Distributed Anomaly Detection on Large Knowledge Graphs

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

von
Farshad Bakhshandegan Moghaddam
aus
Zanjan, Iran

Bonn, 2024

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

Gutachter/Betreuer:	Prof. Dr. Jens Lehmann
Gutachter:	Prof. Dr. Christian Bauckhage

Tag der Promotion:	16.10.2024
Erscheinungsjahr:	2024

Abstract

Digitization has yielded vast data, known as Big Data, fostering data analysis. As this data comes from various sources and is of diverse types, data integration techniques become essential in making analytics more accessible and effective. Knowledge Graphs (KGs) are vital in linking diverse data within a directed multi-graph, utilizing unique resource identifiers. Presently, over 10,000 datasets conform to Semantic Web standards, spanning fields like life sciences, industries, and the Internet of Things. KGs employ various creation approaches, including crowd-sourcing, natural language processing, and knowledge-extraction tools. However, the data used as input is often unvalidated and not cross-checked, making KGs vulnerable to errors at both logical and semantic levels. These errors can manifest across individual triples, impacting the subject, predicate, or object components of the RDF (Resource Description Framework) data, or even happen in relationships across triples, compromising the overall quality of KGs. Detecting these errors is not a trivial task because of the complex structure and the sheer size of modern large-scale KG data which easily surpasses the available memory capacity of current computers (e.g. English DBpedia size is ~ 114 GB). Furthermore, in the majority of cases, there are no defined rules to determine whether entered data is deemed correct or incorrect. The primary objective of this thesis is to identify errors in very large knowledge graphs in a scalable manner without prior knowledge of ground truth. To achieve this, we employ Anomaly Detection (AD), a branch of data mining, to identify errors in KGs. However, most of the traditional AD algorithms are no longer directly applicable to KGs due to the scalability issue and the RDF data complex structure. This thesis endeavors to integrate communication, synchronization, and distribution techniques with AD methods. Like most machine learning techniques, AD necessitates fixed-length numeric feature vectors. Yet, within the context of KGs, there is no native representation available in fixed-length numeric feature vectors. As a preliminary step, we have proposed a methodology to create fixed-length numeric feature vectors by extracting features from the graph via map-reduce operations. Accordingly, we have developed methods that enable SPARQL-based (SPARQL Protocol and RDF Query Language) feature extraction. Subsequently, we have introduced a scalable anomaly detection framework that can directly identify anomalies in RDF data. Moreover, to improve the transparency of the framework's output, we have provided human-readable explanations to assist users in understanding why detected anomalies should be considered as such. In addition, due to the technological complexity, we have enabled the application of our methods through complementary work, such as integrating them into coding notebooks and REST (Representational State Transfer) API-based environments. Finally, we have extended the existing technology stack SANSA through several scientific publications and software releases, to offer these functionalities to the Semantic Web community.

Acknowledgements

During my Ph.D., I was employed as a researcher at the University of Bonn and was part of a team supporting the EU Horizon 2020 Project, PLATOON.[1,2]

I want to express my deep gratitude for the invaluable support and guidance I've received from numerous individuals who have played a significant role in completing this work. I extend immense gratitude to my supervisor, Prof. Dr. Jens Lehmann, and my mentor, Dr. Hajira Jabeen. I am deeply thankful for granting me a position within the SDA, enabling me to successfully navigate my Ph.D. journey with a harmonious blend of trust, autonomy, and support. Your exceptional kindness, endless patience, and insightful mentorship have been crucial in my academic journey. I consider myself incredibly fortunate to have had the privilege of being advised by you during the development of this thesis.

My heartfelt thanks also go to the entire staff members of the Smart Data Analytics (SDA) group at the University of Bonn. Although I started my Ph.D. during COVID-19 and did not have a chance to visit the team members in person, it was a pleasure to be a part of this vibrant community. I want to extend my gratitude to Carsten Draschner, Firas Kassawat, Afshin Sadeghi, and Claus Stadler for their friendship and support throughout my time at the SDA group. Within our research group, I had the privilege of managing and maintaining certain aspects of the SANSA project, and I am grateful for the opportunity to contribute to its development. I am particularly grateful for the fruitful collaborations that allowed me to implement, evaluate, and integrate the research ideas presented in this thesis into the open-source SANSA project.

Last, but certainly not least, I wish to express my deepest gratitude to my wife, family, and friends for their support and boundless love throughout this journey. Their consistent encouragement and understanding have been the foundation upon which I could rely, enriching my life beyond the realm of scientific pursuits. Their belief in me and their constant presence has been a constant source of inspiration, giving me the strength to overcome challenges and pursue my passion. I am profoundly grateful for their support and the countless sacrifices they have made to ensure my success. Finally, my special thanks go to Seyed Reza Hosseini for his priceless and unexpected scientific comments.

[1] https://cordis.europa.eu/project/id/872592
[2] https://platoon-project.eu

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Problem Definition and Challenges . . . 3

1.2.1 Challenge 1: Scalable and Distributed RDF Vectorization . . . 4

1.2.2 Challenge 2: Scalable and Distributed Anomaly Detection . . . 4

1.2.3 Challenge 3: Scalable and Distributed Explainable Anomaly Detection . . . 4

1.2.4 Challenge 4: User-Friendly Distributed Machine Learning Framework for Knowledge Graphs . . . 4

1.3 Research Questions . . . 5

1.4 Thesis Overview . . . 6

1.4.1 Contributions . . . 6

1.4.2 List of Publications . . . 7

1.5 Thesis Outline . . . 8

2 Preliminaries **11**

2.1 Knowledge Graphs, RDF, and Semantic Web . . . 11

2.1.1 Knowledge Graphs (KGs) . . . 11

2.1.2 Resource Description Framework . . . 13

2.1.3 Ontology . . . 14

2.1.4 SPARQL . . . 14

2.2 Scalable and Distributed Data Analytics . . . 16

2.2.1 Distributed Computing . . . 16

2.2.2 Apache Hadoop . . . 17

2.2.3 Apache Spark . . . 17

2.2.4 Apache Livy . . . 17

2.2.5 Apache Jena . . . 17

2.2.6 Scala . . . 18

2.3 Knowledge Graph Quality Metrics . . . 19

2.3.1 Accuracy . . . 19

2.3.2 Completeness . . . 19

2.3.3 Consistency . . . 19

2.3.4 Timeliness . . . 20

2.3.5 Trustworthiness . . . 20

2.4 Anomaly Detection 20
2.4.1 Anomaly Detection Algorithms 22
2.5 Summary 25

3 Related Work **27**
3.1 KGs as input for Machine Learning 28
3.1.1 SPARQL-based Propositionalzation 28
3.1.2 Graph Kernels 29
3.1.3 Knowledge Graph Embeddings (KGEs) 29
3.2 Machine Learning on Semantic Data 30
3.2.1 Statistical Relational Learning (SRL) Frameworks 30
3.2.2 Knowledge Graph Construction and Mining 30
3.2.3 Distributed Graph Processing 31
3.2.4 Distributed Linked Data Processing 31
3.3 Anomaly Detection on KGs 32
3.3.1 Clustering by `rdf:type` 32
3.3.2 Clustering with Constraints 34
3.3.3 Clustering by LHD 34
3.4 Explainable Anomaly Detection on KGs 35
3.4.1 Pre-Model Techniques 36
3.4.2 In-Model Techniques 36
3.4.3 Post-Model Techniques 37
3.5 Summary 38

4 Scalable and Distributed Feature Extractor **39**
4.1 Motivation 39
4.2 Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor 41
4.2.1 Components 41
4.2.2 Implementation 45
4.3 Use Cases 47
4.4 Experimental Results 48
4.4.1 Experiment A: Assessment of the extracted Features 48
4.4.2 Experiment B: Scalability 51
4.5 Summary 53

5 Scalable and Distributed Anomaly Detection on KGs **55**
5.1 Motivation 55
5.2 DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs . . . 56
5.2.1 Components 57
5.2.2 Implementation 61
5.3 Experiments 62
5.3.1 Experiment A: Assessment of the detected Anomalies 62
5.3.2 Experiment B: Scalability 63
5.4 Summary 65

6 Explainable Anomaly Detection on KGs **67**
6.1 Motivation . . . 67
6.2 ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs . . . 68
6.2.1 Components . . . 69
6.2.2 Implementation . . . 73
6.3 Experimental Results . . . 75
6.3.1 Dataset . . . 75
6.3.2 Experiment A: Assessment of the detected Anomalies Explanation . . . 75
6.3.3 Experiment B: Scalability . . . 78
6.4 Summary . . . 80

7 Implementation and Use-Case **83**
7.1 Resources . . . 84
7.1.1 GitHub Repository . . . 84
7.1.2 Documentation . . . 85
7.1.3 Releases . . . 86
7.2 Scalable Semantic Analytics within PaaS . . . 87
7.2.1 Motivation . . . 87
7.2.2 SANSA through Databricks . . . 87
7.2.3 Summary . . . 89
7.3 Micro-Service based Semantic Analytics integration . . . 90
7.3.1 Motivation . . . 90
7.3.2 Architecture . . . 90
7.3.3 Components . . . 90
7.3.4 Usage . . . 92
7.3.5 Summary . . . 94

8 Conclusion and Future Directions **95**
8.1 Review of the Contributions . . . 95
8.2 Future Work . . . 98
8.3 Closing Remarks . . . 99

A SANSA REST docker-compose.yml File **101**

B List of Publications **105**

Bibliography **107**

List of Figures **121**

List of Tables **123**

CHAPTER 1

Introduction

Within this chapter, we provide a scientific rationale for the topic addressed in this thesis in Section 1.1. We primarily demonstrate the interconnectedness between knowledge graph data and scalable anomaly detection, highlighting the intriguing meta-dimensions involved. Section 1.2 outlines the specific problems and challenges encountered in developing scalable distributed anomaly detection for knowledge graphs. In Section 1.3, we formulate the research questions that guided the endeavors presented in this thesis. Furthermore, in Section 1.4, we offer an overview of our key contributions, presenting the scientific publications that have been developed as part of this thesis and serving as the foundation for several core chapters. Lastly, Section 1.5 gives a concise summary of the content of the subsequent chapters and explains how they relate to the scientific publications that resulted from this research.

1.1 Motivation

Over the past decades, our world has seen dramatic changes due to extensive digitization. This shift has brought forth numerous benefits and analytical opportunities through the use of data. Data comes in various formats and from diverse sources, encompassing text, numbers, date-time, images, audio, video, and graph structures. To make the most of data-driven solutions for different purposes, it's crucial to combine information from these diverse sources.

These types of correlated data can be represented as a Knowledge Graph (KG). KGs model data as a directed graph in which nodes present the entities and edges represent the connection between the entities. The idea of interlinking facts, knowledge, and data sources from the Internet was first introduced by Tim Berners-Lee as the Semantic Web [1]. To facilitate this, the World Wide Web Consortium[1] introduced the Resource Description Framework (RDF)[2] as a standard to model the real world in the form of entities and their relationships. RDF data are a collection of triples $\langle \texttt{subject,predicate,object} \rangle$ with rich relationships that can form a potentially huge and complex RDF graph.

Nowadays, many companies in science, engineering, and business, including bio-informatics, life sciences, business intelligence, and social networks publish their data in the RDF format. Furthermore,

[1] https://www.w3.org
[2] https://www.w3.org/RDF/

the Linked Open Data Project initiative [2] has aided the Semantic Web in gaining traction over the last decade. The Linked Open Data (LOD) cloud currently comprises more than 10,000 datasets available online[3] using the RDF standard.

KGs are being exploited in different real-life use cases as semantic search [3, 4], question answering systems [5], personalized recommendation systems [6, 7], decision support systems [8, 9], and many more. However, to gain the maximum benefit, the KGs should ensure a certain level of quality. This is not a problem per se, because quality typically denotes suitability for a certain use case [10]. KGs are being produced in a variety of ways. Crowd-sourcing was used to create some KGs, such as Wikidata [11] and Freebase [12]. Natural language processing techniques were used to create NELL [13], and DBpedia [14] and YAGO [15] were automatically constructed by knowledge extracting tools. Because usually, the entered data is neither restricted nor cross-validated, KGs are prone to various types of errors due to the variety of approaches and freedom in inserting the input data. These errors can manifest across individual triples, impacting the `subject`, `predicate`, or `object` components of the RDF (Resource Description Framework) data, or even encompass errors in relationships across triples and the presence of hyper-relations beyond standard triple formats.

Finding these errors is not a trivial task due to the unique characteristics of KGs such as complex relationships, diverse entities, the dynamic nature of data, multi-modal data including textual, numerical, and categorical information, and scalability challenges. Traditional methods for ensuring data quality in knowledge graphs often rely on rule-based validation, consistency checks, and semantic constraints. While these approaches are valuable, they may struggle to handle the increasing complexity, dynamic nature, and scale of modern knowledge graphs. Moreover, rule-based systems require prior knowledge about what is considered correct and what is deemed incorrect. However, Anomaly Detection (AD) offers a promising avenue for improving the accuracy and effectiveness of error identification in knowledge graphs. Anomaly detection is a critical task in various scientific and industrial domains that involves identifying patterns or instances that deviate significantly from the expected or normal behavior within a given dataset. The primary objective is to detect rare events, outliers, or anomalies that can provide valuable insights or indicate potential issues in the data. Anomaly detection plays a crucial role in a wide range of applications, including network security, fraud detection, intrusion detection, fault diagnosis, quality control, environmental monitoring, anomaly-based predictive maintenance, and much more [16].

The process of anomaly detection typically involves two main steps: modeling the normal behavior and identifying deviations from this model. Statistical and machine learning techniques are commonly employed to model the normal behavior of the data. These methods can include probabilistic models, clustering algorithms, distance-based approaches, or ensemble methods, among others. By capturing the inherent statistical properties or patterns of the normal data, these models provide a reference for determining what is considered normal or expected.

In recent years, the advancement of big data technologies, the proliferation of sensor networks, and the availability of massive amounts of diverse data have posed both opportunities and challenges for anomaly detection. The increasing complexity and dimensionality of data, along with the need for real-time or streaming anomaly detection, have led to the exploration of novel approaches, including deep learning-based anomaly detection, online learning techniques, and ensemble methods. Additionally, the integration of domain knowledge, contextual information, and the combination of multiple anomaly detection techniques have shown promise in enhancing the accuracy and robustness

[3] http://lodstats.aksw.org/

of anomaly detection systems.

AD is already a well-studied field with a focus specifically on the task of anomaly detection in non-relational datasets [17]. Numerous techniques for detecting outliers and anomalies (anomaly and outlier will be used interchangeably in this thesis) in unstructured collections of multiple dimension points have been developed in recent years. However, despite the significant progress made in the field of anomaly detection, and with the current interest in large-scale heterogeneous data in knowledge graphs, most of the traditional algorithms are no longer directly applicable to KGs due to the scalability issue and the RDF data complex structure. Furthermore, to the best of our knowledge, there has not been a lot of dedicated research work on anomaly detection on KGs. Addressing AD over KGs is crucial to advancing the field and enabling anomaly detection systems to effectively detect and mitigate unexpected events or abnormalities in various scientific, industrial, and societal contexts.

Handling huge KGs is a challenging process. This challenge occurs due to the huge size of KGs which no longer can fit within the main memory (RAM) of conventional computers for processing and further utilization. The limitations of computer main memory are determined by available market modules, and the cost of hardware upgrades escalates significantly beyond the bounds of typical consumer-grade specifications. To address this, horizontal scaling becomes necessary as hardware requirements grow. Horizontal scaling involves leveraging a cluster of multiple computers to provide the necessary resources, enabled by software frameworks that facilitate distributed data management and processing.

For distributed data management, the Hadoop Distributed File System (HDFS) is commonly employed. Moreover, Apache Spark [18] is a widely adopted framework for distributed data processing, particularly within data analysis pipelines. However, existing Apache Spark data analysis pipelines typically require data in tabular format. To bridge this gap, the Scalable Semantic Analytics Stack (SANSA) [19] was developed. SANSA offers native methods to operate on RDF KGs, utilizing the power of Apache Spark [18] and Apache Jena [20]. SANSA framework did not offer an anomaly detection possibility because distributed execution of anomaly detection pipelines on KG data presents unique challenges, particularly in feature retrieval, clustering entities, and ensuring efficient communication and coordination among distributed nodes.

The focal point of this thesis is to exploit the existing communication, synchronization, and distribution techniques to perform distributed anomaly detection on huge knowledge graphs to detect the outliers and enhance the quality of the KGs.

1.2 Problem Definition and Challenges

Dealing with the ever-growing amount of RDF data comes with its own set of complex challenges, especially when it comes to making sure the data is of high quality. Working with large RDF datasets and applying any machine learning (ML) to them is considered as one of the most challenging tasks in the Semantic Web [21]. Anomaly detection, as a sub-field of ML, is not exceptional either. In establishing an effective anomaly detection system, crucial scientific considerations come to the forefront: (i) the meticulous vectorization of the KG, (ii) developing a solid and scalable framework to accommodate the substantial size of KGs, (iii) explaining detected anomalies in a human-readable manner, and (iv) making the framework easily runnable for non-technical users without any difficulties or burdens.

1.2.1 Challenge 1: Scalable and Distributed RDF Vectorization

The first challenge to overcome when applying anomaly detection on large-scale RDF datasets is to vectorize RDF data. As RDF data is represented as a graph structure, extracting valuable knowledge and insights from them is not straightforward, especially when the size of the data is enormous. Although Knowledge Graph Embedding models (KGEs) convert the RDF graphs to low-dimensional vector spaces, these vectors suffer from a lack of explainability because they transform the data from one space to another. Moreover, generating embeddings for very large KGs is challenging. Therefore, there is a need to have a scalable approach that is capable of transforming big RDF data into an explainable feature matrix. This matrix can be exploited in many standard machine-learning algorithms.

1.2.2 Challenge 2: Scalable and Distributed Anomaly Detection

As already explained KGs are being produced in a variety of ways. Crowd-sourcing, natural language processing techniques, and extracting tools. The entered data mostly is neither restricted nor cross-validated. For example, DBpedia extracts information by using heuristic information methods from Wikipedia and may extract incorrect values in the output. These incorrect values sometimes act as outliers, showing anomalous behavior when compared with the rest of the data. For example, a village population may be mistakenly recorded in millions or billions in the RDF database and considered as an outlier when it is compared with the population of the other villages. To enhance the quality of data, it is necessary to identify outliers in the RDF dataset. However, the sheer size of the data (e.g. English DBpedia size is $\sim$ 114 GB) makes it difficult for a regular machine with limited memory and processing speed to handle it effectively. Therefore, distributed computation is required to detect outliers. Hence, we need a scalable anomaly detection framework that can deal with massive RDF datasets.

1.2.3 Challenge 3: Scalable and Distributed Explainable Anomaly Detection

Although numerous techniques for outlier detection have been proposed in the literature, the interpretation of identified outliers is typically left to the users. This can result in users being unsure about how to handle the detected outliers. To address this issue, detected outliers should be accompanied by explanations when they are presented. Explanations can help users comprehend outliers and aid in refining the outlier detection process. As a result, explanations can facilitate outlier mitigation, which involves determining how to utilize identified outliers to improve predictive models. Therefore, we need a scalable anomaly detection framework that is not only able to detect the anomalies in the RDF dataset but also able to generate human-readable explanations for the detected outlier.

