram Sankar. Hardware remediation at scale. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 14–17. IEEE, 2018. 13, 65

185. Fred Lin, Keyur Muzumdar, Nikolay Pavlovich Laptev, Mihai-Valentin Curelea, Seunghak Lee, and Sriram Sankar. Fast dimensional analysis for root cause investigation in a large-scale service environment. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(2):1–23, 2020. 62

186. Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. Deep learning of binary hash codes for fast image retrieval. In *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2015, Boston, MA, USA, June 7-12, 2015*, pages 27–35. IEEE Computer Society, 2015. 143

187. Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, and Dongmei Zhang. idice: problem identification for emerging issues. In *Proceedings of the 38th International Conference on Software Engineering*, pages 214–224, 2016. 13, 42

188. Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 480–490, 2018. 42

189. Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 2475–2483, 2015. 135, 143

190. Chao Liu, Long Fei, Xifeng Yan, Jiawei Han, and Samuel P Midkiff. Statistical debugging: A hypothesis testing-based approach. *IEEE Transactions on software engineering*, 32(10):831–848, 2006. 53, 63

191. Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Deep supervised hashing for fast image retrieval. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 2064–2072, 2016. 135, 141, 143

192. Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. What bugs cause production cloud incidents? In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 155–162, 2019. 13

193. Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. Mining invariants from console logs for system problem detection. In *USENIX annual technical conference*, pages 1–14, 2010. 43, 63

194. Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 475–485. IEEE, 2013. 36

195. Sidi Lu, Bing Luo, Tirthak Patel, Yongtao Yao, Devesh Tiwari, and Weisong Shi. Making disk failure predictions smarter! In *FAST*, pages 151–167, 2020. 44

196. Lucene. Lucene apache. 61, 143

197. José María Luna, José Raúl Romero, Cristóbal Romero, and Sebastián Ventura. Discovering subgroups by means of genetic programming. In Krzysztof Krawiec, Alberto Moraglio, Ting Hu, A. Sima Etaner-Uyar, and Bin Hu, editors, *Genetic Programming - 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings*, volume 7831 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2013. 79

198. Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *(NIPS 2017*, pages 4765–4774, 2017. 19, 94, 111

199. Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. Correlating events with time series for incident diagnosis. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1583–1592, 2014. 13, 64

200. Xiao Luo, Chong Chen, Huasong Zhong, Hao Zhang, Minghua Deng, Jianqiang Huang, and Xiansheng Hua. A survey on deep hashing methods. *CoRR*, abs/2003.03369, 2020. 134

201. Yingzhe Lyu, Heng Li, Mohammed Sayagh, Zhen Ming Jiang, and Ahmed E Hassan. An empirical study of the impact of data splitting decisions on the performance of aiops solutions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30 (4):1–38, 2021. 38

202. Yingzhe Lyu, Gopi Krishnan Rajbahadur, Dayi Lin, Boyuan Chen, and Zhen Ming Jiang. Towards a consistent interpretation of aiops models. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(1):1–38, 2021. 2, 16, 19, 36, 37

203. Meng Ma and Zhu Mao. Deep-convolution-based lstm network for remaining useful life prediction. *IEEE Transactions on Industrial Informatics*, 17(3):1658–1667, 2020. 58

204. Adetokunbo Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. A lightweight algorithm for message type extraction in system application logs. *IEEE Transactions on Knowledge and Data Engineering*, 24(11):1921–1936, 2011. 43

205. Vitor Hirota Makiyama, Jordan Raddick, and Rafael DC Santos. Text mining applied to SQL queries: A case study for the sdss skyserver. In *SIMBig*, pages 66–72, 2015. 84, 85

206. Romain Mathonat, Diana Nurbakova, Jean-François Boulicaut, and Mehdi Kaytoue. Seqscout: Using a bandit model to discover interesting subgroups in labeled sequences. In *2019 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2019, Washington, DC, USA, October 5-8, 2019*, pages 81–90. IEEE, 2019. 18, 74

207. Evan K Maxwell, Godmar Back, and Naren Ramakrishnan. Diagnosing memory leaks using graph mining on heap dumps. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 115–124, 2010. 48

208. Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976. 41

209. Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, volume 19, pages 4739–4745, 2019. 43, 56

210. Tim Menzies. The five laws of se for ai. *IEEE Software*, 37(1):81–85, 2019. 14

211. Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, 33(1):2–13, 2006. 56

212. Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference on Program Comprehension*, pages 167–177, 2018. 43

213. Justin Meza, Qiang Wu, Sanjev Kumar, and Onur Mutlu. A large-scale study of flash memory failures in the field. *ACM SIGMETRICS Performance Evaluation Review*, 43 (1):177–190, 2015. 57

214. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 46

215. Seyed Ali Mirheidari, Sajjad Arshad, and Rasool Jalili. Alert correlation algorithms: A survey and taxonomy. In *Cyberspace Safety and Security: 5th International Symposium, CSS 2013, Zhangjiajie, China, November 13-15, 2013, Proceedings 5*, pages 183–197. Springer, 2013. 47, 53, 54

216. Natwar Modani, Rajeev Gupta, Guy M. Lohman, Tanveer Fathima Syeda-Mahmood, and Laurent Mignet. Automatically identifying known software problems. In *Proceedings of the 23rd International Conference on Data Engineering Workshops*, pages 433–441. IEEE Computer Society, 2007. 61, 144

217. Christoph Molnar. *Interpretable Machine Learning*. 2019. 16, 100, 104

218. Katharina Morik, Jean-François Boulicaut, and Arno Siebes, editors. *Local Pattern Detection, International Seminar, Dagstuhl Castle, Germany, April 12-16, 2004, Revised Selected Papers*, volume 3539 of *Lecture Notes in Computer Science*, 2005. Springer. 69

219. Akira Moroo, Akiko Aizawa, and Takayuki Hamamoto. Reranking-based crash report deduplication. In Xudong He, editor, *The 29th International Conference on Software Engineering and Knowledge Engineering*, pages 507–510. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2017. 61, 134, 144

220. Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering*, pages 181–190, 2008. 56

221. Mozilla. Moz SQL parser. 84

222. Mozilla. Mozilla crash reporter, 2012. 134

223. Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461, 2006. 56

224. Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *2013 35th international conference on software engineering (ICSE)*, pages 382–391. IEEE, 2013. 56

225. Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B Dasgupta, and Subhrajit Bhattacharya. Anomaly detection using program control flow graph mining from execution logs. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 215–224, 2016. 13, 29, 43

226. Roberto Natella, Domenico Cotroneo, Joao A Duraes, and Henrique S Madeira. On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering*, 39(1):80–96, 2012. 34

227. Sasho Nedelkoski, Jasmin Bogatinovski, Ajay Kumar Mandapati, Soeren Becker, Jorge Cardoso, and Odej Kao. Multi-source distributed system data for ai-powered analytics. In *Service-Oriented and Cloud Computing: 8th IFIP WG 2.14 European Conference, ESOCC 2020, Heraklion, Crete, Greece, September 28–30, 2020, Proceedings 8*, pages 161–176. Springer, 2020. 38

228. Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970. 61, 144

229. S Nessa, M Abedin, W Eric Wong, L Khan, and Y Qi. Fault localization using n-gram analysis. In *Proceedings of the 3rd International Conference on Wireless Algorithms, Systems, and Applications*, pages 548–559, 2009. 41

230. Hiep Nguyen, Zhiming Shen, Yongmin Tan, and Xiaohui Gu. Fchain: Toward black-box online fault localization for cloud systems. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 21–30. IEEE, 2013. 62

231. Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. Using of jaccard coefficient for keywords similarity. In *Proceedings of the international multiconference of engineers and computer scientists*, volume 1, pages 380–384, 2013. 80

232. Paolo Notaro, Jorge Cardoso, and Michael Gerndt. A survey of aiops methods for failure management. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12(6): 1–45, 2021. 2, 13, 14, 16, 31, 33, 36, 54

233. Petra Kralj Novak, Nada Lavrac, and Geoffrey I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *J. Mach. Learn. Res.*, 10:377–403, 2009. 17, 74