1.2.4 Challenge 4: User-Friendly Distributed Machine Learning Framework for Knowledge Graphs

Generally utilizing big data technologies and distributed computing can present significant challenges. These technologies involve complex systems and specialized tools that often require a deep understanding of programming languages, data management, and infrastructure setup. Non-technical users may struggle with the intricate configuration and optimization required to harness the full potential of these technologies. Hence, providing user-friendly approaches to engage with these technologies, even without any programming or scripting knowledge, would prove advantageous.

1.3 Research Questions

As stated in the motivation section above and identified challenges, we define the main research question:

RQ1: Can we vectorize knowledge graphs in a scalable and distributed manner?

To address this question, we proposed a generic, distributed, and scalable software framework that utilizes Apache Spark (a distributed computing framework (Section 2.2.3)) and is able to automatically transform a given RDF dataset to a standard feature matrix by deep traversing the RDF graph and extracting literals to a given depth. This framework (a.k.a Literal2Feature (see Chapter 4)) enables the use of a wide range of machine learning algorithms for the Semantic Web community. The proposed method is able to extract features automatically by creating a SPARQL query to produce the feature matrix. All steps are performed automatically without human intervention. The results of the research question RQ1 allow us to address the defined challenge (cf. Section 1.2.1).

RQ2: How can we apply anomaly detection on knowledge graphs in a scalable and distributed manner?

In order to answer this question, we focused on the anomalies that appear in literals and proposed a generic, distributed, and scalable software framework (a.k.a DistAD (see Chapter 5)) that can automatically detect anomalies in the KGs by extracting semantic features from RDF data, clustering entities, and applying an anomaly detection algorithm on the level of *numeric objects*, *predicates*, and *multi-feature* scenarios. DistAD, with its modulated components, offers flexibility over different parts of the workflow and lets the end-users select different approaches and granularity based on their use cases. The results of the research question RQ2 allow us to address the defined challenge (cf. Section 1.2.2).

RQ3: Can explainable anomaly detection be performed efficiently and effectively on knowledge graphs?

Intending to answer this research question, we proposed a generic, distributed, and scalable software framework (a.k.a ExPAD (see Chapter 6)) that not only automatically detects numeric anomalies in KGs but also produces human-readable explanations for why a given value of a variable in an observation can be considered as outlier. ExPAD works by evaluating and following distributed supervised decision tree splits on variables to detect and explain anomalous cases. The results of the research question RQ3 allow us to address the defined challenge (cf. Section 1.2.3).

Moreover, to address challenge 4 (cf. Section 1.2.4), we introduce a micro-service architecture of a SANSA-enabled Spark and Hadoop cluster with 2 user-friendly interactive communication

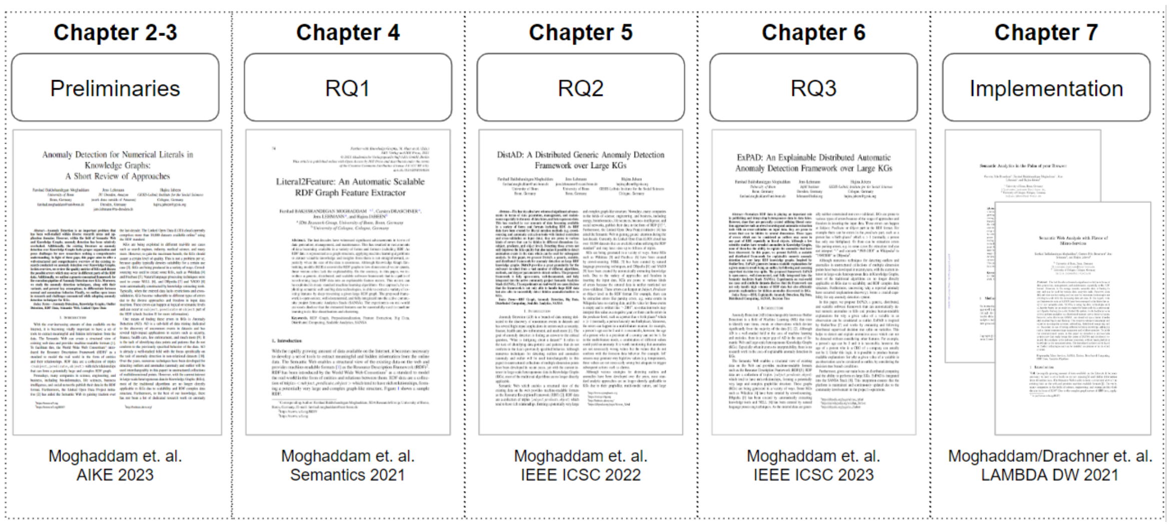

Chapter 2-3
Preliminaries
Anomaly Detection for Numerical Literals in Knowledge Graphs: A Short Review of Approaches
Moghaddam et. al.
AIKE 2023
Chapter 4
RQ1
Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor
Moghaddam et. al.
Semantics 2021
Chapter 5
RQ2
DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs
Moghaddam et. al.
IEEE ICSC 2022
Chapter 6
RQ3
ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs
Moghaddam et. al.
IEEE ICSC 2023
Chapter 7
Implementation
Moghaddam/Drachner et. al.
LAMBDA DW 2021

Figure 1.1: Chapter, Research Questions and Publication Overview

mechanisms a.k.a. REST API and Zeppelin Notebook[4]. Our introduced architecture is based on Docker technologies[5] and can be deployed on-premise without any technical knowledge. Moreover, we explain how to set up SANSA in Databricks [22] as one of several Platform as a Service (PaaS) providers for users who can not provide the necessary hardware. For more information please check Chapter 7.

1.4 Thesis Overview

This section gives an overview of our main contributions conducted during this thesis and the research areas investigated. References to scientific publications covering this study and an overview of the thesis outline are also covered.

1.4.1 Contributions

Our contributions cover several areas of scalable distributed anomaly detection over KGs. These areas include the automatic development of feature-extracting SPARQL queries, distributed anomaly detection, and distributed explainable anomaly detection pipelines for KGs.

1. *Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor [23]*: Literal2Feature represents a versatile, distributed, and scalable software framework designed to transform extensive RDF data into an interpretable feature matrix. This feature matrix can be leveraged by a wide range of standard machine learning algorithms. Our approach harnesses the power of the Semantic Web and Big Data technologies to extract a diverse set of features from large-scale RDF graphs through in-depth traversals. Our proposed framework is publicly available as an

[4] https://zeppelin.apache.org/
[5] https://www.docker.com/

open-source solution, accompanied by comprehensive technical documentation, and seamlessly integrated into the Semantic Analytics Stack community project [19].

2. *DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs [24]*: DistAD represents a distributed and scalable framework designed for anomaly detection on large RDF knowledge graphs. This distributed framework empowers end-users with a high level of granularity, enabling them to choose from a wide range of algorithms, methods, and (hyper-)parameters for detecting outliers. This framework is also openly available as an open-source solution, with comprehensive technical documentation, and integrated into the Semantic Analytics Stack community project [19].

3. *ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs [25]*: ExPAD is a distributed and scalable framework specifically designed for explainable numeric anomaly detection on extremely large RDF knowledge graphs. ExPAD employs a supervised decision tree approach to generate explanations that are easily understandable by humans. These explanations shed light on why a particular result is classified as an outlier, utilizing a meticulous assessment of decision tree splits. ExPAD is publicly available as an open-source solution and integrated into the Semantic Analytics Stack community project [19].

4. *Semantic Web Analysis with Flavor of Micro-Services [26]*: The development of distributed machine learning pipelines based on knowledge graphs requires profound expertise in various fields, including computer systems, networking, and distributed computing. Additionally, setting up the necessary cluster infrastructure demands considerable time and effort, even for skilled developers. To overcome these challenges, we introduce an architecture based on micro-services and REST APIs. This architecture enables end-users to effortlessly access the functionalities of SANSA and perform distributed semantic data analysis without requiring extensive technical knowledge in these specific domains.

5. *Semantic Analytics in the palm of your browser [27]*: The initial setup of a distributed in-memory cluster computation system, along with its dependencies and environments, can present considerable challenges and resource requirements. However, we aim to alleviate these barriers and facilitate the analysis and testing of the SANSA framework by enabling its deployment and utilization solely through a web browser. In this regard, we demonstrate the seamless execution of the SANSA stack within Databricks, eliminating the need for additional Apache Spark knowledge or installations.

6. *Anomaly Detection for Numerical Literals in Knowledge Graphs: A Short Review of Approaches [28]*: As within the field of Semantic Web and knowledge graphs, anomaly detection has been relatively overlooked and also the existing literature on anomaly detection over knowledge graphs lacks proper organization and poses challenges for new researchers seeking a comprehensive understanding, we offer a well-structured and comprehensive overview of the existing research conducted on anomaly detection over knowledge graphs to fill this gaps.

1.4.2 List of Publications

In addition to the thesis itself, this body of work encompasses several scientific publications authored by Farshad Bakhshandegan Moghaddam (Figure 1.1).

- *Conference Papers:*
  1. **Farshad Bakhshandegan Moghaddam**, Carsten Felix Draschner, Jens Lehmann and Hajira Jabeen, "Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor", SEMANTICS, 2021, IOS Press, pp. 74–88. `https://doi.org/10.3233/SSW210036`.
  2. Carsten Felix Draschner, Claus Stadler, **Farshad Bakhshandegan Moghaddam**, Jens Lehmann, Hajira Jabeen, "DistRDF2ML - Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs", Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM). Association for Computing Machinery, New York, NY, USA, 4465–4474. `https://doi.org/10.1145/3459637.3481999`.
  3. **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs", IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 243-250, `https://doi.org/10.1109/ICSC52841.2022.00047`.
  4. **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs", IEEE 17th International Conference on Semantic Computing (ICSC), 2023, pp. 204-211, `https://doi.org/10.1109/ICSC56153.2023.00040`.
  5. **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "Anomaly Detection for Numerical Literals in Knowledge Graphs: A Short Review of Approaches", The Sixth IEEE International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2023, pp. 46-53, `https://doi.org/10.1109/AIKE59827.2023.00015`.

- *Workshops, Demos, and Doctoral Consortium:*
  6. **Farshad Bakhshandegan Moghaddam**, Carsten Felix Draschner, Jens Lehmann, Hajira Jabeen, "Semantic Web Analysis with Flavor of Micro-Services", LAMBDA Doctoral Workshop 2021, `http://ceur-ws.org/Vol-3195/paper1.pdf`
  7. Carsten Felix Draschner, **Farshad Bakhshandegan Moghaddam**, Jens Lehmann, Hajira Jabeen, "Semantic Analytics in the Palm of Your Browser", LAMBDA Doctoral Workshop 2021, `http://ceur-ws.org/Vol-3195/paper2.pdf`

1.5 Thesis Outline

This thesis comprises 8 chapters, each addressing different aspects of the research topic. Chapter 1 serves as an introductory chapter, providing an overview of the scientific fields, and highlighting the problems and challenges. It also outlines the research questions, contributions, and associated papers. Chapter 2 focuses on establishing a common understanding by introducing key terms and concepts, including Semantic Web technologies, scalable and distributed data analytics, and anomaly detection. In Chapter 3, the related literature is explored, primarily focusing on scalable semantic data analytics such as ML frameworks over KGs, anomaly detection, and interpretable AI. Chapter 4 introduces

Literal2Feature, an automated approach for generating SPARQL queries to extract features from KG literals. Chapter 5 demonstrates the distributed anomaly detection framework over KGs (DistAD). Moving forward, Chapter 6 introduces ExPAD, an explainable distributed anomaly detection approach designed for KGs. Chapter 7 presents technical details and approaches for accessible notebooks and REST interfaces, enhancing the accessibility and usability of the developed methods. Finally, Chapter 8 summarizes the thesis on scalable distributed anomaly detection for knowledge graphs. This chapter also offers an outlook on potential future work and research directions.

CHAPTER **2**

Preliminaries

In this chapter, we establish the background of the thesis work by formally explaining the recurring terms. The first section starts with Knowledge Graph and Semantic Web terms. The second section introduces terms and technologies based on technical details in distributed computing and processing. The third section discusses the knowledge graph quality metrics. Finally, we provide an overview of anomaly detection and its associated strategies.

2.1 Knowledge Graphs, RDF, and Semantic Web

This section attempts to define the terminology used in this thesis in the domains of knowledge graphs and Semantic Web. It contains the definition of numerous significant terminologies, including KGs, the Resource Description Framework (RDF), Ontology, and SPARQL.

2.1.1 Knowledge Graphs (KGs)

The rise of knowledge graphs, as a concept for structuring the amount of organized knowledge found on the internet and integrating information from various data sources has received widespread recognition. Moreover, knowledge graphs have begun to play a role in the field of machine learning serving as a means to include knowledge as a representation to clarify learned information [29].

A knowledge graph is a type of labeled graph where the labels have meanings. The graph consists of nodes, edges, and labels. Nodes can represent entities, like people, companies, or computer systems. Edges connect pairs of nodes and represent the relationships between them such as friendship between two people, customer relationships between a company and an individual, or network connections between computer systems. The labels, on the edges, describe the nature of these relationships like explaining what kind of friendship exists between two individuals.

Knowledge graphs have gained attention, particularly following the introduction of the Google Knowledge Graph in 2012. Various definitions have been put forward to describe them [30]. Here, some of the most recent and prominent definitions are presented as follows:

Definition 2.1.1 (Knowledge Graph)**.** *A knowledge graph is a semi-structured data model characterized by three components: (i) a ground extensional component, that is, a set of relational constructs for schema and data (which can be effectively modeled as graphs or generalizations thereof);*

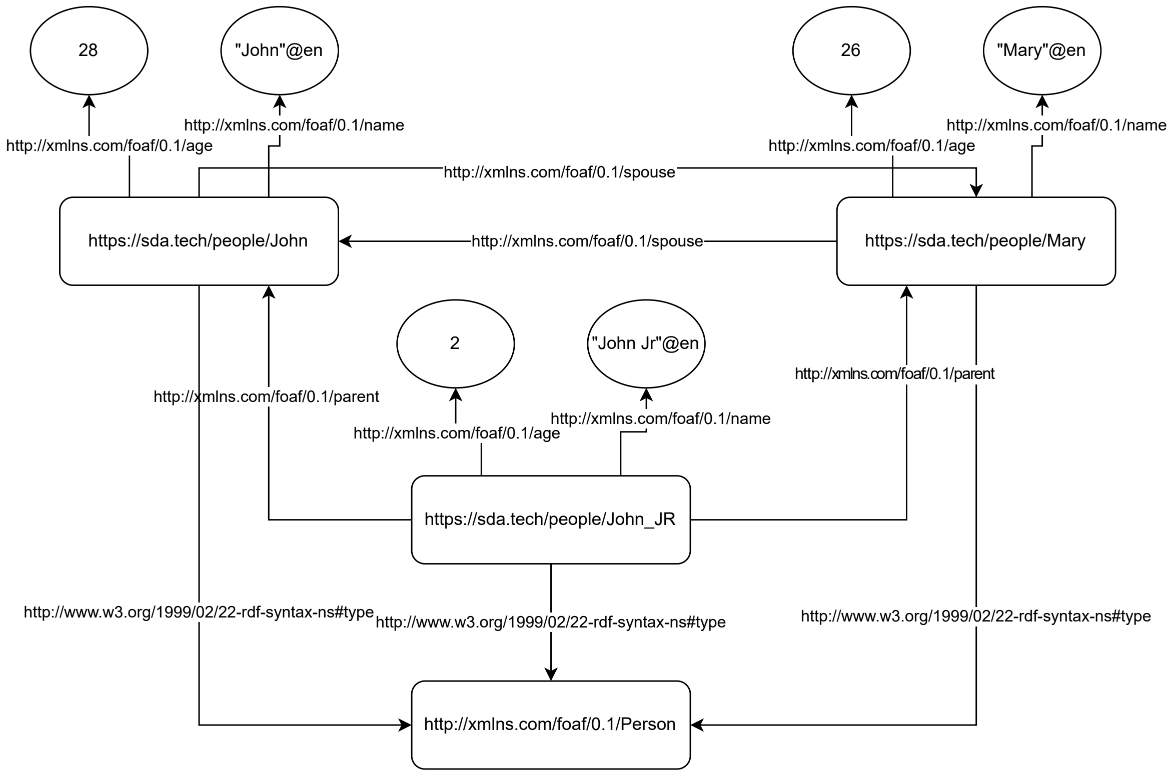

28
"John"@en
26
"Mary"@en
http://xmlns.com/foaf/0.1/age
http://xmlns.com/foaf/0.1/name
http://xmlns.com/foaf/0.1/spouse
https://sda.tech/people/John
https://sda.tech/people/Mary
2
"John Jr"@en
http://xmlns.com/foaf/0.1/parent
https://sda.tech/people/John_JR
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://xmlns.com/foaf/0.1/Person

Figure 2.1: A sample RDF graph

(ii) an intentional component, that is, a set of inference rules over the constructs of the ground extensional component; (iii) a derived extensional component that can be produced as the result of the application of the inference rules over the ground extensional component (with the so-called "reasoning" process) [31].

Definition 2.1.2 (Knowledge Graph)**.** *A knowledge graph mainly describes real-world entities and their interrelations, organized in a graph; defines possible classes and relations of entities in a schema; allows for potentially interrelating arbitrary entities with each other; covers various topical domains [32].*

Definition 2.1.3 (Knowledge Graph)**.** *A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge [33].*

More formally, [34] gives the formal definition of a knowledge graph as an RDF graph.

Definition 2.1.4 (Knowledge Graph)**.** *An RDF graph consists of a set of RDF triples (see Section 2.1.2), where each RDF triple (s, p, o) is an ordered set of the following RDF terms: a subject $s \in U \cup B$, a predicate $p \in U$, and an object $U \cup B \cup L$. An RDF term is either a URI $u \in U$, a blank node $b \in B$, or a literal $l \in L$. U, B and L are pairwise disjoint.*

Figure 2.1 depicts a sample knowledge graph (Literals are specified with round shapes).

2.1.2 Resource Description Framework

Resource Description Framework (RDF) is a framework for describing resources [1]. A resource is a fact or a thing that can be described and identified. RDF has been introduced by the World Wide Web Consortium (W3C)[1] as a standard to model the real world in the form of entities and relations between them. W3C maintains the standards for RDF, including the foundational concepts, semantics, and specifications for different formats. The first syntax defined for RDF was based on the Extensible Markup Language (XML). Other syntaxes are now more commonly used, including Terse RDF Triple Language (Turtle), JavaScript Object Notation for Linked Data (JSON-LD), and N-Triples. RDF data is a collection of triples ⟨`subject,predicate,object`⟩ which tends to have rich relationships, forming a potentially very large and complex graph-like structure. The subject and object are nodes that represent things. The predicate is an arc because it represents the relationship between the nodes. Figure 2.2 represents a schematic RDF triple.

Overall, the RDF standard contains three different types of nodes:

- **Uniform Resource Identifier (URI)** The URI format is a standardized way of identifying a resource, whether it is abstract or physical. A Uniform Resource Locator (URL) is a specific type of URI frequently used in RDF statements. In 2014, when the RDF specification was updated to version 1.1 by W3C, the Internationalized Resource Identifier (IRI) was introduced as a node type. IRIs are comparable and compatible with URIs, allowing the use of international character sets.

- **Literal** A literal is a distinct data value that may take the form of a string, date, or numerical value.

- **Blank Node** A blank node identifier, also referred to as a bnode or an anonymous resource, denotes a subject for which only the relationship is known. These identifiers are identified by a specific syntax.

The formal definition of RDF triple is as follows:

Definition 2.1.5 (RDF Triple)**.** *Given an infinite set U of URIs, an infinite set B of blank nodes, and an infinite set L of literals, a triple $\langle s, p, o\rangle \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple where s, p, o represent the subject, predicate, and object, respectively, of the triple.*

Listing 2.1 shows 3 sample triples from Figure 2.1.

```
+--------------------------------+---------------------------------+----------+
|Subject                         |Predicate                        |Object    |
+--------------------------------+---------------------------------+----------+
|https://sda.tech/people/John_JR |http://xmlns.com/foaf/0.1/age    |2         |
|https://sda.tech/people/John    |http://xmlns.com/foaf/0.1/name   |"John"@en |
|https://sda.tech/people/Mary    |http://xmlns.com/foaf/0.1/age    |26        |
+--------------------------------+---------------------------------+----------+
```

Listing 2.1: Sample RDF Triples

[1] https://www.w3.org

2.1.3 Ontology

Ontology plays a critical role in the development and implementation of the Semantic Web. Ontology is essentially a formal and explicit specification of a shared conceptualization of a domain, which describes the types of entities and relationships that exist within the domain. It provides a standardized vocabulary for describing information, allowing for more accurate and efficient information retrieval and knowledge sharing.

In the Semantic Web, ontologies can be represented by Resource Description Framework Schema (RDFS) and Web Ontology Language (OWL). RDFS provides a foundational framework for creating ontologies, offering constructs for defining classes, properties, and hierarchies. OWL, an extension of RDFS, introduces more expressive elements, enabling the specification of richer relationships and supporting complex reasoning capabilities.

These frameworks facilitate machine-understandable depictions of domain knowledge, fostering seamless data integration from diverse sources. By establishing a universal language for expressing meaning, ontologies, RDFS, and OWL collectively promote interoperability among different systems and applications.

One of the key benefits of ontologies is that they enable the creation of intelligent systems that can reason about the meaning of data. By using logical rules and inference mechanisms, ontologies can be used to infer new knowledge from existing data or to check the consistency of data. This makes it possible to automate many tasks that would otherwise require human intervention, such as data integration, classification, and decision-making.

Ontologies can be developed in a variety of ways, ranging from manual creation to automatic extraction from data. However, creating a high-quality ontology requires significant domain knowledge and expertise, as well as an understanding of the needs and requirements of the intended users. Ontology development typically involves a collaborative effort between domain experts, knowledge engineers, and ontology developers. Therefore ontology is a fundamental concept in the Semantic Web, providing a powerful tool for knowledge representation, integration, and reasoning. As the amount of data and information available on the web continues to grow, ontologies become increasingly important for enabling efficient and effective information retrieval and sharing.

2.1.4 SPARQL

An RDF graph is a data format that uses a directed and labeled graph to represent information on the Web. SPARQL, a recursive acronym for Simple Protocol and RDF Query Language (pronounced “sparkle”), is a query language used to retrieve and manipulate data stored in RDF. SPARQL is capable of expressing queries across a variety of data sources, regardless of whether the data is stored natively as RDF or accessed as RDF through middleware. It provides features for querying required and optional graph patterns, along with their conjunctions and disjunctions. Additionally, SPARQL allows for extensible value testing and constraining queries by source RDF graph. The outcome of SPARQL queries may be sets or RDF graphs.

The upcoming section will delve into the fundamentals of SPARQL and its syntax, drawing comparisons to the definitions found in [35]. More details can also be found in the W3C specification of SPARQL[2].

[2] https://www.w3.org/TR/rdf-sparql-query/

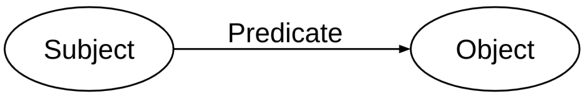


Figure 2.2: An RDF triple

Definition 2.1.6 (RDF Terms)**.** *Let U be the set of all IRIs. Let RDF_L be the set of all RDF Literals, and let RDF_B be the set of all blank nodes in RDF graphs. The set of RDF Terms, T, is* $(U \cup RDF_L \cup RDF_B)$.

Definition 2.1.7 (Query Variable)**.** *A query variable is a member of the set V where V is considered infinite and disjoint from T.*

Definition 2.1.8 (Triple Pattern)**.** *Let V be a set of variables such that $V \cap T = \emptyset$. A triple pattern tp is a member of the set $(T \cup V) \times (U \cap V) \times (T \cup V)$.*

Definition 2.1.9 (Basic Graph Pattern (BGP))**.** *Let $tp = \{tp_1, tp_2, tp_3, ..., tp_n\}$ be a set of triple patterns. A Basic Graph Pattern BGP is a conjunction of triple patterns, i.e $BGP = tp_1 \wedge tp_2 \wedge tp_3 \wedge, ..., \wedge tp_n$.*

Definition 2.1.10 (Solution Modifiers)**.** *A solution modifier is a mapping from a set of V to a set of T . More formally, $SM = \{(v, modifier(v)) | v \in V\}$, where the modifier is one of the project, distinct, order, limit, and offset modifiers.*

Definition 2.1.11 (Result Set)**.** *Given $Q = (BGP, D, SM, SELECT\ V)$, then a result set QS is a solution formed by matching dataset D with graph pattern BGP.*

Definition 2.1.12 (SPARQL Query)**.** *A SPARQL query is a tuple (BGP, D, SM, QS).*

Let us consider an example for a better understanding of SPARQL. Assume that we want to know "Who are the parents of John Jr. and how old are they?" from the sample knowledge base (as depicted in Figure 2.1). Listing 2.2 depicts a simple SPARQL query to retrieve information.

Listing 2.2: A SPARQL query to retrieve the parents of John Jr and their ages

```
1 PREFIX sdaMember: <http://sda.tech/people/>
2 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3
4 SELECT ?parent ?parent_age WHERE {
5   sdaMember:John_jr foaf:parent ?parent.
6   ?parent foaf:age ?parent_age
7 }
```

SPARQL has a similar SQL-like syntax. At the beginning prefixes as optional headers are given. It helps the reader to make the rest of the query more readable. Then the query form is defined. Here `SELECT` query has been used. The `WHERE` clause is used to add constraints and define the SPARQL query.

More specifically, in Listing 2.2, lines 1-2 define prefixes as a short version of URIs. Line 4 is the `SELECT` clause which declares the variables that should be retrieved as an output when executing the query. There are two variables `?parent` and `?parent_age`. In SPARQL, the variables are defined with a "?" symbol. The following statements (lines 5-6) include two Basic Graph Pattern (BGP)s. The first one (line 5) states that in the statement with subject `sdaMember:John_jr` and property `foaf:parent`, we assign the value of its object to a variable called `?parent`. When evaluated, this variable will contain the values of `sdaMember:John` and `sdaMember:Merry`. Afterwords (line 6), the same variable `?parent` with an associated value will be the subject of the next statement. The remaining variable `?parent_age` then will take the values 28 and 26 respectively. As an output, both values of the variables `?parent` and `?parent_age` will be retrieved. The above query returns the following result set on our data graph (Table 2.1).

Table 2.1: The result of the SPARQL query specified in Listing 2.2

?parent	**?parent_age**
`<http://sda.tech/people/John>`	28
`<http://sda.tech/people/Marry>`	26

2.2 Scalable and Distributed Data Analytics

Big data is a term that means a large volume of data that is difficult to handle by a single machine with traditional techniques. The size of KGs can become extremely large. This makes it difficult to execute the algorithms with nondeterministic polynomial time (NP) or even polynomial complexity on the data in a reasonable amount of time, even with high-performance resources. Furthermore, because the data requires significant processing power to be represented and analyzed, additional processing power is needed in addition to dealing with computational complexity. Because of the size of the datasets, serial computation is not feasible for large KGs. Therefore, a distributed or parallel computing framework is necessary to distribute the datasets across cores via networks and speed up processing.

2.2.1 Distributed Computing

Distributed computing is a computing paradigm that involves multiple computers working together to solve a computational problem. In a distributed computing system, the workload is distributed across multiple nodes, each with its own processing power and memory. This approach allows for large-scale data processing and analysis that would be impossible on a single machine. Distributed computing systems typically include a central coordinator that manages the distribution of tasks and data among the nodes, as well as mechanisms for communication and synchronization between the nodes. Examples of distributed computing frameworks include Apache Hadoop and Apache Spark, which are widely used for processing big data in various industries. Distributed computing is a crucial component of modern computing systems, as it enables the processing and analysis of vast amounts of data that would otherwise be impossible to handle.

2.2.2 Apache Hadoop

Apache Hadoop [36] is a set of frameworks for distributed storage and processing of large-scale datasets across a cluster of computers. Its ecosystem includes built-in mechanisms that ensure fault tolerance and high availability on top of commodity hardware, eliminating the need for specialized hardware. This makes it highly scalable and cost-effective. Hadoop File System (HDFS) [37] is a core component of distributed handling of large-scale datasets, which includes data partitioning and fault tolerance techniques. HDFS reduces data movement and duplication by splitting data into blocks and distributing replicated blocks across nodes in the system. The HDFS architecture follows a driver/worker model, with the namenode serving as the driver that manages all operational steps, including the namespace, replication, and metadata, and tracks processed operations. To ensure fault tolerance, a fallback passive namenode is initiated for the namenode, which can take over the driver's work in case of failure to enable faster recovery. The datanodes serve as workers, providing access to and storage of data.

2.2.3 Apache Spark

Apache Spark[3] is a distributed computing framework that is designed to process large-scale data in parallel across a cluster of computers. It is known for its speed and ease of use, as it allows users to perform complex data processing tasks with simple and concise code. Spark offers a range of programming languages, including Java, Python, and Scala, making it accessible to a wide range of developers. Spark's built-in machine learning library, MLlib [38], provides a convenient way to perform various machine learning tasks, such as classification, clustering, and regression, on large datasets. Another key feature of Spark is its ability to integrate with other big data tools, such as Hadoop, making it a versatile option for data processing and analysis. Figure 2.3 depicts a cluster mode overview architecture of Spark.

2.2.4 Apache Livy

Apache Livy[4] is an open-source project that enables running and managing interactive Apache Spark applications on remote clusters. It provides a RESTful interface that allows users to submit Spark jobs and perform interactive queries from various programming languages such as Python, R, and Scala. Livy helps simplify the deployment and management of Spark applications on shared clusters, as it allows multiple users to work on the same cluster concurrently. Additionally, Livy provides a session management feature that enables users to reuse their Spark sessions, which can help reduce overhead and increase application performance.

2.2.5 Apache Jena

Apache Jena[5] is a Java framework and toolkit for building semantic web and linked data applications. It provides a set of libraries and tools for working with RDF, OWL (Web Ontology Language), and SPARQL standards. At its core, Apache Jena provides a comprehensive set of APIs for creating,

[3] https://spark.apache.org/
[4] https://livy.apache.org/
[5] https://jena.apache.org/

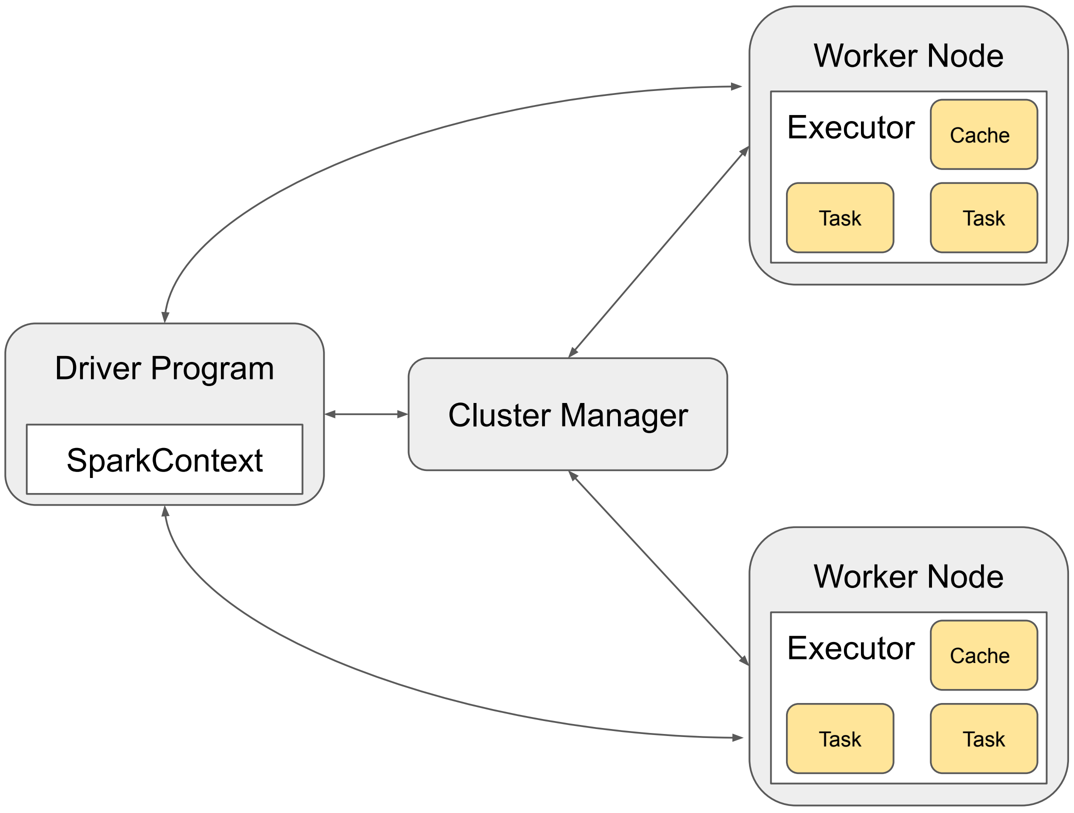


Figure 2.3: Apache Spark architecture diagram

managing, and querying RDF data. With numerous functions, it natively handles KG data, enabling the creation of entities, literals, and triple data, as well as supporting SPARQL filter functions. Thus, KG data can be effectively worked with in traditional Java programs.

2.2.6 Scala

Scala [39] is a general-purpose programming language that is designed to be concise, expressive, and scalable. It is a hybrid functional and object-oriented language that runs on the Java Virtual Machine (JVM), allowing for seamless integration with existing Java code and libraries. Scala is known for its ability to handle complex data processing tasks with ease, making it a popular choice for big data processing and analytics. Additionally, Scala's type inference system and support for functional programming paradigms allow for code that is both concise and expressive. Another key feature of Scala is its excellent support for concurrent and parallel programming. This is crucial for big data processing, where tasks are often parallelized to improve performance.

2.3 Knowledge Graph Quality Metrics

Knowledge graphs have become increasingly popular as a way to organize and represent structured data, allowing for more effective search and discovery. However, the usefulness of a knowledge graph depends heavily on its quality, and there are a number of metrics that can be used to assess this quality. As the main goal of this thesis is to improve the quality of KGs, in this section, we will explore some of the key knowledge graph quality metrics that are commonly used. Table 2.2 lists the metrics, definitions, and anomalous examples.

Table 2.2: Definitions of evaluation dimensions

Dimension	**Definition**	**Example (negative)**
Accuracy	Correctness of facts	`(Barack_Obama, birthPlace, Germany)`
Completeness	Coverage of Knowledge by the KG	-
Consistency	Degree of self-contradiction in the KG	`(John, spouseOf, Mary)` `(Mary, sisterOf, John)`
Timeliness	Degree to which knowledge is up-to-date	`(Barack_Obama, presidentOf, USA)`
Trustworthiness	Degree of objectivity, authority, and verifiability of a KG	-

2.3.1 Accuracy

Accuracy [40] is an important quality metric for a knowledge graph. It measures the extent to which the graph represents the true relationships and attributes for a given domain. An accurate knowledge graph should have relationships and attributes that are as close to the truth as possible.

To measure accuracy, one can use the error rate and accuracy metrics. Error rate measures the ratio of the number of incorrect relationships or attributes to the total number of relationships or attributes in the graph. Accuracy measures the ratio of the number of correct relationships or attributes to the total number of relationships or attributes in the graph. A low error rate and high accuracy indicate a more accurate knowledge graph. This thesis primarily centers on enhancing accuracy.

2.3.2 Completeness

Completeness [41] is one of the most critical quality metrics for a knowledge graph. It measures the extent to which the graph represents all the relevant entities and relationships for a given domain. A complete knowledge graph should include all the relevant entities and relationships that exist in the domain it represents.

To measure completeness, we can use the recall metric. Recall is the ratio of the number of correct entities or relationships to the total number of entities or relationships that should be in the graph. A high recall indicates a more complete knowledge graph.

2.3.3 Consistency

Consistency [42] is another critical quality metric for a knowledge graph. It measures the extent to which the graph is free from contradictions and inconsistencies. A consistent knowledge graph should not have any conflicting information or contradictory relationships.

Table 2.3: Knowledge graph quality metric correlation

	Accuracy	Completeness	Consistency	Timeliness	Trustworthiness
Accuracy		↓	↑	-	↑
Completeness	↓		↓	-	-
Consistency	↑	↓		-	↑
Timeliness	-	-	-		-
Trustworthiness	↑	-	↑	-	

*↑, ↓, and - indicate positive, negative, or almost not correlated

To measure consistency, we can use coherence and conflict resolution metrics. Coherence measures the extent to which the information in the knowledge graph is consistent with the domain knowledge. Conflict resolution measures the ability of the knowledge graph to resolve conflicting information. A high coherence and conflict resolution indicate a more consistent knowledge graph.

2.3.4 Timeliness

Timeliness [43] is another quality metric for a knowledge graph. It measures the extent to which the graph is up-to-date with the latest information for a given domain. A timely knowledge graph should have the latest information about entities and relationships in the domain.

To measure timeliness, we can use the freshness and recency metrics. Freshness measures the extent to which the knowledge graph includes the latest information about entities and relationships. Recency measures the time elapsed since the last update to the knowledge graph. A high freshness and low recency indicate a more timely knowledge graph.

2.3.5 Trustworthiness

Trustworthiness [44] refers to the degree of reliability and credibility of the information contained within a knowledge graph. It measures the extent to which the data in the knowledge graph can be trusted and relied upon by users for making informed decisions or drawing accurate conclusions.

Trustworthiness is an important quality metric because knowledge graphs often integrate data from various sources, and not all sources are equally reliable. Evaluating the trustworthiness of a knowledge graph helps users assess the credibility of the information and determine the level of confidence they can place in the data.

In conclusion, knowledge graph quality metrics are critical for evaluating the effectiveness of a knowledge graph. By measuring the completeness, consistency, accuracy, timeliness, and coherency of a knowledge graph, one can assess and improve its quality. Table 2.3 shows the correlation between the mentioned metrics [42, 45, 46].

2.4 Anomaly Detection

Anomaly Detection (AD) is a branch of data mining dedicated to the discovery of uncommon events in datasets and has several high-impact applications in domains such as security, finance, health care, law enforcement, and much more [16]. The goal of anomaly detection is finding an answer to the critical question, “What is intriguing about a dataset?” It refers to the task of identifying data point(s) and patterns that do not conform to the data’s previously specified behavior. Although numerous techniques for detecting outliers and anomalies (anomaly and outlier will be used interchangeably in this thesis) in unstructured collections of multiple dimension points have been developed in recent years, yet with the current interest in large-scale heterogeneous data in knowledge graphs, most of the traditional algorithms are no longer directly applicable. This occurs due to the intricate structure of knowledge graphs, which cannot be directly input into anomaly detection approaches. Furthermore, the substantial size of large KGs poses challenges in loading them into the main memory for effective execution of AD.