234. Oracle help center. Active session history. 85

235. Celestino Ordóñez, Fernando Sánchez Lasheras, Javier Roca-Pardiñas, and Francisco Javier de Cos Juez. A hybrid arima–svm model for the study of the remaining useful life of aircraft engines. *Journal of Computational and Applied Mathematics*, 346: 184–191, 2019. 52

236. Thomas J Ostrand, Elaine J Weyuker, and Robert M Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005. 56

237. Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010. 38

238. Phuong Pham, Vivek Jain, Lukas Dauterman, Justin Ormont, and Navendu Jain. Deeptriage: Automated transfer assistance for incidents in cloud services. In *ACM SIGKDD*, pages 3281–3289, 2020. 46, 60, 94, 98

239. Barbara FI Pieters, Arno Knobbe, and Sašo Dzeroski. Subgroup discovery in ranked data, with an application to gene set enrichment. In *Proceedings preference learning workshop (PL 2010) at ECML PKDD*, volume 10, pages 1–18, 2010. 78

240. Natthakul Pingclasai, Hideaki Hata, and Ken-ichi Matsumoto. Classifying bug reports to bugs and other requests using topic modeling. In *2013 20Th asia-pacific software engineering conference (APSEC)*, volume 2, pages 13–18. IEEE, 2013. 35

241. Teerat Pitakrat, Dušan Okanović, André van Hoorn, and Lars Grunske. Hora: Architecture-aware online failure prediction. *Journal of Systems and Software*, 137: 669–685, 2018. 13, 28, 58

242. Pankaj Prasad and Charley Rich. Market guide for aiops platforms. *Retrieved March*, 12:2020, 2018. 2, 9, 10, 18, 28, 32, 54

243. Hugo Manuel Proença, Peter Grünwald, Thomas Bäck, and Matthijs van Leeuwen. Discovering outstanding subgroup lists for numeric targets using MDL. In Frank Hutter, Kristian Kersting, Jefrey Lijffijt, and Isabel Valera, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part I*, volume 12457 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2020. 79

244. Xianping Qu and Jingjing Ha. Next generation of devops: Aiops in practice@ baidu. *SREcon17*, 2017. 13

245. Baishakhi Ray, Vincent Hellendoorn, Saheel Godhane, Zhaopeng Tu, Alberto Bacchelli, and Premkumar Devanbu. On the" naturalness" of buggy code. In *Proceedings of the 38th International Conference on Software Engineering*, pages 428–439, 2016. 14

246. Lena Reiter and FH Wedel. Aiops–a systematic literature review. 2021. 37

247. Youcef Remil, Anes Bendimerad, Romain Mathonat, Philippe Chaleat, and Mehdi Kaytoue. "what makes my queries slow?": Subgroup discovery for SQL workload analysis. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*, pages 642–652. IEEE, 2021. 18, 21, 42

248. Youcef Remil, Anes Bendimerad, Marc Plantevit, Céline Robardet, and Mehdi Kaytoue. Interpretable summaries of black box incident triaging with subgroup discovery. In *8th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2021, Porto, Portugal, October 6-9, 2021*, pages 1–10. IEEE, 2021. 22, 42

249. Youcef Remil, Anes Bendimerad, Mathieu Chambard, Romain Mathonat, Marc Plantevit, and Mehdi Kaytoue. Subjectively interesting subgroups with hierarchical targets: Application to java memory analysis. In *International Conference on Data Mining Workshops, ICDMW (Accepted)*. IEEE, 2023. 23, 48

250. Youcef Remil, Anes Bendimerad, Romain Mathonat, and Mehdi Kaytoue. Aiops solutions for incident management: Technical guidelines and a comprehensive literature review. *ACM Transactions on Software Engineering and Methodology (TOSEM) (under submission)*, 2023. 21, 54

251. Youcef Remil, Anes Bendimerad, Romain Mathonat, Chedy Raissi, and Mehdi Kaytoue. Deeplsh: Deep locality-sensitive hash learning for fast and efficient near-duplicate crash report detection. In *46th International Conference on Software Engineering, ICSE (Accepted)*, 2024. 23, 48

252. Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3009–3017, 2019. 55

253. Manos Renieres and Steven P Reiss. Fault localization with nearest neighbor queries. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, pages 30–39. IEEE, 2003. 29, 53, 62

254. Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *ACM SIGKDD*, pages 1135–1144, 2016. 19, 94, 95, 99, 103, 107, 111

255. Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 19, 94

256. Laxmi Rijal, Ricardo Colomo-Palacios, and Mary Sánchez-Gordón. Aiops: A multivocal literature review. *Artificial Intelligence for Cloud and Edge Computing*, pages 31–50, 2022. 2, 16, 32, 54

257. Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *29th International Conference on Software Engineering (ICSE'07)*, pages 499–510. IEEE, 2007. 61

258. Korosh Koochekian Sabor, Abdelwahab Hamou-Lhadj, and Alf Larsson. DURFEX: A feature extraction technique for efficient detection of duplicate bug reports. In *2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017*, pages 240–250, 2017. 48, 61, 134, 143, 144

259. Caitlin Sadowski and Greg Levin. Simhash: Hash-based similarity detection, 2007. 20, 143

260. Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):1–42, 2010. 16, 29, 51, 54

261. Areeg Samir and Claus Pahl. A controller architecture for anomaly detection, root cause analysis and self-adaptation for cluster architectures. In *Intl Conf Adaptive and Self-Adaptive Systems and Applications*, 2019. 63

262. Mark Sanderson. Test collection based evaluation of information retrieval systems. *Found. Trends Inf. Retr.*, 4(4):247–375, 2010. 53, 145

263. Tobias Scheffer and Stefan Wrobel. Finding the most interesting patterns in a database quickly by using sequential sampling. *J. Mach. Learn. Res.*, 3:833–862, 2002. 77

264. Adrian Schröter, Nicolas Bettenburg, and Rahul Premraj. Do stack traces help developers fix bugs? In *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE)*, pages 118–121, 2010. 136

265. Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 19

266. Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. e-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference*, pages 3215–3222, 2019. 44

267. Bikash Sharma, Praveen Jayachandran, Akshat Verma, and Chita R Das. Cloudpd: Problem determination and diagnosis in shared dynamic clouds. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013. 55

268. Shijun Shen, Jiuling Zhang, Daochao Huang, and Jun Xiao. Evolving from traditional systems to aiops: Design, implementation and measurements. In *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pages 276–280. IEEE, 2020. 2, 9, 10

269. Akbar Siami Namin, James H Andrews, and Duncan J Murdoch. Sufficient mutation operators for measuring test effectiveness. In *Proceedings of the 30th international conference on Software engineering*, pages 351–360, 2008. 34, 52

270. Arno Siebes. Data surveying: Foundations of an inductive query language. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95), Montreal, Canada, August 20-21, 1995*, pages 269–274. AAAI Press, 1995. 69, 71

271. Abraham Silberschatz and Alexander Tuzhilin. On subjective measures of interestingness in knowledge discovery. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95), Montreal, Canada, August 20-21, 1995*, pages 275–281. AAAI Press, 1995. 77, 82

272. Vladimir Sor and Satish Narayana Srirama. Memory leak detection in java: Taxonomy and classification of approaches. *J. Syst. Softw.*, 96:139–151, 2014. 54, 114

273. Charles Spearman. The proof and measurement of association between two things. 1961. 53

274. Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 407–419, 1995. 74

275. Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2828–2837, 2019. 55

276. Ya Su, Youjian Zhao, Wentao Xia, Rong Liu, Jiahao Bu, Jing Zhu, Yuanpu Cao, Haibin Li, Chenhao Niu, Yiyin Zhang, et al. Coflux: robustly correlating kpis by fluctuations for service troubleshooting. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10, 2019. 64

277. Yongqian Sun, Youjian Zhao, Ya Su, Dapeng Liu, Xiaohui Nie, Yuan Meng, Shiwen Cheng, Dan Pei, Shenglin Zhang, Xianping Qu, et al. Hotspot: Anomaly localization for additive kpis with multi-dimensional attributes. *IEEE Access*, 6:10909–10923, 2018. 62