The first and most widely used definition of an outlier dates from 1980 and is given in [47]:

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

As the definition indicates, anomalies are not necessarily wrong values but values that do not conform with normal data behavior. Outliers are also referred to as abnormalities, discordants, deviants, or anomalies in the data mining and statistics literature [48]. Anomaly detection is a well-studied area and has found its way to many real-world scenarios such as Intrusion Detection Systems, Financial Fraud Detection, IoT and sensor data, Medical Diagnosis, Law Enforcement, Earth Science, and many more [48].

We need to distinguish the difference between *outlier* and *natural outliers*. *Natural outliers* are the values that are not wrong. For instance, consider a property like `dbpedia-owl:population-Total`, designed to denote the overall population of a `dbpedia-owl:PopulatedPlace`. This category contains a diverse range of locales, including villages, towns, cities, states, countries, and continents. That means that most continents will appear to be outliers by most metrics because they are only few, in comparison to the population of the villages, towns, and cities, that make up the majority of the entries. These types of anomalies are called *natural outliers*. Thus, when using outlier detection to find errors in data, special care must be taken to distinguish natural outliers from outliers caused by actual data errors.

Anomaly detection techniques can be categorized from different aspects. Generally, they can be grouped to *supervised*, *semi-supervised*, and *unsupervised*. Moreover, the methods can be categorized as *neighbor-based*, *subspace-based*, and *ensemble-based* detection methods. Based on the number of features they are being applied, the methods can be classified as *univariate* or *multi-variate*. Although a comprehensive review of anomaly detection techniques is beyond the scope of this thesis; we refer the reader to previous survey publications for a thorough discussion of such approaches [16, 17, 49].

Supervised and *semi-supervised* approaches require training data in which outlier/normal values are labeled. In contrast, *unsupervised* approaches do not rely on any training data and are independent of such data. As the creation of training labeled data would be rather expensive and labor-intensive, the most used outlier detection methods are unsupervised.

The primary objective of *Neighbor-Based Detection* methods is to identify outliers using neighborhood data. For example, the anomaly score of a data point can be defined as the average distance or weighted distance to its k-nearest neighbors [50, 51]. Another approach is to consider the Local Outlier Factor (LOF) [52] as the measurement of anomaly degree, in which the anomaly score was measured relative to its neighborhood. In contrast, methods in *Subspace-Based Detection* attempt to project high-dimension data to lower dimensions and then search for anomalies. The reason for this is that anomalies frequently exhibit abnormal behavior in one or more low-dimensional sub-spaces. The full-dimensional analysis would obscure low-dimensional abnormal behaviors [53]. For example, [54] demonstrated that for an object in a high-dimensional space, only a subset of relevant features provides useful information, while the rest is irrelevant to the task. In the literature, subspace learning is a popular technique for dealing with high-dimensional problems [55–58]. Aside from that, the *Ensemble-Based Detection* method detects anomalies by utilizing various learning techniques or even multiple sub-spaces at the same time. Because of the complexity of the data, none of the outlier detection methods can detect all anomalies in a low-dimensional subspace. One ensemble strategy is, for example, summarizing the anomaly scores and selecting the best one after ranking [59].

If the AD approach considers multiple dimensions of data at once (for example considering longitude and latitude together to detect a geo-coordinate as an anomaly) it is called *multi-variate* and if it just utilizes a single dimension (for example only checking the age of people to detect anomalies) it is called *univariate*.

2.4.1 Anomaly Detection Algorithms

In this section, we briefly summarize the most common statistical anomaly detection algorithms.

Interquartile Range

The Interquartile Range (IQR) [60] technique is a statistical metric that is based on calculating the first quartile ($Q1$), the median ($Q2$), and the third quartile ($Q3$) of a numerical dataset. The difference between $Q3$ and $Q1$ is called IQR. Outliers are data points that are less than $Q1 - 1.5 \times IQR$ and more than $Q3 + 1.5 \times IQR$. Figure 2.4 depicts the IQR and the outliers ranges.

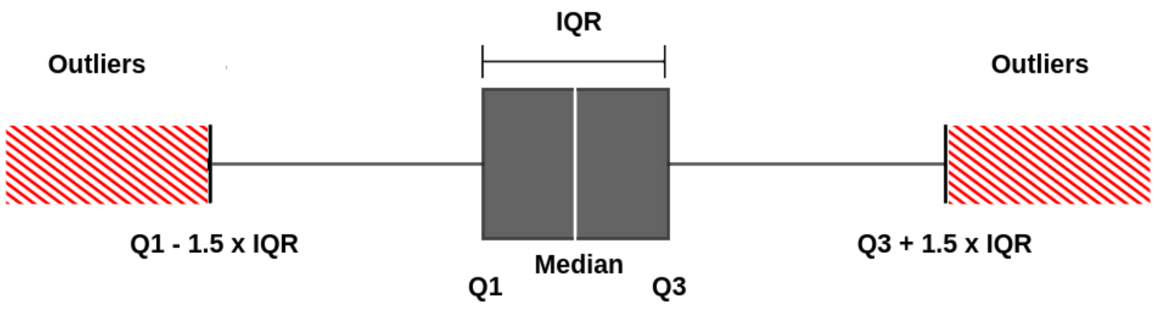


Figure 2.4: Box and Whiskers plot for Interquartile Range

Median Absolute Deviation

The Median Absolute Deviation (MAD) [61] is a measure of the variability of a univariate sample of numeric data. The MAD for a data collection $X = \{x_1, x_2, ..., x_n\}$ is defined as the median of the absolute deviations from the median of the data. So if $\widetilde{x} = median(X)$ then:

$$MAD = median(|x_i - \widetilde{x}|)$$

The values in X that are more than $\widetilde{x} + 2.5 \times MAD$ and less than $\widetilde{x} - 2.5 \times MAD$ are outliers. MAD is more resistant to data set outliers than the standard deviation technique.

Z-Score

The Z-Score is the number of standard deviations by which the value of a raw score (i.e., an observed value or data point) is above or below the mean value of what is being observed or measured. Positive standard scores are assigned to raw scores that are greater than the mean, while negative standard scores are assigned to raw scores that are less than the mean. For example, a Z-Score of 1.5 indicates that the data point is 1.5 units away from the mean, indicating that it is an outlier. The Z-Score is defined as:

$$z - score = \frac{x - \mu}{\sigma}$$

where μ is the mean of the data and σ is the standard deviation.

Kernel Density Estimation

Kernel Density Estimation (KDE) [62] is a non-parametric approach to estimating the probability density function of a random variable. Let $x_1, x_2, ..., x_n$ be independent and identically distributed samples drawn from some univariate data with an unknown density f at any given point x. Then the kernel density estimator of f is:

$$\hat{f}_h(x) = \frac{1}{n \times h} \sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

where K is the kernel, a non-negative function, and $h > 0$ is a smoothing parameter called the bandwidth. To obtain outlier scores for a given dataset, firstly a KDE should be constructed from the data and then the resultant probability at each point should be calculated. To put this probability into context, it should be compared to the mean probability across all points $p = \frac{1}{n} \sum_{i=1}^{n} \hat{f}_h(x_i)$. The relative value of one data point being normal is therefore $\hat{v}(x) = \frac{\hat{f}_h(x)}{p}$, where $\hat{v}(x) > 1$ represents an above average and $\hat{v}(x) < 1$ represents a below average. A predefined threshold can be used to produce a binary classification, for example, all x with $\hat{v}(x) < 0.2$ are deemed outliers.

Local Outlier Factor

Local Outlier Factor (LOF) [52] is one of the density-based anomaly detection approaches. LOF is particularly useful when dealing with datasets where anomalies are not globally rare but are instead clustered in certain regions. The local outlier factor is based on the concept of local density, where locality is defined by k nearest neighbors, the distance between which is used to estimate the density. Regions with similar densities and spots with much lower densities than their neighbors can be detected by comparing an object's local density to the local densities of its neighbors. These are known as outliers.

Global Anomaly Score (GAS)

GAS is one of the most often used nearest-neighbor algorithms. The anomaly score is either set to the average distance of the k nearest neighbors, as recommended in [51], or to the distance to the k^{th} neighbor, as proposed in [50], following the intuition that outliers are located in rather sparsely populated areas of the vector space. It is worth noting that the first strategy is far more resistant to statistical variations.

One-Class Support Vector Machine

One-class SVM is a multi-variant anomaly detection algorithm which unlike traditional SVM aims to develop a decision boundary that produces the greatest separation between the points and the origin [63]. A one-class SVM projects data into a higher dimensional space using the kernel's implicit transformation function, $\varphi(\cdot)$. The algorithm then learns the decision boundary (a hyperplane) that separates the bulk of the data from the origin. Only a few data points are allowed to fall on the opposite side of the decision boundary; these data points are known as outliers. Figure 2.5 depicts how One-Class SVM detects an anomaly.

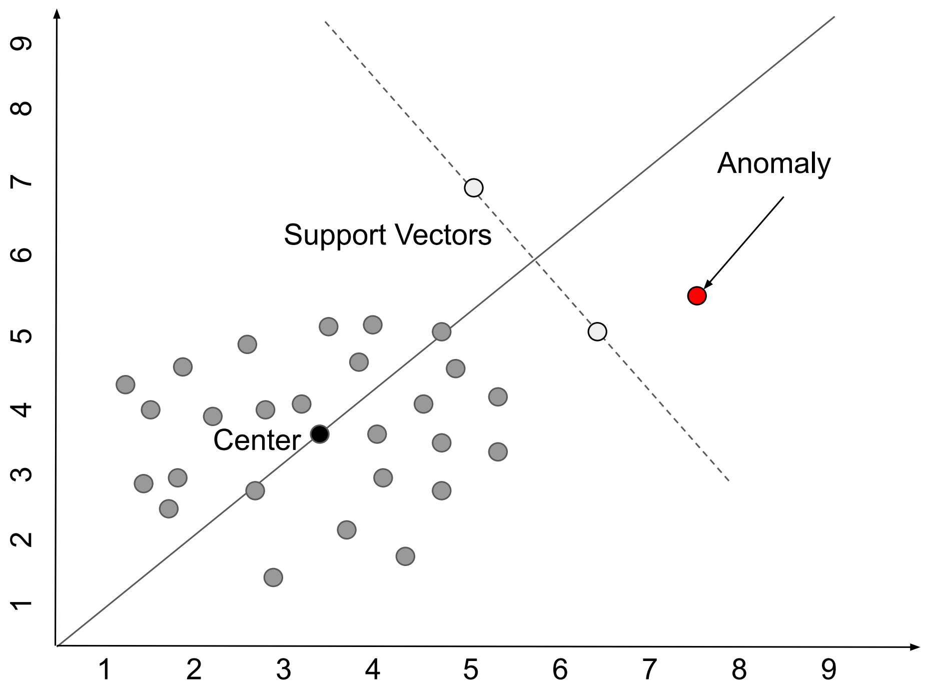


Figure 2.5: An example of One-Class Support Vector Machine in a 2D space for detecting anomalies

Isolation Forest

Isolation Forest (IF) [64] is also a multi-variant anomaly detection algorithm that identifies anomalies through isolation. Like Random Forests, Isolation Forest is built with decision trees. From the training data, the algorithm generates an ensemble of isolation trees. The Isolation Forest approach is based on the fact that anomalous examples in a dataset are easier to isolate (separate) from the rest of the regular points. In an Isolation Forest, randomly sub-sampled data is processed in a tree structure based on randomly selected features. Anomalies are less likely to arise in greater-depth samples because they require more cuts to separate them. Similarly, samples that end up in shorter branches indicate anomalies. Figure 2.6 depicts how IF detects a point that's more likely to be an anomaly.

2.5 Summary

In this chapter, we explained pivotal terms such as Knowledge Graph and Semantic Web and related terminologies such as RDF, Ontology, SPARQL, etc. elucidating their interconnections. Our exploration extended to an examination of technologies utilized in distributed computing and processing, aiming to enhance the comprehension of how data is managed across diverse systems. Additionally, we introduced knowledge graph quality metrics and delved into their correlation. Concluding our discussion, we focused on anomaly detection, providing an overview of its types and methods, and introduced some prominent anomaly detection approaches and their characteristics.

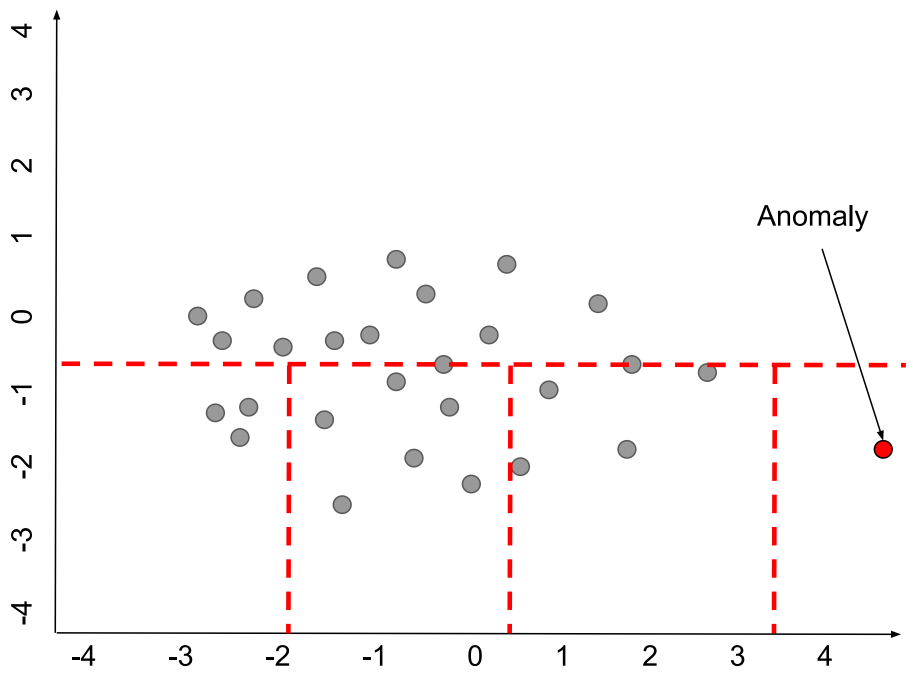


Figure 2.6: An example of isolating an anomalous point in a 2D space with IF

CHAPTER 3

Related Work

This chapter provides a comprehensive review of the related work that has informed the development of this thesis. In particular, we focus on four main areas: KGs as input for machine learning (ML) techniques, machine learning frameworks that operate on KGs, existing approaches of anomaly detection on KG data, and explainable AI. Section 3.1 presents useful techniques for building ML pipelines with knowledge graph data. Specifically, the emphasis is on the various techniques that have been proposed for extracting fixed-length numeric feature vectors from KG data. These techniques are crucial for our research as they allow us to represent KG data in a format that can be easily processed by machine learning algorithms. In Section 3.2 we review the already existing frameworks that are able to operate on KG data and apply ML techniques to them. Section 3.3 presents a summary of the related work on anomaly detection on KG data. We describe the most commonly used techniques in this area and discuss their strengths and limitations. Finally, in Section 3.4, we delve into the topic of explainability in the field of machine learning, with a particular focus on explainability techniques in the context of anomaly detection. We discuss the importance of explainability in this area and present the various approaches that have been proposed to provide explanations for anomalous behavior in KG data.

The content of this chapter is influenced by the related work sections of our following publications [23–25, 28]. These sources have provided valuable context and background information that has shaped our understanding of the current state of research in the areas of KG feature extraction, anomaly detection on KGs, and explainable anomaly detection. We have used these studies to guide our research direction and identify important gaps in the existing literature that our work intends to fill. The sources are as follows:

- **Farshad Bakhshandegan Moghaddam**, Carsten Felix Draschner, Jens Lehmann and Hajira Jabeen, "Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor", SEMANTICS, 2021, IOS Press, pp. 74–88. `https://doi.org/10.3233/SSW210036`.
- **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs", IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 243-250, `https://doi.org/10.1109/ICSC52841.2022.00047`.
- **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "ExPAD: An

Explainable Distributed Automatic Anomaly Detection Framework over Large KGs", IEEE 17th International Conference on Semantic Computing (ICSC), 2023, pp. 204-211, `https://doi.org/10.1109/ICSC56153.2023.00040`.

- **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "Anomaly Detection for Numerical Literals in Knowledge Graphs: A Short Review of Approaches", The Sixth IEEE International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2023, pp. 46-53, `https://doi.org/10.1109/AIKE59827.2023.00015`.

3.1 KGs as input for Machine Learning

This section explores previous scientific research that employs ML and data analytics on KG data. Additionally, we present various possibilities for generating fixed-length numeric features from the multi-modal KG data. The majority of classical ML methods require fixed-length numeric feature or vectors to represent a sample's information [65, 66]. However, generating such a vector is not a trivial task since a node in a KG can have connections to an arbitrary number of other nodes, and these nodes can possess additional information. To tackle this issue, several approaches have been developed to approximate the representation of sample information. This section outlines various methods that use propositionalization [67–70], graph kernels [71–73], and KG embeddings [74–76] to generate these feature vectors.

3.1.1 SPARQL-based Propositionalzation

Before being able to run any ML technique on KGs, the KGs should be featurized (also referred to as vectorization or prepositionalization). This step generates features from KGs and prepares machine learning-friendly features for subsequent processes. There are plenty of works in the area of prepositionalization [67–70]. In this section, our aim is to provide a summary of these works.

In recent years, a variety of approaches have been proposed for generating features from LOD. Many of these methods assume a manual design of the feature selection mechanism and in most situations these methods require the user to create a SPARQL query to retrieve the features. For instance, LiDDM [67] enables the users to specify SPARQL queries for retrieving features from RDF graphs that can be used in various machine learning algorithms. However, the presence of a user-defined SPARQL query is inevitable. Similarly, an automatic feature generation approach is proposed in [68], where the user has to define the type of features in the form of SPARQL queries. Another approach, RapidMiner[1] *semweb* plugin [69] preprocesses RDF data using user-specified SPARQL queries, such that the data can be handled in RapidMiner. [77] have considered using user-specified SPARQL queries in combination with SPARQL aggregates. Another approach is FeGeLOD [78] and its successor, the RapidMiner Linked Open Data Extension [70]. FeGeLOD [78] is an open-source and unsupervised approach for enriching data with features derived from LOD. This approach uses six unsupervised feature generation techniques to explore the data and fetch the features. It uses 6 predefined SPARQL queries to extract information from KGs and comprises three subsequent steps: entity recognition, the actual feature generation, and the optional selection of a subset of the generated features.

[1] http://www.rapidminer.com/

In the context of this thesis, in Chapter 4, we developed an approach that does not require any predefined SPARQL query for extracting the features. Moreover, our approach is generic and can be scaled horizontally over a cluster of nodes to handle large amounts of data.

3.1.2 Graph Kernels

Graph kernels are mathematical functions that measure the similarity between pairs of graphs. They are widely used in machine learning and data mining applications that involve graph data, such as social networks, chemical compounds, and protein structures. Graph kernels can be categorized into different types, such as node-based, subgraph-based, and graphlet-based kernels, depending on the features they consider for similarity computation. [79] presents how graph kernels can be used to extract feature vectors that can be used in classical data analytics. However, applying graph kernel methods on KGs is computationally expensive due to the size of KGs. In this regard, there are simplified approximations that can be applied to large KGs.

For instance, subsequent research, exemplified by Deepwalk [72], introduces simplified approximation methods that enable the extraction of subgraphs within knowledge graphs. The techniques employed in Word2Vec [80] are further applied in Node2Vec [71], where subgraphs are interpreted as entities resembling sentences in a document. Deepwalk and Node2Vec are specifically designed for graphs with a single type of edges. The subgraphs correspond to sentences, and the random walks associated with each entity in the document form a collection. This transformation of the problem allows the generation of fixed-length numeric feature vectors similar to those used in Natural Language Processing (NLP). RDF2Vec [73] offers a dedicated implementation for RDF KGs, enabling native operations on Semantic Web KGs. It also handles the directed labeled edges commonly found in KGs.