278. Ashish Sureka and Pankaj Jalote. Detecting duplicate bug report using character n-gram-based features. In *2010 Asia Pacific software engineering conference*, pages 366–374. IEEE, 2010. 61

279. Zhiyuan Tan, Aruna Jamdagni, Xiangjian He, Priyadarsi Nanda, and Ren Ping Liu. A system for denial-of-service attack detection based on multivariate correlation analysis. *IEEE transactions on parallel and distributed systems*, 25(2):447–456, 2013. 64

280. Liang Tang, Tao Li, and Chang-Shing Perng. Logsig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 785–794, 2011. 43

281. Yuan Tian, David Lo, and Chengnian Sun. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *2012 19th Working Conference on Reverse Engineering*, pages 215–224. IEEE, 2012. 35

282. Yuan Tian, David Lo, and Chengnian Sun. Drone: Predicting priority of reported bugs by multi-factor analysis. In *2013 IEEE International Conference on Software Maintenance*, pages 200–209. IEEE, 2013. 59

283. Quoc Trung Tran, Konstantinos Morfonios, and Neoklis Polyzotis. Oracle workload intelligence. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1669–1681. ACM, 2015. 82

284. Risto Vaarandi. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*, pages 119–126. Ieee, 2003. 43

285. Kalyanaraman Vaidyanathan and Kishor S Trivedi. A measurement-based model for estimation of resource exhaustion in operational software systems. In *Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No. PR00443)*, pages 84–93. IEEE, 1999. 34, 57

286. Matthijs van Leeuwen and Arno J. Knobbe. Diverse subgroup set discovery. *Data Min. Knowl. Discov.*, 25(2):208–242, 2012. 81, 128

287. Matthijs van Leeuwen and Antti Ukkonen. Discovering skylines of subgroup sets. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Zelezný, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*, volume 8190 of *Lecture Notes in Computer Science*, pages 272–287. Springer, 2013. 79

288. Matthijs van Leeuwen, Tijl De Bie, Eirini Spyropoulou, and Cédric Mesnage. Subjective interestingness of subgraph patterns. *Mach. Learn.*, 105(1):41–75, 2016. 18

289. Roman Vasiliev, Dmitrij V. Koznov, George A. Chernishev, Aleksandr Khvorov, Dmitry V. Luciv, and Nikita Povarov. Tracesim: a method for calculating stack trace similarity. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, FSE 2020*, pages 25–30, 2020. 53, 61, 134, 136, 144

290. Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016. 38

291. Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204, 2010. 57

292. Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014. 20, 134, 137

293. Qing Wang, Wubai Zhou, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Constructing the knowledge base for cognitive it service management. In *2017 IEEE International Conference on Services Computing (SCC)*, pages 410–417. IEEE, 2017. 65

294. Song Wang, Taiyue Liu, and Lin Tan. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering*, pages 297–308, 2016. 41

295. Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks.

In *2017 IEEE international conference on intelligence and security informatics (ISI)*, pages 43–48. IEEE, 2017. 45, 56

296. Xiaofang Wang, Yi Shi, and Kris M. Kitani. Deep supervised hashing with triplet labels. In *Computer Vision - ACCV 2016 - 13th Asian Conference on Computer Vision*, pages 70–84, 2016. 135, 143

297. Xuanrun Wang, Kanglin Yin, Qianyu Ouyang, Xidao Wen, Shenglin Zhang, Wenchi Zhang, Li Cao, Jiuxue Han, Xing Jin, and Dan Pei. Identifying erroneous software changes through self-supervised contrastive learning on time series data. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 366–377. IEEE, 2022. 13, 29, 44

298. Yu Wang, Qiang Miao, Eden WM Ma, Kwok-Leung Tsui, and Michael G Pecht. Online anomaly detection for hard disk drives based on mahalanobis distance. *IEEE Transactions on Reliability*, 62(1):136–145, 2013. 57