In Chapter 4, we use graph kernels and random walks to create explainable feature extraction SPARQL queries. Due to the exploratory behavior of random walks, SPARQL queries can be generated without prior knowledge of the KG and without the existence of an ontology [23].

3.1.3 Knowledge Graph Embeddings (KGEs)

Another option to produce fixed-length numeric feature vectors is Knowledge Graph Embeddings (KGE). KGE is a popular technique used to represent knowledge graphs in a low-dimensional vector space. This technique maps entities and relationships in the KGs to continuous vectors in a low-dimensional space while preserving their semantic similarity. KGE has been widely used in many tasks such as entity prediction, link prediction, and relation prediction. There are various approaches for KGE such as TransE [81], SimplE [82], and DistMult [83], which are based on different assumptions and optimization objectives. Recent studies have shown that KGE outperforms traditional feature-based methods in many KG-related tasks. For instance, [84] has proposed a neural network-based approach called ComplEx, which uses complex-valued embeddings for KGs, and showed that it outperforms many state-of-the-art methods in link prediction tasks. Another study [85] proposed a novel KGE method called RotatE, which uses rotations in the embedding space to capture rich semantic interactions between entities and relations in KGs, and showed that it achieves superior performance in entity prediction tasks. TransE [81] which is the initial approach, assumes that for a triple (h, r, t), the embedding of h and r, when summed up, should be similar to the embedding of t.

Mathematically, the model tries to minimize the following scoring function for each triple in the KG:

$$||h + r - t||$$

where h, r, and t are the embeddings of the head entity, relation, and tail entity respectively, and $||.||$ represents a distance metric such as $L1$ or $L2$. The optimization objective is to minimize the margin between the score of the true triple and the scores of the corrupted triples (i.e., triples where one of the entities is replaced by another entity).

Creating KGEs for large-scale knowledge graphs poses a non-trivial challenge due to the complexities involved in parallelizing these algorithms across multiple nodes. Additionally, embeddings encounter a deficiency in explainability as they undergo a transformation of data from one space to another. In Chapter 4 we benchmarked against latent embeddings as part of the evaluation of our novel approach.

3.2 Machine Learning on Semantic Data

There are numerous machine learning frameworks and algorithms for RDF data. Below, we have highlighted several notable frameworks.

3.2.1 Statistical Relational Learning (SRL) Frameworks

TensorLog [86] and ProPPR (Programming with Personalized PageRank) [87] are recent frameworks for efficient probabilistic inference in first-order logic. TensorLog is a framework that combines logic programming and deep learning techniques to perform inference tasks on RDF data. It is specifically designed for knowledge graph completion, which involves predicting missing relationships or facts within a knowledge graph. At its core, TensorLog integrates two key components: Prolog [88], a logic programming language, and TensorFlow [89], a popular deep learning framework. This combination allows for the seamless integration of logical reasoning and neural network-based learning. Moreover, ProPPR is a probabilistic logic programming language and an inference engine that combines the expressiveness of logic programming with the efficiency of personalized PageRank algorithms. It is designed to handle large-scale knowledge graphs and perform efficient probabilistic reasoning tasks. At its core, ProPPR extends the standard Prolog language with the addition of probabilistic facts and rules. These probabilistic rules allow for the encoding of uncertainty and probabilistic dependencies in the knowledge representation. The language also introduces the notion of a "random walk" as a mechanism for performing inference. Besides this, DL-Learner [90] is a framework for automated machine learning (AutoML) and knowledge extraction from structured data. It focuses on the application of machine learning techniques to perform tasks such as ontology learning, knowledge graph completion, and semantic annotation. DL-Learner provides a flexible and extensible platform for integrating machine learning algorithms into knowledge acquisition and representation. It supports various input formats, including RDF data, ontologies, and knowledge graphs.

3.2.2 Knowledge Graph Construction and Mining

AMIE [91] and AMIE+ [92] are rule-mining algorithms designed for discovering association rules from knowledge graphs or RDF data. These algorithms are specifically developed for mining rules in the form of triple patterns $\langle$`subject,predicate,object`$\rangle$ that represent relationships between entities in the knowledge graph. The goal of AMIE and AMIE+ is to automatically discover interesting and meaningful association rules that capture patterns and dependencies within the data. These rules provide insights into the underlying structure and relationships in the knowledge graph, aiding in tasks such as knowledge discovery, recommendation systems, and data exploration. When it comes to Knowledge Graph Embedding, there are several libraries available that can be incorporated into traditional machine learning pipelines, including PyKeen [93] and Open-KE [94]. However, these libraries do not specifically prioritize distributed processing for handling large-scale KG data.

3.2.3 Distributed Graph Processing

For dealing with large KG datasets, specialized frameworks have emerged, such as PyTorch Big-Graph [95], DGL-KE [96], and Graphvite [97]. These frameworks employ distributed computing techniques to efficiently manage the memory load of KGs and their corresponding embeddings. One significant challenge in this context is optimizing KGE in a distributed manner. Since KGE tensors are spread across multiple machines, sharing and synchronizing vectors throughout the optimization cycles is crucial. Moreover, GraphX [98] is a distributed graph processing framework developed within the Apache Spark ecosystem. It provides tools for efficiently performing computations and analytics on large-scale graphs. GraphX abstracts graphs as collections of vertices and edges, to Spark's RDDs, enabling seamless integration with Spark's parallel processing capabilities. It offers specialized data structures for vertices and edges, facilitating fault-tolerant and distributed graph operations. With high-level operators and built-in algorithms, such as PageRank and connected components, GraphX simplifies the implementation of graph-based computations while allowing integration with Spark's broader data processing capabilities.

3.2.4 Distributed Linked Data Processing

Distributed data processing approaches often rely on Apache Spark, which enables the creation of structured downstream pipelines. However, Apache Spark lacks native support for knowledge graphs. Although GraphX [98] provides basic tools for graph processing, they do not operate directly on labeled RDF KGs. To address this limitation, the SANSA stack was developed, combining the distributed computing capabilities of Apache Spark with Apache Jena's RDF tools. SANSA forms a Scalable Semantic Analytics Stack, enabling distributed semantic data analytics for various use cases. Within the SANSA stack, several research projects have been undertaken, focusing on specific distributed semantic data analytics scenarios. One such project is Sparqlify [99], a framework that allows the rewriting of SPARQL queries to SQL, facilitating distributed query execution on RDF KGs within the SANSA environment. Additionally, research efforts have been devoted to conducting statistical surveys on the properties of RDF KGs, as demonstrated in the works [100, 101]. SANSA has also been utilized for experimental proof of concept projects, including clustering on linked geodata [102].

To the best of our knowledge, none of the mentioned frameworks offer the capability of anomaly detection on KGs. Moreover, due to its native functionalities, we selected SANSA as the foundation for developing this thesis. Additionally, all the proposed approaches in this thesis are integrated into the SANSA framework.

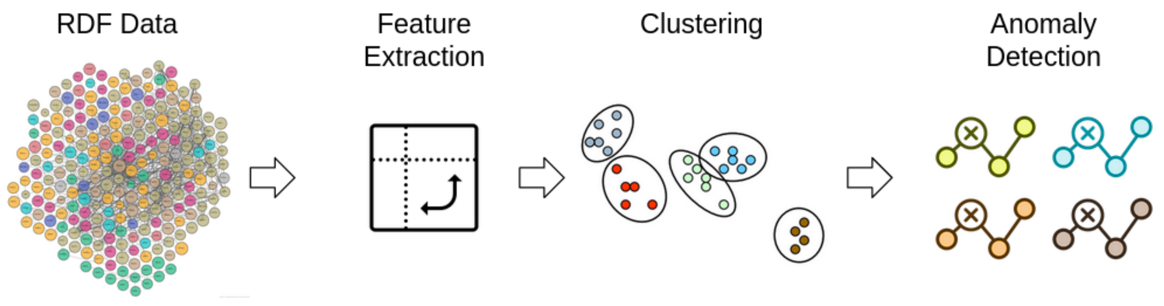


Fig 3.1: Standard anomaly detection pipeline over KGs

3.3 Anomaly Detection on KGs

Although anomaly detection is already a well-studied field with the focus specifically on the task of anomaly detection in non-relational datasets [17], to the best of our knowledge there has not been much dedicated research work, particularly on the use of anomaly detection techniques on a large RDF dataset. Therefore in this section, we discuss a few existing outlier detection methods.

Different types of errors can be hidden in the different dimensions of knowledge graphs. Most of the already existing research in this area focuses on outliers in numerical literals [78, 103–105] (except [106] which focuses on the predicates). This is influenced by the fact that outlier detection predominantly revolves around numeric data, making numeric literals a logical focal point. These methods try to find anomalies in a single literal value. That is, given one property, such as `dbo:elevation`, representing the elevation of a place, these methods want to detect anomalous values that are used as literal objects of that property. Furthermore, collecting all values of a specific attribute, such as weight, and attempting to perform anomaly detection for this attribute is conceptually incorrect in KGs. The reason is that the same predicate can be used for different entities. For example, the weight of vehicles can not be compared to the weight of animals. Therefore, to overcome this problem, the existing works use a mechanism for clustering entities before applying anomaly detection techniques. Moreover, there should be a mechanism to first extract features from KG before applying any anomaly detection techniques (explained in Section 3.1).

Figure 3.1 depicts the standard pipeline of anomaly detection over KGs and Table 3.1 summarizes the existing approaches and our proposed approaches plus their main characteristics.

To be able to achieve precise anomaly detection results and avoid natural outliers, one needs to perform clustering over entities to not compare for example the height of animals with the height of a building. In the following sections, we briefly explain the clustering approaches only used in anomaly detection over KGs.

3.3.1 Clustering by `rdf:type`

One of the early works in the area of detecting incorrect numerical data in DBpedia is [103]. The authors argued that the traditional outlier detection approaches are limited by the existence of natural outliers and performed the process of finding numerical outliers in two steps. In the first step, all types of an entity are considered as a vector of boolean values (one-hot encode), representing whether or not the entity is of a certain *type*. The authors used the FeGeLOD framework [78] for vectorizing the

Table 3.1: Existing works of AD on KGs and their characteristics

		[103]	[104]	[107]	[105]	DistAD [24]	ExPAD [25]
AD Level		Numeric Literals	Numeric Literals	Links	Numeric Literals	Numeric Literals + Number of Predicates	Numeric Literals
AD Algorithm	Univariant	✓	✓	×	✓	✓	✓
	Multi-variant	×	×	✓	×	✓	×
	Graph-based	×	×	✓	×	×	×
	Statistical	✓	✓	×	✓	✓	✓
	Supervised	×	×	×	×	×	×
	Semi-supervised	×	×	✓	×	×	×
	Unsupervised	✓	✓	✓	✓	✓	✓
	Algorithm	IQR MAD KDE	IQR LOF	GAS LOF One-ClassSVM	IQR	IQR MAD Z-Score Isolation Forest	IQR MAD Z-Score
Feature Extractor		FeGeLOD	SPARQL	Graph features	-	Literal2Feature Pivoting	Literal2Feature Pivoting
Clustering		rdf:type + Expectation Maximization	constraint + lattice	-	LHD + Cohorting	DistSim + Bisecting Kmeans	Decision Tree
Scalability		×	×	×	✓	✓	✓
Explainability		×	×	×	×	×	✓
Streaming		×	×	×	×	×	×

entities and did the clustering with the Estimation Maximization (EM) algorithm [108], using the implementation in WEKA [109]. In the next step, the outliers are detected. The authors have compared different outlier detection techniques, such as IQR [60], KDE [62], and dispersion estimators, and reported that the IQR performs the best. In addition, they reported that the run-time on datasets containing only two properties- *DBpedia-owl:populationTotal* and *DBpedia-owl:elevation* is over 24 hours due to the slow clustering algorithm.

Furthermore, [106] introduced an innovative unsupervised multi-dimensional anomaly detection methodology designed to identify erroneous links within RDF datasets. In pursuit of identifying inaccurate links through outlier detection, they used a multi-step process. Initially, they transformed each link into a feature vector, by considering both direct types and all incoming and outgoing properties of resources within the interconnected datasets. Their approach contained three stages: (i) the extraction of a link set, accompanied by the creation of a distinct feature vector representation for each link; (ii) the execution of outlier detection on the ensemble of generated vectors, and assigning an outlier score to each individual link; and (iii) sorting the links in descending order based on their respective outlier scores. In the course of their experimentation, they leveraged the capabilities of the RapidMiner[2] platform. Also for anomaly detection, they used the k-NN global anomaly score (GAS), the Local Outlier Factor (LOF), and One-Class Support Vector Machines. Although they reported encouraging outcomes, it is worth noting that their methodology is not scalable and has not applied to authentic Linked Open Data (LOD) datasets.

3.3.2 Clustering with Constraints

This approach is introduced in [104] which is close to [103]. This outlier detection method cross-checks the results of outliers by exploiting the *"sameAs"* properties in the knowledge graph. Outlier detection is accomplished through dataset inspection using specialized SPARQL queries against the knowledge graph. The authors begin by selecting the interesting properties for outlier detection. The sub-population is generated in the second step by applying a set of constraints (top-down ILP algorithms) to classes, properties, and property values. This exploration is laid out as a lattice, with the root node consisting of a property and the number of instances that correspond to it. After the lattice has been generated, the outliers on all unpruned nodes of the lattice must be found. The outlier score results are saved as a set of constraints that returns the corresponding instance set. Outliers are classified as natural or real using the data interlinking property and comparison with different datasets. This procedure improves the handling of natural outliers, lowering the false positive rate.

3.3.3 Clustering by LHD

As for some entities, the `rdf:type` information could be missing, [105] introduced a new way of clustering based on Linked Hypernyms Dataset (LHD) [110]. LHD contains types from the DBpedia namespace that have been extracted from the opening sentences of Wikipedia articles written in various languages. These types were identified using Hearst pattern matching on part-of-speech annotated text and disambiguated to correspond with DBpedia concepts. Moreover, they used Locality Sensitive Hashing (LSH) [111] for creating clusters (they called the clusters cohorts because the same data can appear in more than one cluster). LSH is an important class of hashing techniques that hashes data points into buckets, so that the data points that are close to each other are in the same buckets with

[2] http://www.rapidminer.com

high probability, while data points that are far away from each other are likely in different buckets. [105] used `rdf:type` and LHD information to create vectors and then hashed it based on LSH and further generated cohorts. This approach was only applicable to DBpedia and the accuracy of it was questionable because the same data could appear in multiple cohorts which could result in natural outliers.

As shown in Table 3.1, the majority of existing works face scalability issues, making them impractical for large-scale knowledge graphs. In response to this challenge, we introduced DistAD (Chapter 5). DistAD is versatile and applicable to any RDF data; it is scalable and capable of execution on very large KGs; and it is modular and configurable to cater to different use cases.

3.4 Explainable Anomaly Detection on KGs

As the use of anomaly detection algorithms in critical safety areas becomes more common, there's a growing need for explanations behind important decisions made within these areas, which is both an ethical requirement and a regulatory demand. However, the anomaly detection field has mainly focused on accuracy in identifying anomalies, often neglecting to explain why those decisions are reached. For example, [112] criticizes the prevailing practice of algorithms simply providing rankings for anomalies without giving users explanations for why certain data points deviate. Similarly, [113] points out that while many techniques are excellent at identifying global and local anomalous patterns, they often overlook the crucial aspect of understanding why certain instances are considered outliers. Additionally, [114] notes that only a few studies on outlier detection make an effort to offer insights that help clarify the nature of instances that stand out. Furthermore, [115] claims that present outlier detection systems frequently lack explanations for why specific cases are tagged as outliers, failing to highlight the distinguishing characteristics that define them as such.

According to [116], the terms "interpretability" and "explainability" refer to the ability to make sense to humans using clear language. Furthermore, [117] defines Explainable Artificial Intelligence (XAI) as AI that provides details or reasoning to make its operations understandable to a specific audience. [118] goes on to describe interpretable or eXplainable Machine Learning (XML) as extracting relevant insights from a machine learning model about inherent data relationships or lessons learned through the model's learning process. This knowledge is important if it helps the target audience understand the issues at hand. As a result, eXplainable Anomaly Detection (XAD) refers to extracting relevant insights from an anomaly detection model about inherent data relationships or lessons learned from the model's learning process, as long as this knowledge sheds light on the anomaly detection problem under investigation by the end user.

The process of analyzing anomalies involves two equally crucial tasks: (i) detecting anomalies (as explained in Section 3.3) and (ii) explaining them. Anomaly explanation involves clarifying why a particular anomaly is labeled as such. Since the terms "anomaly" and "outlier" are often used interchangeably, anomaly explanation is also known as outlier explanation, outlier interpretation, outlier description, and outlier characterization.

When an anomaly is identified by an anomaly detection algorithm, XAD aims to make the anomaly understandable by enhancing the interpretability of the anomaly detection method. There are various ways to achieve interpretability in an anomaly detector. When the anomaly detector is inherently interpretable (e.g. logistic regression, shallow decision trees, rule-based models, etc.), it's relatively easy to understand why a flagged anomaly is considered unusual. On the other hand, when the anomaly detector lacks inherent interpretability (e.g. Isolation Forest [64], RNN [119], CNN [120]), post-hoc explainable AI techniques like LIME [121], Anchors [122], and SHAP [123] can be used to interpret the anomaly detector and elaborate on the reasoning behind its decisions.

In general, there are three categories of approaches for making anomaly detection explainable: *pre-model techniques*, *in-model techniques*, and *post-model techniques*. Pre-model techniques are implemented before the anomaly detection process, such as filter feature selection methods. In-model techniques use inherently explainable models, allowing for easy explanations during anomaly detection, like linear regression-based anomaly detection methods that can reveal feature coefficients. In contrast, post-model techniques, also known as post-hoc techniques, strive to explain the decisions made by an anomaly detection model after the model is built and implemented. For example, SHAP-based interpretation methods [124] fall under this category.

3.4.1 Pre-Model Techniques

In Pre-model techniques, the aim is to make models simpler and more accurate in detecting anomalies by having meaningful and relevant features while removing unnecessary ones. This helps to understand the data relationships better. For instance, in [125], it's highlighted that the effort needed to investigate an anomaly increases with the number of features describing it. To simplify this, techniques like dimensionality reduction can be used. This involves methods such as feature projection and feature selection, which help decrease the number of features describing an object. This makes explaining anomalies easier. However, methods like Principal Component Analysis (PCA) transform original features into a new set, making them harder to interpret. On the other hand, feature selection keeps the most important original features, enhancing interpretability and overcoming the challenge of dealing with high-dimensional data.

There are a handful of unsupervised feature selection methods for anomaly detection. Specifically, CBRW_FS [126] and CBRW [127] are two unsupervised selection methods. These methods choose a subset of features without considering the subsequent anomaly detection techniques, but they work only with categorical data by modeling the connections between feature values. Moreover, [128] devise an optimization framework for concurrently selecting features and instances in categorical data anomaly detection by assuming strong similarities among uncommon instances. Nevertheless, this assumption is commonly not met, as anomalies tend to be isolated and distinct from each other.

Meanwhile, [129] and [130] try to identify a relevant feature subset for anomaly detection by exploring correlations between features. They assume that anomalies are instances that deviate from normal feature dependencies, so only features connected to others are considered important. However, this definition of anomalies doesn't work well with many standard anomaly detection methods. Additionally, Isolation Forest [64] can also be used to select features for anomaly detection. A method based on Isolation Forest, called IBFS [131], selects features that contribute the most to the outlierness of anomalies detected by Isolation Forest.

3.4.2 In-Model Techniques

In the realm of In-Model techniques, the models used for anomaly detection offer insights into the relationships they've learned from the data. This empowers end-users to grasp the reasoning behind the decisions made. In general, methods which are using transparent models such as Linear Models

(Linear Regression, Logistic Regression), Decision Trees, Gaussian Processes, Rule-based Learners, Generative Additive Models, and Bayesian Models, are inherently explanatory.

For instance, [132] employs frequent pattern mining to identify and explain anomalies in transactional data. They specifically utilize the Apriori algorithm [133] to uncover frequent patterns, using the most contradictory frequent patterns to elucidate each identified anomaly. Similarly, [134] proposes a model to capture common motion and background patterns in video data, treating deviations from these patterns as anomalies. Likewise, [135] introduces the DRGMiner model, which mines frequent patterns in dynamic graphs and treats graphs that deviate from these patterns as anomalies.

Moreover, decision trees and their variations have been suggested for anomaly detection, yielding explanations inherent to the detection outcomes. For example, [136] introduces the Composition-based Decision Tree (CDT) to identify and interpret anomalies in time series data. This involves constructing a CDT after preprocessing and labeling time series data. The CDT extends a decision tree to this labeled data, extracting rules that describe observed anomalies and identify novel ones. The quality of explanations is evaluated based on the number of utilized patterns and rule lengths. Additionally, [137] presents an anomaly detection approach involving supervised decision tree splits on features. Confidence intervals are created for each branch's target feature, enabling explanations from branch conditions and general distribution statistics of non-anomalous instances falling into the same branch. Furthermore, [138] proposes the Decision Tree-based AutoEncoder (DTAE) model for anomaly detection. This model employs a decision tree to illustrate the encoding and decoding steps of an AutoEncoder (AE), determining anomalies by comparing the input to the output.

3.4.3 Post-Model Techniques

Post-model techniques examine an anomaly detection model after the detection process is finished, or they may examine a specific anomaly without requiring a pre-existing anomaly detection model. Essentially, these methods do not influence the anomaly detection process itself, instead, they work by analyzing the connection between the input provided to the anomaly detection model (if applicable) and the resulting output. Within this category, approaches like LIME, Anchors, SHAP, and RESP can be employed.

LIME (Local Interpretable Model-agnostic Explanations) [121] is a model-agnostic interpretability technique designed to explain the predictions of complex machine learning models. The primary objective of LIME is to provide local explanations, offering insights into how a model arrives at specific predictions for individual instances or samples. LIME operates by approximating the behavior of the underlying model in the local vicinity of a particular data point. It creates a simplified, interpretable model, such as linear regression or decision trees, around the instance of interest. This simpler model is trained using perturbed versions of the original instance, ensuring a diverse set of samples that cover the relevant feature space. Once the interpretable model is trained, LIME calculates the feature importance or contribution of each input feature to the prediction. By examining the coefficients or feature weights in the interpretable model, LIME provides insights into which features have the most influence on the model's decision-making process for that particular instance.

In [122] authors improved upon LIME by replacing its linear model with a logical rule for explaining a data instance. It introduces a rule-based method to generate easily understandable, if-then-like explanations that capture the conditions under which a particular model's prediction holds true. It aims to identify the minimal set of feature conditions, called anchors, that are both sufficient and necessary for a prediction. Anchors can be derived through a combination of optimization and sampling techniques, ensuring both interpretability and accuracy.

SHAP (SHapley Additive exPlanations) [123] is a unified framework for explaining the predictions of machine learning models. It leverages concepts from cooperative game theory to assign importance values to each feature in a prediction, capturing their contribution to the overall prediction. SHAP provides a comprehensive and consistent way of attributing feature importance that satisfies several desirable properties, including fairness, consistency, and local accuracy. By considering all possible combinations of features, SHAP computes the Shapley values, which quantify the marginal contribution of each feature to the prediction. These values allow for a nuanced understanding of the impact of individual features and their interactions on the model's output.

Moreover, RESP [139] focuses on providing explanations for the outcomes of classification models by considering causal relationships. This model aims to go beyond traditional feature importance techniques and delve into the underlying causal mechanisms that contribute to the predictions. The model operates under the assumption that causality plays a significant role in the decision-making process of classification models. It analyzes the dependencies and causal relationships between the features used for classification, considering both direct and indirect causal effects. To explain the classification outcomes, the model applies causal inference methods, such as Bayesian networks or structural equation modeling, to uncover the causal structure within the data. By identifying the causal relationships, the model can determine which features have a direct influence on the predicted outcome and which features impact the outcome indirectly through other variables.

3.5 Summary

In this section, we tried to provide a review of the related work. In particular, we focused on four main areas: KGs as input for machine learning techniques, machine learning frameworks that operate on KGs, existing approaches to anomaly detection on KG data, and explainable AI.

Through our investigation, we discovered that Knowledge Graph Embeddings encounter challenges in achieving effective anomaly detection due to scalability issues and a lack of interpretability. In response, we introduced Literal2Feature (detailed in Chapter 4) to specifically address these concerns. Furthermore, we identified shortcomings in existing approaches and frameworks for anomaly detection on knowledge graphs, particularly in scalability and explainability. Consequently, we put forth DistAD (covered in Chapter 5) and ExPAD (discussed in Chapter 6) as solutions to these challenges.

CHAPTER 4

Scalable and Distributed Feature Extractor

Acknowledgement This chapter is based on our scientific publication: **Farshad Bakhshandegan Moghaddam**, Carsten Felix Draschner, Jens Lehmann and Hajira Jabeen, "Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor", SEMANTICS, 2021, IOS Press, pp. 74–88. `https://doi.org/10.3233/SSW210036`.

4.1 Motivation

With the rapidly growing amount of data available on the Internet, it becomes necessary to have a set of tools to extract meaningful and hidden information from online data. The Semantic Web is able to form a structural view of the existing data on the web and provides machine-readable formats [1]. In order to enable this, the Resource Description Framework (RDF)[1] has been introduced by the World Wide Web Consortium[2] as a standard to model the real world in the form of entities and relations between them. RDF data are a collection of triples ⟨`subject,predicate,object`⟩ which tend to have rich relationships, forming a potentially very large and complex RDF graph. Figure 4.1 shows a sample RDF graph.

Currently, many companies in the fields of science, engineering, and business, including bioinformatics, life sciences, business intelligence, and social networks publish their data in the form of RDF. In addition, the Linked Open Data Project initiative [2] also helped the Semantic Web to gain more and more attention in the past decade. Currently, the Linked Open Data (LOD) cloud comprises more than 10,000 datasets available online[3] using the RDF standard. Nowadays, RDF data can have sizes up to billions of triples[4].

Besides this, Machine Learning, a field of discovering how machines can perform tasks without being explicitly programmed to do so, is growing and finding its way in human daily life. Some prominent examples are autonomous driving, face detection, weather forecasting, etc. Recently with the rapid growth of computational power, training machine learning algorithms at scale is getting much more feasible. However, most of the well-known machine learning algorithms for classification,

[1] https://www.w3.org/RDF/
[2] https://www.w3.org
[3] http://lodstats.aksw.org/
[4] https://www.w3.org/wiki/DataSetRDFDumps

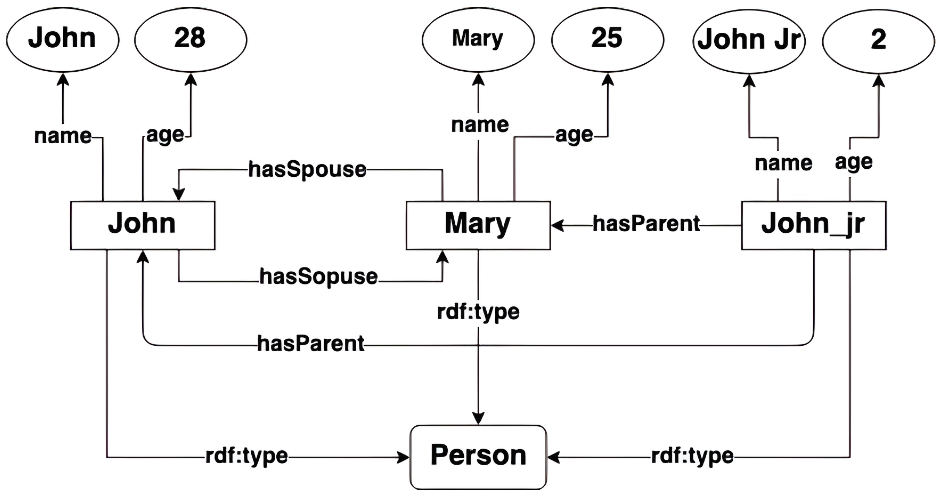


Figure 4.1: Literal2Feature: a sample RDF graph

regression, and clustering need to work with a standard shape of data, i.e. a *feature matrix*. In this format, the data is mostly presented as a 2D matrix, in which rows present the data points and columns indicate the features. Normally, for supervised learning, one (or more) of the columns can be considered as a label (target) for the given row (data point).

Due to the complex graph nature of RDF data, applying standard machine learning algorithms to this data is cumbersome. Although there are efforts in the community to incorporate RDF graphs directly in the machine learning algorithms, they are mostly focused on the structural properties of RDF datasets [29, 87, 90, 91] and offer limited support for RDF literals. Moreover, the challenges in the current big data era (limited computational resources) cause analytical approaches to mostly fail to operate on large-scale data.

Even though Knowledge Graph Embeddings (KGE) are getting popular as a paradigm to obtain low-dimensional feature representations of knowledge graphs, most of them do not exploit literals in their learning process (an exception is [140] for numerical literal values). Moreover, the feature vectors obtained by KGE models are latent features, which are not explainable.

To tackle the aforementioned issues, we propose Literal2Feature, a generic, distributed, and scalable software framework that is able to automatically transform a given RDF dataset to a standard feature matrix (also dubbed *Prepositionalization*) by deep traversing the RDF graph and extracting literals to a given depth. Literal2Feature enables the use of a wide range of machine learning algorithms for the Semantic Web community. The proposed method is able to extract features automatically by creating a SPARQL query to produce the feature matrix. All steps are performed automatically without human intervention (details in Section 4.2). In addition, Literal2Feature is integrated into the SANSA stack [19] and interacts with the different SANSA computational layers. In contrast to KGEs, our proposed approach successfully utilizes literals as extracted features and provides high-level explainability for each feature. Moreover, our approach enables ML practitioners to extract explainable features and to select the features of interest, based on their objectives and scenarios.

In this chapter, we address the following research question:

RQ1: Can we vectorize knowledge graphs in a scalable and distributed manner?

To summarize, the main contributions of this chapter are as follows:

- Introducing a distributed generic framework that can automatically extract semantic features from an RDF graph
- Integrating the approach into the SANSA stack
- Covering the code by unit tests, documenting it in Scala docs, and providing a tutorial
- Making the code and the framework open source and publicly available on GitHub[5]
- Evaluation of the results over multiple datasets on classification and clustering scenarios, and comparing it with similar approaches
- Empirical evaluation of scalability

The rest of the chapter is structured as follows: Literal2Feature, workflow, and implementation are detailed in Section 4.2. The use cases are discussed in Section 4.3. Section 4.4 covers the evaluation of the Literal2Feature and contains an empirical evaluation for scalability. Finally, we summarize our work in Section 4.5.

4.2 Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor

In this section, we present the system architecture of Literal2Feature. The main goal of the framework is to retrieve literals for each entity based on the predefined graph depth. In other words, Literal2Feature, by deeply traversing the RDF graph, is able to gather literals for each entity up to a predefined level and consider them as a feature vector for the given entity. We believe that literals contain valuable information for each entity which can be used for any subsequent machine learning pipeline.

Figure 4.2 shows the high-level system architecture overview. The core section of the framework consists of five main components: 1) Seed Generator, 2) Graph Search, 3) Path Extractor, 4) SPARQL Generator, and 5) SPARQL Executor. Below, each part is discussed in more detail.

4.2.1 Components

Seed Generator Component

The input of the system is an RDF graph $\mathcal{G}$ and a set of RDF triples $\mathcal{T}$ which defines the entities that the user is interested in generating features for. $\mathcal{T}$ will automatically formulate a `SELECT` SPARQL query which is used to generate only starting points for the deep search (seeds). In other words, this query specifies the entities for which the user is interested in extracting features (e.g. in Figure 4.1, all the persons). By executing the `SELECT` query over the given RDF data, the starting points of the search are generated. These seeds are sorted based on the number of outgoing links. The higher the

[5] https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1_DistRDF2ML

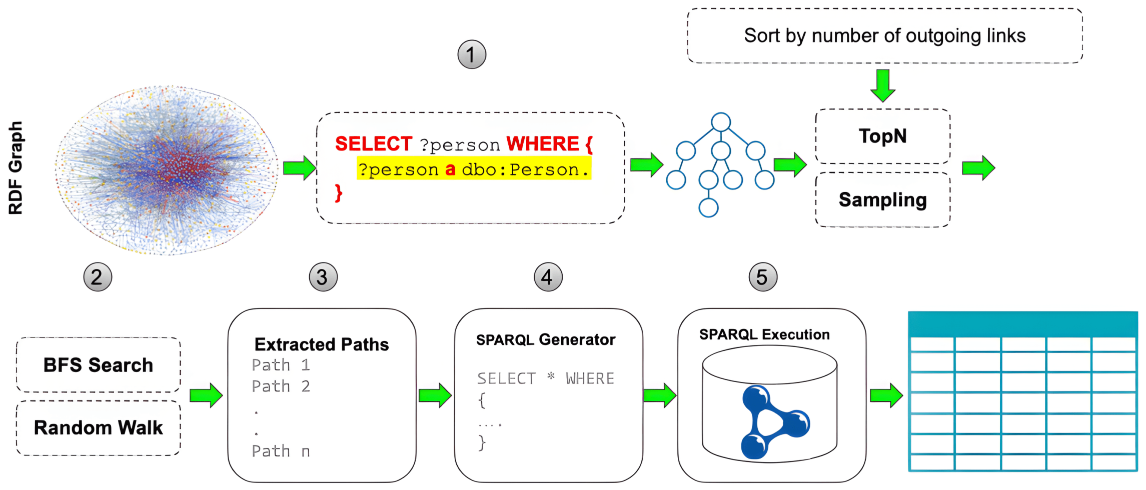


Figure 4.2: Literal2feature system architecture abstract overview

number of outgoing links, the higher the chance of generating more features. To execute the query, Sparklify [141], a SANSA built-in distributed SPARQL executor is used.

Graph Search Component

The next step executes a graph search algorithm starting from the seed nodes. Without loss of generality, in this framework, we use two different strategies, (i) full graph search, and (ii) approximated graph search. For full graph search, we use Breadth-First Search (BFS) [142] to be able to traverse the entire graph. Moreover, for the approximation, we use the Random Walk model [143]. In both approaches the user will have full control over the depth, the number of walks, and the direction of the search (downward, upward). There is a relation between extracted feature completeness and the search execution time with the search strategy. Full graph search is able to extract all the existing features, but the execution time is higher than the random walk model. Three different factors that have an impact on the number of extracted features and the execution times are (i) the depth of search, (ii) the average branching factor of nodes of the graph, and (iii) the number of random walks (only in the Random Walk model). Figures 4.3, 4.4, 4.5, and 4.6 depict the impact of each factor. It can be seen that most of the parameters have an exponential impact on the full graph search. However, the Random Walk model depicts a linear and logarithmic behavior in terms of time and the number of extracted features.

Each search walk (regardless of the selected search method) generates a path and continues till one of the following conditions occurs:

- Reaching a node which is a literal node
- Exceeding the pre-configured length of the walk

Path Generator Component

By considering the nodes and edge labels (RDF entities and properties) we could generate properties paths. Each properties path encodes all the needed properties to reach a literal from the given seed

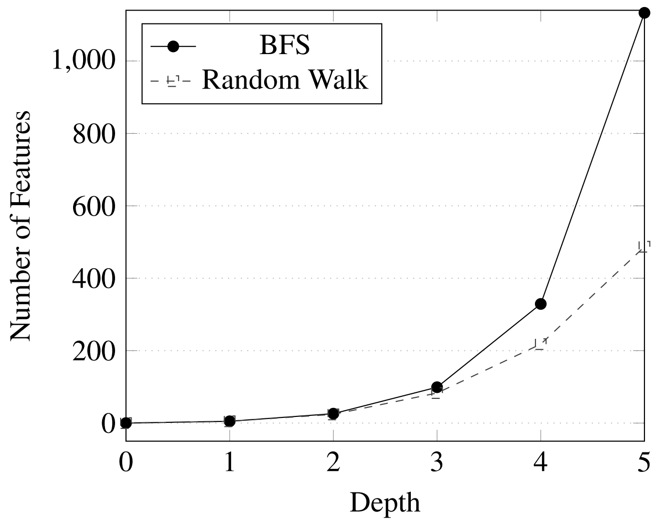


Figure 4.3: Literal2Feature evaluation: depth vs. #features on the Engie dataset

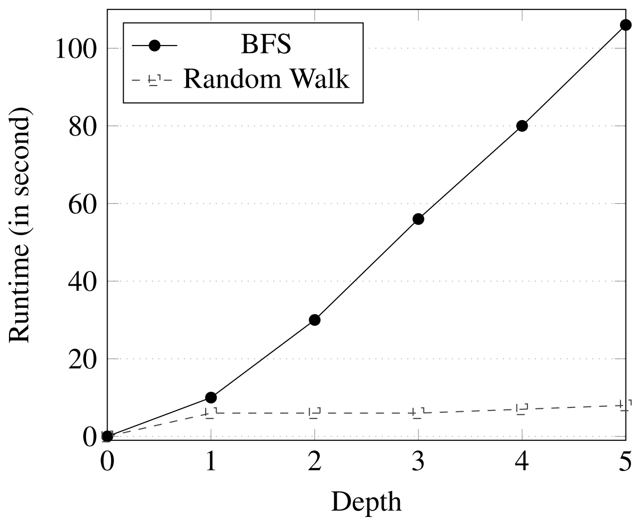


Figure 4.4: Literal2Feature evaluation: depth vs. time on the Engie dataset

node. For example, based on Figure 4.1, `John_jr->`*`hasParent`*`->Mary->`*`age`* encodes the path which can fetch the "age of the mother of `John_jr`". Each walk of the search algorithm generates a properties path. Due to the probabilistic nature of the algorithms (in the Random Walk method) and repetition in the RDF graphs, there is a chance of having duplicated paths. The final output of this component is distinct properties paths.