299. Markus Weninger, Elias Gander, and Hanspeter Mössenböck. Analyzing data structure growth over time to facilitate memory leak detection. In Varsha Apte, Antinisca Di Marco, Marin Litoiu, and José Merseguer, editors, *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, ICPE 2019, Mumbai, India, April 7-11, 2019*, pages 273–284. ACM, 2019. 114

300. Frank Wilcoxon. *Individual comparisons by ranking methods*. Springer, 1992. 53

301. W Eric Wong, Vidroha Debroy, and Dianxiang Xu. Towards better fault localization: A crosstab-based statistical approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):378–396, 2011. 63

302. W Eric Wong, Vidroha Debroy, Ruizhi Gao, and Yihao Li. The dstar method for effective software fault localization. *IEEE Transactions on Reliability*, 63(1):290–308, 2013. 53, 63

303. W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016. 35, 54

304. Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In *Principles of Data Mining and Knowledge Discovery, First European Symposium, PKDD '97, Trondheim, Norway, June 24-27, 1997, Proceedings*, volume 1263, pages 78–87. Springer, 1997. 17, 68, 69

305. Stefan Wrobel. Inductive logic programming for knowledge discovery in databases. In *Relational data mining*, pages 74–101. Springer, 2001. 72

306. Di Wu, Yiping Ke, Jeffrey Xu Yu, Philip S Yu, and Lei Chen. Detecting leaders from correlated time series. In *Database Systems for Advanced Applications: 15th International Conference, DASFAA 2010, Tsukuba, Japan, April 1-4, 2010, Proceedings, Part I 15*, pages 352–367. Springer, 2010. 13, 64

307. Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. Crashlocator: locating crashing faults based on crash stacks. In *International Symposium on Software Testing and Analysis, ISSTA*, pages 204–214, 2014. 29

308. Yuting Wu, Mei Yuan, Shaopeng Dong, Li Lin, and Yingqi Liu. Remaining useful life estimation of engineered systems using vanilla lstm neural networks. *Neurocomputing*, 275:167–179, 2018. 58

309. Sheng-Qu Xi, Yuan Yao, Xu-Sheng Xiao, Feng Xu, and Jian Lv. Bug triaging based on tossing sequence modeling. *Journal of Computer Science and Technology*, 34:942–956, 2019. 13, 46, 60, 94

310. Xin Xia, David Lo, Ying Ding, Jafar M Al-Kofahi, Tien N Nguyen, and Xinyu Wang. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297, 2016. 60

311. Zhihua Xia, Xinhui Wang, Liangao Zhang, Zhan Qin, Xingming Sun, and Kui Ren. A privacy-preserving and copy-deterrence content-based image retrieval scheme in cloud computing. *IEEE transactions on information forensics and security*, 11(11):2594–2608, 2016. 20

312. Jiang Xiao, Zhuang Xiong, Song Wu, Yusheng Yi, Hai Jin, and Kan Hu. Disk failure prediction in data centers via online learning. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018. 58

313. Yichen Xie and Alexander Aiken. Context- and path-sensitive memory leak detection. In Michel Wermelinger and Harald C. Gall, editors, *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, Lisbon, Portugal, September 5-9, 2005*, pages 115–125. ACM, 2005. 114

314. Zhe Xie, Haowen Xu, Wenxiao Chen, Wanxue Li, Huai Jiang, Liangfei Su, Hanzhang Wang, and Dan Pei. Unsupervised anomaly detection on microservice traces through graph vae. In *Proceedings of the ACM Web Conference 2023*, pages 2874–2884, 2023. 55

315. Chang Xu, Gang Wang, Xiaoguang Liu, Dongdong Guo, and Tie-Yan Liu. Health status assessment and failure prediction for hard drives with recurrent neural networks. *IEEE Transactions on Computers*, 65(11):3502–3508, 2016. 13, 57

316. Guoqing Xu and Atanas Rountev. Precise memory leak detection for java software using container profiling. *ACM Trans. Softw. Eng. Methodol.*, 22(3):17:1–17:28, 2013. 48

317. Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 world wide web conference*, pages 187–196, 2018. 55

318. Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132, 2009. 43, 55

319. Jifeng Xuan, He Jiang, Zhilei Ren, Jun Yan, and Zhongxuan Luo. Automatic bug triage using semi-supervised text classification. In *SEKE*, pages 209–214, 2010. 13, 60

320. Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo, and Xindong Wu. Towards effective bug triage with software data reduction techniques. *IEEE transactions on knowledge and data engineering*, 27(1):264–280, 2014. 13, 45, 46

321. Geunseok Yang, Tao Zhang, and Byungjeong Lee. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 97–106. IEEE, 2014. 61

322. Zunwen You, Zengchang Qin, and Zheng Zheng. Statistical fault localization using execution sequence. In *2012 International Conference on Machine Learning and Cybernetics*, volume 3, pages 899–905. IEEE, 2012. 41

323. Philip S. Yu, Ming-Syan Chen, Hans-Ulrich Heiss, and Sukho Lee. On workload characterization of relational database environments. *IEEE Trans. Software Eng.*, 18(4): 347–355, 1992. 82

324. Yi Yu, Roger Zimmermann, Ye Wang, and Vincent Oria. Scalable content-based music retrieval using chord progression histogram and tree-structure lsh. *IEEE Transactions on Multimedia*, 15(8):1969–1981, 2013. 20

325. Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V Chawla. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1409–1416, 2019. 55

326. Jie Zhang, Xiaoyin Wang, Dan Hao, Bing Xie, Lu Zhang, and Hong Mei. A survey on bug-report analysis. *Sci. China Inf. Sci.*, 58(2):1–24, 2015. 33, 45, 46, 54

327. Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. Automated it system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1291–1300. IEEE, 2016. 13, 28

328. Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2017. 58

329. Tao Zhang, He Jiang, Xiapu Luo, and Alvin TS Chan. A literature review of research in bug resolution: Tasks, challenges and future directions. *The Computer Journal*, 59 (5):741–773, 2016. 12, 35

330. Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 807–817, 2019. 56

331. Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. Efficient training of very deep neural networks for supervised hashing. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1487–1495. IEEE Computer Society, 2016. 143

332. Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, et al. Understanding and handling alert storm for online service systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, pages 162–171, 2020. 47, 60

333. Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, et al. Real-time incident prediction for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 315–326, 2020. 13, 29

334. Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. Automatically and adaptively identifying severe alerts for online service systems. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2420–2429. IEEE, 2020. 47, 59

335. Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, et al. An empirical investigation of practical log anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1404–1415, 2021. 36, 54

336. Ying Zhao, Xiang Liu, Siqing Gan, and Weimin Zheng. Predicting disk failures with hmm-and hsmm-based approaches. In *Advances in Data Mining. Applications and Theoretical Aspects: 10th Industrial Conference, ICDM 2010, Berlin, Germany, July 12-14, 2010. Proceedings 10*, pages 390–404. Springer, 2010. 57

337. Shuai Zheng, Kosta Ristovski, Ahmed Farahat, and Chetan Gupta. Long short-term memory network for remaining useful life estimation. In *2017 IEEE international conference on prognostics and health management (ICPHM)*, pages 88–95. IEEE, 2017. 13, 28, 44, 58

338. Wubai Zhou, Liang Tang, Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Resolution recommendation for event tickets in service management. *IEEE Transactions on Network and Service Management*, 13(4):954–967, 2016. 64

339. Alexander Zien, Nicole Krämer, Sören Sonnenburg, and Gunnar Rätsch. The feature importance ranking measure. *arXiv preprint arXiv:0906.4258*, 2009. 94

340. Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005. 99

FOLIO ADMINISTRATIF

THESE DE L'INSA LYON, MEMBRE DE L'UNIVERSITE DE LYON

NOM : REMIL DATE de SOUTENANCE : 06/10/2023
Prénoms : Youcef

TITRE : **A Data Mining Perspective on Explainable AIOps with Applications to Software Maintenance**.
NATURE : Doctorat Numéro d'ordre : 2023ISAL0072
Ecole doctorale : InfoMaths (ED 512)