SPARQL Generator Component

By gathering all the properties paths, ignoring the entities, and only keeping properties, each path is transformed automatically to a SPARQL query. For instance, the above-mentioned example will be transformed to `->`*`hasParent`*`->`*`age`* and then to the following SPARQL query (Listing 4.1,

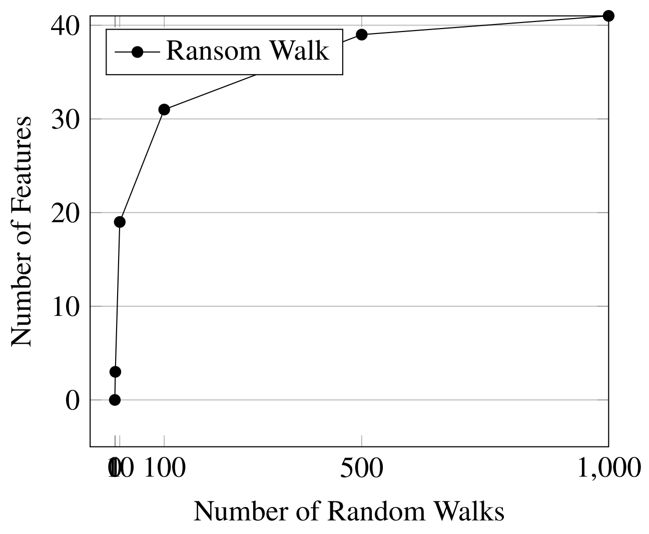


Figure 4.5: Literal2Feature evaluation: #walks vs. #features on the Engie dataset

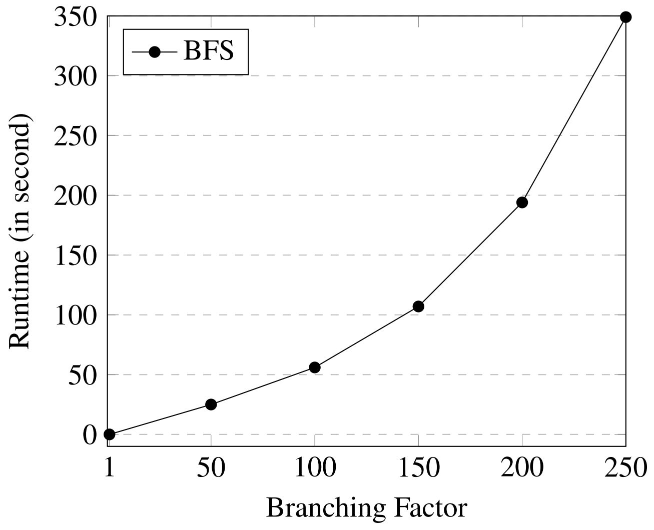


Figure 4.6: Literal2Feature evaluation: branching factor vs. time on synthetic data on a single machine

prefixes have been omitted for simplicity). Due to the data sparsity issue and as all the seeds may not have all the possible properties, each properties-path is wrapped with an `Optional` block to ensure the successful generation of the final result. To have explanatory names for the projected variables, the prefix part of each RDF property is omitted and after splitting based on the underscore character, the last part of each property is selected and concatenated (see Listing 4.1). These names are human-readable and convey a high level of explainability for each feature.

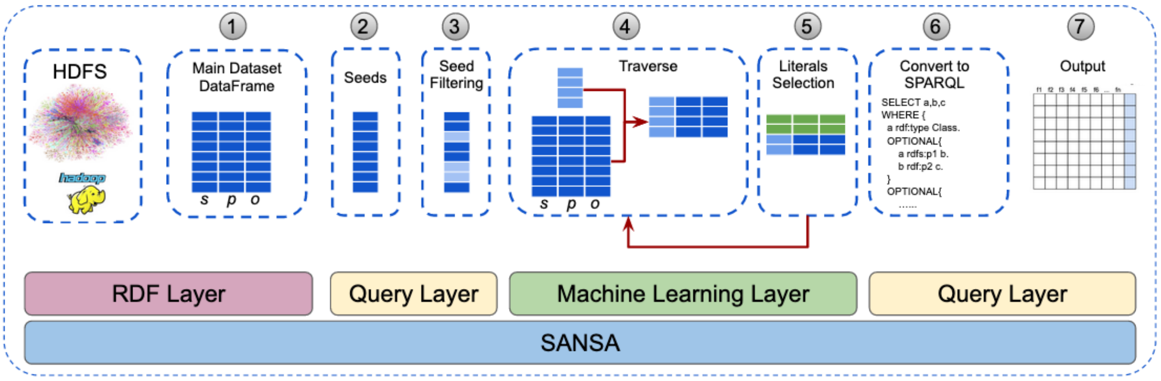


Figure 4.7: Literal2Feature execution pipeline (best viewed in color)

SPARQL Executor Component

After generating the SPARQL query based on the generated property paths, we use the built-in SPARQL engine in SANSA to execute the query over the given RDF data. The result of this component is the desired feature matrix. Due to the structure of the graph, the SPARQL query may result in multiple rows for a single entity. Based on Figure 4.1, this behavior can be seen when we want to extract the age of a parent of `John_jr` as he has two parents. In such a case, one of the rows is selected randomly and kept for the subsequent machine learning pipeline.

Listing 4.1: Literal2Feature: Sample result of the generated SPARQL query

```
SELECT ?person ?hasParent_age WHERE {
?person a Person.
 OPTIONAL {
   ?person hasParent ?parent.
   ?parent age ?hasParent_age.
 }
}
```

4.2.2 Implementation

As the programming language of SANSA is Scala[6], we have selected this language and its APIs in Apache Spark to provide the distributed implementation of Literal2Feature. Moreover, we benefit from SANSA IO and Query layers. Technically, Literal2Feature can be divided into the following steps 1) Reading RDF data as a data frame, 2) Generating seeds, 3) Filtering seeds, 4) Traversing the graph by joining data frames up to a certain depth, 5) Selecting paths which end with literals, 6) Converting the paths to SPARQL, and 7) Executing the SPARQL and output the result, as shown in Figure 4.7 which depicts the framework execution pipeline. Moreover, Algorithm 1 demonstrates the working of Literal2Feature that takes RDF triples as input, creates a data frame, applies a series of transformations and actions, and returns a feature matrix as an output. The *traverse* function which deeply searches the graph is explained in Algorithm 2.

[6] https://www.scala-lang.org/

Algorithmus 1 : Literal2Feature Algorithm

```
Data : input: an RDF dataset, config: configuration from Table 4.1
Result : Feature Matrix
1  val graph: Dateframe = spark.read.rdf(lang)(input).cache();
2  val seedQuery: String = generateSeedQuery(config);
3  var seeds: Dataframe = executeSparql(seedQuery,graph);
4  seeds = sort(seeds,DESC);
5  seeds = sampleSeeds(config);
6  val paths: Dataframe = traverse(seeds,graph);
7  val finalQuery = generateSparql(paths);
8  return executeSparql(finalQuery,graph);
```

Algorithmus 2 : Graph Traverse Algorithm

```
Data : seeds: filtered seeds, graph: Dataframe of RDF data
Result : A dataframe which contains all the information regarding the paths
1  var withOpenEnd: DataFrame = seeds;
2  var withLiteralEnd: DataFrame = spark.emptyDataFrame;
3  var currentPaths = seeds;
4  for i = 0 to (iteration-1) do
5      val left: DataFrame = currentPaths;
6      val right: DataFrame = graph;
7      var joinedPaths = left.join(right);
8      if search mode == RandomWalk then
9          joinedPaths = joinedPaths.sample(NumberOfRandomWalks);
10     currentPaths = joinedPaths.where(ends in entity);
11     currentPaths = removeCycles(currentPaths);
12     val finalPaths = joinedPaths.where(ends in literals);
13     if finalPaths.size > 0 then
14         withLiteralEnd =withLiteralEnd.union(finalPaths);
15     else
16         break;
17 return withLiteralEnd;
```

Table 4.1: Literal2Feature: Main parameters of the framework

Parameter	Description	Value Data Type
Input Data	Path of the input file or folder	`string`
Output Data	Path of the output file	`string`
Search Depth	To which depth going deep	`integer`
Search Depth Up	To which level going up	`integer`
Number of Random Walks	Number of random walks	`integer`
Number of nodes (seeds)	Number of starting nodes for search	`integer`
Ratio of nodes (seeds)	Percentage of starting nodes for search	`double`
Search Algorithm	BFS or RandomWalk	`Enumeration`
Seed Selector	List of triples which defines the seeds	`string`
Seed Variable	Variable name for seeds in the SPARQL result	`string`

Regarding the configuration, Table 4.1 shows the main configurable parameters of Literal2Feature. Moreover, a working sample can be found in SANSA-Notebook[7] which is an interactive Spark Zeppelin Notebook[8] for running SANSA-Examples via docker[9].

4.3 Use Cases

Literal2Feature is a generic tool that can be used in many use cases. To validate this, we develop use case implementations in several domains and projects.

PLATOON Digital PLAtform and analytic TOOls for eNergy (PLATOON[10]) is a Horizon 2020 Project aiming to deploy distributed/edge processing and data analytics technologies for optimized real-time energy system management in a simple way for the energy domain. PLATOON will use SANSA stack as a generic data analytics framework in which Literal2Feature is an integral module in the pipeline. Literal2Feature makes it possible for non-semantic experts to deduce features and enrich their use case-related data.

Engie Engie SA[11] is a French multinational electric utility company that operates in the fields of energy transition, electricity generation and distribution, natural gas, nuclear, renewable energy, and petroleum. Together with Engie, we worked on a dataset related to accidents that occurred in France. One of the major challenges is the prediction and classification of accidents in an effective and

[7] https://github.com/SANSA-Stack/SANSA-Notebooks/tree/stack-merge/sansa-notebooks
[8] https://zeppelin.apache.org/
[9] https://www.docker.com/
[10] https://cordis.europa.eu/project/id/872592/de
[11] https://www.engie.com

Table 4.2: Literal2Feature: Dataset statistics (GT=Ground Truth)

Dataset	**Format**	**#Triples**	**|GT|**	**Classification Scenario**	**Classes**
Accident	N-Triple	5,961,107	57,783	How dangerous is an accident?	4
				Which side of vehicle is shocked?	10
Carcinogenesis	OWL	74,567	298	Is a drug carcinogenesis?	2

scalable manner. In order to perform this task efficiently and effectively, Literal2Feature integrated into SANSA stack became an integral module to solve classification.

4.4 Experimental Results

In this section, we present two sets of experiments to analyze different aspects of Literal2Feature. In the first experiment, the quality and usefulness of the extracted features will be analyzed over classification and clustering scenarios, and in the second experiment, the scalability of the proposed framework will be investigated.

4.4.1 Experiment A: Assessment of the extracted Features

Literal2Feature is a generic tool that can be used in many scenarios, from statistical analysis to classification and clustering. In this section, we will analyze the quality of the extracted features for classification and clustering scenarios. To this end, two datasets have been exploited. Engie accident dataset[12] and Carcinogenesis dataset [144]. The accident dataset contains the accident data that occurred in France in 2018. Besides, the Carcinogenesis dataset contains information about drug molecules and their features. An overview of datasets is given in Table 4.2.

Classification

For the above-mentioned datasets, three classification scenarios have been defined as follows:

1- Accident classification based on how dangerous was an accident (4 classes)

2- Accident classification based on which side of the vehicle was shocked in the accident (10 classes)

3- Carcinogenic drugs classification (2 classes)

As baselines, FeGeLOD [78], RDF2Vec [145], TransE [81], SimplE [82], and DistMult [83] have been selected (KGEs has been trained by OpenKE[13] on NVIDIA GeForce GTX 1080 Ti). Moreover, for the learning algorithms, we selected Random Forest (RF), Logistic Regression (LR), Multi-Layer Perceptron (MLP), and XGBoost (XG) [146] to cover the full spectrum from decision trees to neural networks. As the accident dataset has imbalanced labels, only the F1-measure is reported. The results

[12] Can not be publicly published due to Intellectual Property concerns

[13] https://github.com/thunlp/OpenKE

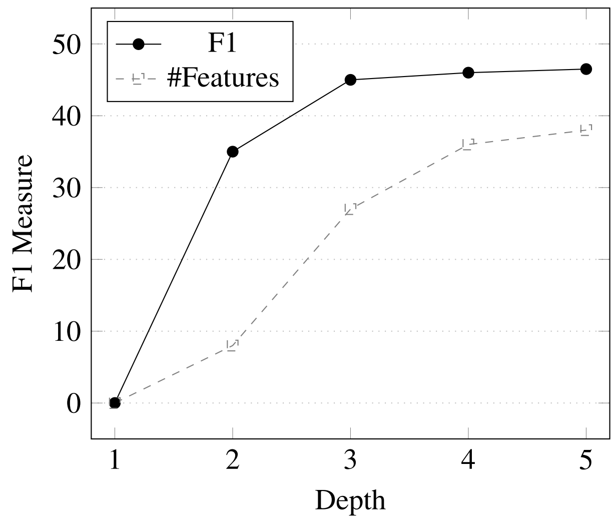


Figure 4.8: Literal2Feature evaluation: depth impact on the classification Result and number of extracted features

are summarized in Table 4.3. These experiments have been conducted on a single node with an Intel Core i5 CPU and 8GB of RAM. For FeGeLOD its original default configurations have been preserved and for KGEs the dimension of vectors has been set to 200. Moreover, all the algorithms have been trained with 5-fold cross-validation.

The focus of this experiment is on showing the quality of the extracted features and comparing it with other approaches. As can be seen in all cases the extracted features from our proposed framework yield a higher F1 score. For example, in Accident Scenario 1, Literal2Feature achieves 0.46 in comparison to 0.37 for DistMult. The same behavior can be observed in Accident Scenario 2. Here Literal2Feature achieves 0.21, however, the F1 of all the other baselines is less than 0.09. In the Carcinogenesis binary classification scenario, Literal2Feature still outperforms other methods. Although in this case, FeGeLOD results are comparable, however, it should be mentioned, that in this specific case, FeGeLOD could achieve 0.61 with 247 extracted features, however, Literal2Feature achieved 0.62 with only 8 extracted features. So this fact reveals that the extracted features by Literal2Feature are much more informative.

As a hypothesis, the further we go deeper in the graph, the more likely to find features, but the less trustworthy and relevant the features become. To prove it, we run the classification scenario on the Engie dataset with 4 classes (Accident Scenario 1) with MLP algorithm for the features in different depths. As shown in Figure 4.8, there is no significant change in F1 after step 3. It indicates that considering the features till depth 3 is sufficient and there is no need to consider further features as it can increase the running time of the system.

Clustering

To show the usefulness of the extracted features, we designed a clustering scenario as well. However, as our datasets have no ground truth for the clustering scenario, we selected the Silhouette Coefficient to measure the quality of the different clustering methods. The Silhouette Coefficient is defined for each sample and is composed of two scores: (i) the mean distance between a sample and all other

Table 4.3: Literal2Feature: The F1-Measure evaluation results

Approach		**Accident Scenario 1**	**Accident Scenario 2**	**Carcinogenesis**
FeGeLOD [78]	*RF*	0.17 ± 0.01	0.05 ± 0.001	0.59 ± 0.07
	LR	0.20 ± 0.02	0.05 ± 0.003	0.61 ± 0.06
	MLP	0.18 ± 0.01	0.05 ± 0.001	0.58 ± 0.07
	XG	0.18 ± 0.02	0.05 ± 0.004	0.58 ± 0.05
RDF2Vec[145]	*RF*	0.17 ± 0.004	0.05 ± 0.0001	0.41 ± 0.13
	LR	0.25 ± 0.02	0.07 ± 0.006	0.49 ± 0.12
	MLP	0.23 ± 0.02	0.09 ± 0.008	0.51 ± 0.18
	XG	0.23 ± 0.02	0.07 ± 0.006	0.42 ± 0.10
TransE [81]	*RF*	0.16 ± 0.001	0.05 ± 0.0001	0.52 ± 0.09
	LR	0.17 ± 0.002	0.05 ± 0.001	0.56 ± 0.06
	MLP	0.24 ± 0.006	0.08 ± 0.001	0.56 ± 0.06
	XG	0.24 ± 0.005	0.06 ± 0.001	0.51 ± 0.06
DistMult [83]	*RF*	0.22 ± 0.01	0.05 ± 0.0001	0.50 ± 0.06
	LR	0.22 ± 0.005	0.05 ± 0.001	0.53 ± 0.06
	MLP	0.26 ± 0.005	0.09 ± 0.004	0.54 ± 0.04
	XG	0.37 ± 0.04	0.09 ± 0.002	0.58 ± 0.05
SimplE [82]	*RF*	0.22 ± 0.01	0.05 ± 0.0001	0.45 ± 0.09
	LR	0.23 ± 0.003	0.05 ± 0.001	0.48 ± 0.04
	MLP	0.28 ± 0.004	0.09 ± 0.004	0.53 ± 0.04
	XG	0.36 ± 0.03	0.08 ± 0.003	0.50 ± 0.07
Literal2Feature	RF	0.37 ± 0.007	0.10 ± 0.01	0.57 ± 0.07
	LR	0.41 ± 0.002	0.12 ± 0.005	**0.62 ± 0.08**
	MLP	**0.46 ± 0.006**	**0.21 ± 0.005**	0.48 ± 0.07
	XG	0.38 ± 0.18	0.19 ± 0.07	0.53 ± 0.04

points in the same class, and (ii) the mean distance between a sample and all other points in the next nearest cluster. The Silhouette Coefficient s for a single sample is then given as:

$$s = \frac{b - a}{max(a, b)} \tag{4.1}$$

The Silhouette Coefficient for a set of samples is given as the mean of the Silhouette Coefficient for each sample and a higher Silhouette Coefficient score relates to a model with better defined clusters. As a clustering algorithm, K-Means has been selected and by applying elbow-method, the optimal number of clusters has been set to 4. Table 4.4 shows the result of the clustering.

Table 4.4: Literal2Feature: Silhouette Coefficient

	Engie	**Carcinogenesis**
RDF2Vec	0.004	0.263
TransE	0.113	**0.669**
SimplE	0.008	0.008
DistMult	0.006	0.009
Literal2Feature	**0.133**	0.247

Table 4.5: Literal2Feature: Synthetic dataset description

Dataset	**#Seeds**	**Size**	**#Triples**
DS 1	1	6.5 MB	127 K
DS 2	300	2.2 GB	38 M
DS 3	600	4.5 GB	76 M
DS 4	1200	9.1 GB	153 M
DS 5	2400	13 GB	306 M
DS 6	6000	47 GB	765 M

As the result depicts, Literal2Feature achieved better clustering for the Engie dataset and also obtained comparable results to RDF2Vec for the Carcinogenesis dataset where TransE achieved the best clustering score.

4.4.2 Experiment B: Scalability

In this experiment, we evaluate the scalability of our approach by different data sizes and varying cluster processing setups. To be able to have different sizes of datasets, we implemented an RDF data simulator which generates synthetic RDF graphs based on the given depth, branching factor, and number of the seeds. Table 4.5 listed the generated datasets and their characteristics. The branching

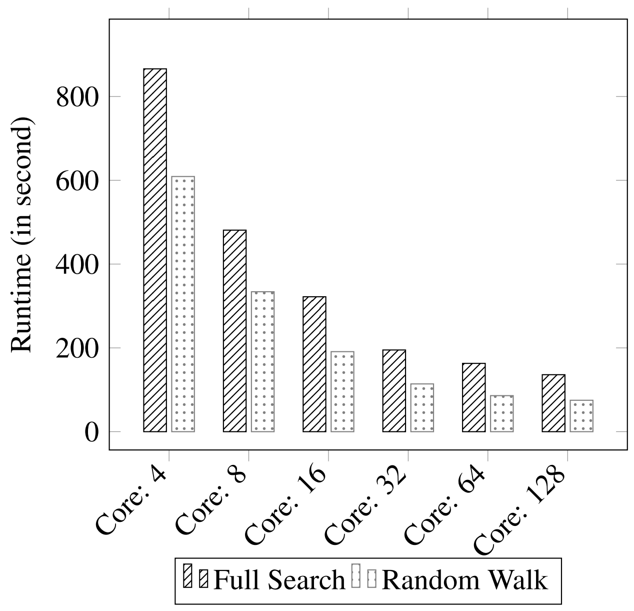


Figure 4.9: Literal2Feature evaluation: processing power scalability on DS 4 Dataset

factor is set to 50, depth to 3, and all the literals lie at the last depth to form a complete tree. Worth mentioning that the German DBpedia size is 48GB with 336 million triples, however, as the branching factor of nodes is not fixed and equal in real datasets, we decided to do the experiments on synthetic data which has the highest CPU and memory consumption.