Spécialité : Informatique

RESUME : La supervision des systèmes informatiques modernes présente de nouveaux défis en termes de scalabilité, de fiabilité et d'efficacité. Les méthodes traditionnelles de maintenance basées sur l'exécution de tâches manuelles ont prouvé leur inefficacité. De manière similaire, les systèmes experts à base de règles sont limités dans leur capacité actuelle à gérer et anticiper le grand nombre d'alertes générées par les systèmes informatiques. AIOps for Operating Systems (AIOps) propose d'utiliser des techniques avancées d'apprentissage automatique centrées sur la donnée pour améliorer et automatiser les systèmes de supervision. Cependant, il existe plusieurs défis à relever pour concrétiser cette vision, qui sont partagés à la fois par la communauté scientifique et les ingénieurs sur le terrain. Tout d'abord, le manque d'une terminologie claire et unifiée dans le domaine de l'AIOps rend difficile la progression, l'implémentation et la comparaison des contributions provenant de différentes disciplines. De plus, les exigences et les métriques nécessaires à la construction de modèles AIOps, alignés avec les contraintes industrielles, ne sont pas suffisamment élaborées. Deuxièmement, les contributions théoriques en matière d'AIOps se sont principalement concentrées sur les modèles prédictifs pour la détection et prédictions des incidents, en négligeant souvent la capacité des modèles descriptifs à gérer et résoudre les défis liés à la qualité, à la complexité, au volume et à la diversité des données. Troisièmement, la dépendance excessive aux modèles boîte noire opaques limite leur adoption par les praticiens de l'industrie. Enfin, les solutions AIOps existantes ne prêtent pas toujours suffisamment d'importance à l'évaluation des performances des modèles et aux problèmes de scalabilité lors du développement et de l'évaluation des modèles.

Nous proposons d'abord une approche systématique de l'AIOps qui organise les connaissances dans ce domaine de recherche en fournissant une catégorisation en accord avec les normes et les exigences de l'industrie. Deuxièmement, nous explorons l'application de la découverte de sous-groupes qui est une technique prometteuse de fouille de données qui permet l'extraction d'hypothèses intéressantes à partir de vastes ensembles de données diversifiées. Ainsi, les utilisateurs sont en mesure de comprendre, d'interagir avec et d'interpréter les processus sous-jacents aux modèles. Nos contributions dans ce domaine comprennent une application pratique axée sur l'identification de fragments de requêtes SQL suspects permettant de localiser les problèmes de dégradation de performances. De plus, nous développons un mécanisme d'interprétation pour les modèles de triage des incidents, offrant des explications contextualisées pour les décisions prises par le modèle. Nous abordons également le problème de l'analyse des problématiques de saturation de la mémoire Java, caractérisé par un ensemble de données volumineux et complexes intégrant des données hiérarchiques. Enfin, nous traitons la scalabilité en étudiant le problème de la détection de la déduplication des incidents. Notre objectif est de rechercher de manière efficace et rapide les rapports de plantage les plus similaires en combinant des techniques de hachage sensibles à la localité (LSH) et des techniques d'apprentissage contrastif dans un cadre unifié. Pour garantir la pertinence et la praticité de nos propositions, ce projet de thèse implique une collaboration entre des chercheurs en fouille de données et des praticiens d'Infologic, un éditeur de logiciels français.

MOTS-CLÉS : AIOps, Procédure de gestion des incidents, Fouilles de données, Découverte de sous-groupes, IA Explicable, Hachage sensible à la localité.

Laboratoire (s) de recherche : Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS)
Directeur de thèse : Pr. Jean-François Boulicaut (INSA Lyon)
HDR. Mehdi Kaytoue (INSA Lyon, Infologic)
Dr. Anes Bendimerad (Infologic)

Président de jury : Pr. Christel Vrain (Université d'Orléans)

Composition du jury :
Pr. Romain Robbes (LaBRI, Université de Bordeaux) – Rapporteur
Pr. Alexandre Termier (Université Rennes) – Rapporteur
Pr. Arnaud Soulet (Université de Tours) – Examinateur
HDR. Peggy Cellier (INSA Rennes) - Examinatrice