Scalability over the number of cores

To adjust the processing power, the number of available cores is regulated. In this experiment, *DS 4* is selected as a pilot dataset, and the number of cores increases starting from $2^2 = 4$ up to $2^7 = 128$. The experiments were carried out on a small cluster of 4 nodes (1 master, 3 workers): AMD Opteron(tm) CPU Processor 6376 @ 2300MHz (64 Cores), 256 GB RAM. Moreover, the machines were connected via a Gigabit network. All experiments have been executed three times and the average value is reported in the results. Figure 4.9 shows the scalability over cluster setups. It is obviously clear that increasing the computational power horizontally decreases the execution time. The runtime does not include the time for data ingestion from the Hadoop file system and SPARQL query execution because we relied on the SANSA SPARQL engine for SPARQL execution and the execution time depends on the machine (i.e. single node, cluster) on which the query is being run.

In the beginning, by doubling the number of cores, the execution time dramatically decreases almost by a factor of 2. However, by adding more cores, the execution time only slightly decreases. This behavior can be seen due to the overhead of shuffling data between nodes and network latency. The maximum speed-up is 6.3x.

Scalability over dataset size

To analyze the scalability over different datasets, we fix the computational power with 64 cores and run the experiments for all the datasets introduced in Table 4.5. By comparing the runtime as shown

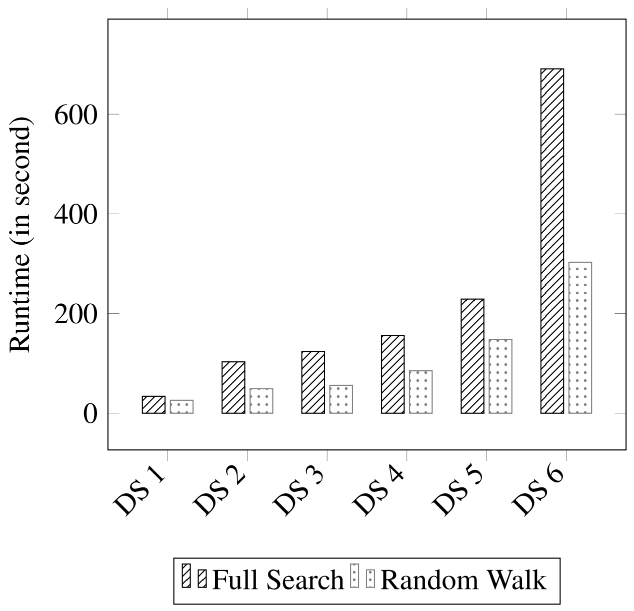


Figure 4.10: Literal2Feature evaluation: sizeup performance evaluation over 64 Cores

in Figure 4.10, we note that the execution time does not increase exponentially. So doubling the size of the dataset does not increase the execution time with the factor of 2. This behavior is due to the available resources (memory) and partition size. On the other hand, as expected, it can be noted that the random walk consumes much less time as compared to the full search, which requires comparatively more resources.

4.5 Summary

In this chapter, we introduced Literal2Feature, a generic distributed framework for transforming RDF data to a feature matrix that can be used in most machine learning algorithms. By providing full control over different hyper-parameters, users will have a substantial level of flexibility in using the framework. Our experiments also showed that the framework can be used to analyze RDF data with existing statistical machine learning pipelines. Moreover, our experiments show that the Literal2Feature can be successfully scaled over a cluster of nodes. Literal2Feature becomes one central feature-extracting tool for DistAD in Chapter 5 and builds one feature-extracting opportunity within ExPAD in Chapter 6.

CHAPTER 5

Scalable and Distributed Anomaly Detection on KGs

Acknowledgement This chapter is based on our scientific publication: **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, “DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs”, IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 243-250, `https://doi.org/10.1109/ICSC52841.2022.00047`.

5.1 Motivation

Anomaly Detection (AD) is a branch of data mining dedicated to the discovery of uncommon events in datasets and has several high-impact applications in sectors such as security, finance, health care, law enforcement, and much more [16]. The goal of anomaly detection is finding an answer to the critical question, “What is intriguing about a dataset?” It refers to the task of identifying data point(s) and patterns that do not conform to the data’s previously specified behavior. Although numerous techniques for detecting outliers and anomalies in unstructured collections of multiple dimension points have been developed in recent years, with the current interest in large-scale heterogeneous data in knowledge graphs, most of the traditional algorithms are no longer directly applicable.

As already explained, KGs are being generated in a variety of ways. However, due to the variety of approaches and freedom in inserting the input data, KGs are prone to various kinds of errors because the entered data is neither restricted nor cross-validated. These errors can happen at `subject`, `predicate`, or `object` part in the RDF format. For example, there can be extraction errors like parsing errors, e.g. some events in Wikipedia have no starting date, and the value for those events is empty and is written like “- 2001”, so extraction tools may interpret this value as a negative year, or there can be errors in the `predicate` part, such as a person has n birth places[1] which $n \gg 1$ (normally, a person has only one birthplace). Moreover, the errors can happen in a multi-feature manner, for example, a person’s age can be 5 and it is reasonable, however, the age of a person who is a president of a country can not be 5. So in the multi-feature mode, a combination of different values could yield an anomaly. It is worth mentioning that anomalies are not necessarily wrong values but values that do not conform with the foreseen data behavior. For example, IoT sensors may generate

[1] https://dbpedia.org/page/Alireza_Afzal

very high/low values (e.g. temperature), these values are not necessarily wrong but adequate to trigger subsequent actions such as alarms.

Although various strategies for detecting outliers and anomalies have been developed over the years, most standard analytic approaches are no longer directly applicable to KGs due to their graph-like, multi-modal nature, and large size. To tackle the aforementioned issues, in this chapter, we propose DistAD, a generic, distributed, and scalable software framework that can automatically detect anomalies in the KGs by extracting semantic features from RDF data, clustering entities, and applying an anomaly detection algorithm on the level of *numeric objects*, *predicates*, and *multi-feature* scenarios. DistAD offers flexibility over different parts of the workflow and lets the end-users select different approaches and granularity based on their use cases. In addition, DistAD is integrated into the SANSA stack [19] and interacts with the different SANSA computational layers.

In this chapter, we address the following research question:

RQ2: How can we apply anomaly detection on knowledge graphs in a scalable and distributed manner?

To summarize, the main contributions of this chapter are as follows:

- Introducing a distributed generic framework that can automatically detect anomalies in a large RDF graph
- Integrating the approach into the SANSA stack
- Covering the code by unit tests, documenting it in Scala docs, and providing a tutorial
- Making DistAD and the framework open-source and publicly available on GitHub[2]
- Evaluation of the results over multiple datasets and empirical evaluation of scalability

The rest of the chapter is structured as follows: DistAD workflow and implementation are detailed in Section 5.2. Section 5.3 covers the evaluation of the DistAD and demonstrates the scalability. Finally, we summarize our work in Section 5.4.

5.2 DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs

In this section, we present DistAD, a generic framework for anomaly detection in KGs. The framework performs anomaly detection by extracting semantic features from entities for calculating similarity, applying clustering on the entities, and running multiple anomaly detection algorithms to detect the outliers on the different levels and granularity. The output of DistAD is a list of anomalous RDF triples. In the development process of the DistAD framework, we considered the factors of availability of the framework, its impact on the community, and usability.

(i) Availability: All the components of the framework are fully integrated into the SANSA ecosystem within the machine learning layer. The framework is fully available for the community as an open-source GitHub repository.

[2] https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.3_DistAD

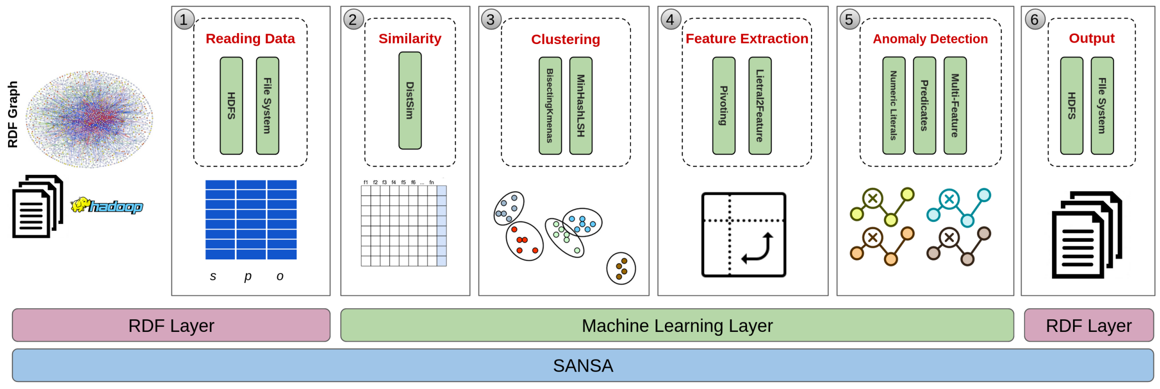


Figure 5.1: DistAD: system architecture abstract overview

(ii) Impact: DistAD offers practitioners from the semantic web domain to detect the outliers in their dataset and improve the quality of the existing dataset. Moreover, thanks to its power in handling huge RDF data, it can be used in IoT domains to detect anomalous events in the RDFized sensor data in many areas such as energy domain[3].

(iii) Usability: The framework is not only documented on the code level but we also provide a tutorial on its use. We also have numerous samples available as an example class to significantly assist in a try-out barrier.

DistAD offers options to select different algorithms and hyperparameters to prepare a customized pipeline for anomaly detection. Table 5.1 lists the possibilities and Figure 5.1 depicts the high-level system architecture overview. In the following, each step of the framework is explained in detail based on Figure 5.1.

5.2.1 Components

Step 1: Reading Data

The first step of DistAD is reading the RDF data and loading it into memory as a dataframe. The RDF data can reside in normal file systems or HDFS (Hadoop File System). The framework supports *N-Triple* and *Turtle* file formats.

Step 2: Similarity Calculation

Most clustering algorithms require a mechanism to calculate the similarity between different data points. In our framework, we used DistSim [148]. The semantic similarity estimations of DistSim operate based on feature sets. These feature sets are derived from the RDF data by the Feature-Extractor Module, which is implemented as a Transformer. DistSim provides multiple modes for the feature extraction module. However, in this framework, we only use the *predicates* (OR mode in [148]) as main features. In short, in this mode, two entities will be similar if they share many common predicates. This helps the clustering algorithm to group similar entities together. Moreover, we implemented two variations, i.e. *Full* and *Partial* similarity. In *Full* similarity, we consider all the existing predicates of

[3] https://platoon-project.eu/

Table 5.1: DistAD configurable components

Feature	**Options**	**Comments**
Similarity Calculation	`Full Similarity`	Consider all the predicates to generate features
	`Partial Similarity`	Consider only numerical predicates to generate features
Clustering Algorithm	`BisectingKmeans`[147]	Hierarchical version of K-Means clustering algorithm
	`MinHashLSH`	Cohorting based on Local Sensitivity Hash
Feature Extraction	`Pivoting/Grouping`	Basic operation for extracting feature from RDF data
	`Literal2Feature`[23]	Sophisticated method for extracting features from RDF data
Anomaly Detection Type	`Numeric Literals`	Detects anomalies only in the numeric values
	`Predicates`	Detects anomalies on the predicate level
	`Multi-feature`	Detects anomalies on a set of features
Anomaly Detection Algorithms	`Interquartile Range`[60]	Used for single-value features (numeric literals and predicates)
	`Median Absolute Deviation`[61]	Used for single-value features (numeric literals and predicates)
	`Z-score`	Used for single-value features (numeric literals and predicates)
	`IsolationForest`[64]	Used for multi-feature scenarios
Cluster Detection	`Silhouette Method`	For detecting the best optimal number of clusters

an entity to calculate the similarity between entities. In *Partial* mode, we only consider predicates that have numeric literals as objects. Although the accuracy of the *Full* mode is higher due to considering all the available predicates, the *Partial* mode benefits from faster operation due to less number of predicates.

Step 3: Clustering Algorithms

Clustering is a key point in most anomaly detection techniques on KGs. The reason why clustering is needed in anomaly detection in KGs is that the traditional methods may gather all values of a certain predicate, such as *dbp:weight*, and attempt to detect anomalies for this feature. However, in KGs, comparing a feature from different entity types (e.g., the weight of persons against the weight of vehicles) is logically incorrect. Therefore here we mention two clustering algorithms that have been integrated into the DistAD framework.

BisectingKmeans [147]

The bisecting K-Means clustering algorithm is a variation of the standard K-Means algorithm. The method begins with a single cluster containing all of the points. Iteratively, it discovers divisible clusters on the bottom level and bisects each one using k-means until there are k total leaf clusters or none are divisible. To promote parallelism, the bisecting steps of clusters on the same level are grouped. If bisecting all divisible clusters on the bottom level results in more than k leaf clusters, the bigger clusters take precedence.

Besides the algorithm, we integrated the Silhouette method with squared Euclidean distance, which is a heuristic approach to determine the optimal number of clusters in a data set. The Silhouette Coefficient value presents a measure of how close each point in one cluster is to points in the neighboring clusters. This measure has a range of $[-1, 1]$, and a higher Silhouette Coefficient score relates to a model with better-defined clusters. Our implementation can automatically select the optional k for the clustering.

MinHashLSH

Locality Sensitive Hashing (LSH) [111] is an important class of hashing techniques that are commonly used in clustering, approximate nearest neighbor search, and outlier detection with large datasets. LSH hashes data points into buckets using a family of functions (“LSH families”), so that data points that are near to each other are in the same bucket with a high likelihood. MinHash[4] is an LSH family method for Jaccard distance where input features are sets of natural numbers. The output of MinHashLSH is a pairwise Jaccard similarity between data points.

Step 4: Feature Extraction

Before being able to apply any anomaly detection algorithm to the RDF data, the KGs should be vectorized (Prepositionalized). This step moves each feature (object, predicate,...) to a separate column in Spark dataframes for further subsequent analysis. To this end, we integrated the following approaches.

[4] https://spark.apache.org/docs/latest/ml-features#minhash-for-jaccard-distance

Pivoting/Grouping

Pivoting is a reshaping mechanism that Spark provides over dataframes. Pivoting reshapes data (produce a "pivot" table) based on column values. The following example depicts how pivoting works on sample RDF data if one wants to pivot the Listing 5.1 based on "Predicate" and aggregate over "Object". The result is shown in Listing 5.2.

```
+-----------------+--------------+-----------+
|Subject          |Predicate     |Object     |
+-----------------+--------------+-----------+
|dbr:Barack_Obama |dbo:birthPlace|dbr:Hawaii |
|dbr:Barack_Obama |dbo:birthDate |1961-08-04 |
|dbr:Angela_Merkel|dbo:birthPlace|dbr:Hamburg|
|dbr:Angela_Merkel|dbo:birthDate |1954-07-17 |
+-----------------+--------------+-----------+
```

Listing 5.1: Original dataframe before pivoting

```
+-----------------+-------------+--------------+
|Subject          |dbo:birthDate|dbo:birthPlace|
+-----------------+-------------+--------------+
|dbr:Barack_Obama |1961-08-04   |dbr:Hawaii    |
|dbr:Angela_Merkel|1954-07-17   |dbr:Hamburg   |
+-----------------+-------------+--------------+
```

Listing 5.2: Dataframe after pivoting based on "Predicate" and aggregating over "Object"

Literal2Feature

Literal2Feature [23] is a generic, distributed, and a scalable software framework that can automatically transform a given RDF dataset to a standard feature matrix by deep traversing the RDF graph and extracting literals to a given depth. The result of Literal2Feature is a SPARQL query that extracts the features. This option helps the user to extract features that are not in the direct vicinity of an entity for the outlier detection purpose. A possible small sample feature extracting SPARQL query created by the Literal2Feature model is shown in Listing 5.3. This query is executed by the SANSA built-in SPARQL engine and the result is the vectorized RDF dataframe. Literal2Feature is fully introduced in Chapter 4.

Step 5: Anomaly Detection Type

In DistAD, we have provided 3 types of anomaly detection methods in the framework i.e. (i) Numeric Literals, (ii) Predicates, and (iii) Multi-Feature.

Literal Values

Due to the liberty of KG curation methods, KGs are prone to various kinds of errors. As explained earlier, the errors may happen in `Subject`, `Predicate`, or `Object` parts. The focus here is on numeric literals. For example, there can be extraction errors like parsing errors, e.g. "3-4" can be interpreted as "3", and "-4".

Predicates

This type of error happens when an entity has more/less than a usual number of the same predicate. For example, a person normally may have none to a few children. However, if he/she has, for example, 200 children, then this type of (potential) error should be detected to increase the quality of the KGs. In this case, the dataframe containing RDF data is transformed into a new dataframe by grouping based on subjects and predicates and counting based on predicates.

```
SELECT
?movie
?movie__down_title
?movie__down_runtime
WHERE {
 ?movie a <http://data.linkedmdb.org/movie/film> .
 OPTIONAL { ?movie <http://purl.org/dc/terms/title> ?movie__down_title .}
 OPTIONAL { ?movie <http://data.linkedmdb.org/movie/runtime> ?
  movie__down_runtime .}
}
```

Listing 5.3: Sample SPARQL query of Literal2Feature

Multi-Feature

Multi-feature anomaly detection helps users to detect contextual anomalies. Typically, there exists some hidden correlation between multiple features, for example, there is a positive correlation between the height and the age of a person. By considering these features separately, the algorithm will not detect a person with a 1.8-meter height as abnormal. However, having contextual information such as the person's age (e.g., 2 years old) can make this combination anomalous.

Anomaly Detection Algorithms

To cover the mentioned anomaly detection methods, we have integrated multiple prominent anomaly detection algorithms. For detecting anomalies in the numerical literals and predicates, IQR, MAD, and Z-Score are implemented, and for the multi-feature scenario, Isolation Forest [64] is integrated. The technical definition of these algorithms has been presented in Section 2.4.1.

Step 6: Output

The last step of the framework is saving the output to a file. The output is the list of anomalous RDF triples. The triples can be saved as a normal file on a file system or on HDFS.

5.2.2 Implementation

As the programming language of SANSA is Scala[5], we have selected this language and its APIs in Apache Spark to provide the distributed implementation of DistAD. Moreover, we benefit from SANSA IO, ML, and Query layers. Technically, as it can be seen in Figure 5.1, DistAD can be divided into the following steps 1) Reading RDF data as a data frame, 2) vectorizing the RDF data,

[5] https://www.scala-lang.org/

Table 5.2: DistAD: Dataset statistics

Dataset	**Accident**	**DBpedia1**	**DBpedia2**
Format	Turtle	N-Triple	N-Triple
#Triples	5,961,107	1,000,000	10,000,000
#Distinct Predicates	63	15,520	29,375
#Distinct Numeric Predicates	5	7,067	14,372
File Size	461 MB	137 MB	1.4 GB

3) Clustering the data, 4) Extracting features for anomaly detection, 5) Running anomaly detection algorithm, and 6) Save the anomalies in a file.

5.3 Experiments

In this section, we present two sets of experiments to analyze different aspects of DistAD. In the first experiment, the correctness of the extracted anomalies will be analyzed over different datasets, and in the second experiment, the scalability of the proposed framework will be investigated.

5.3.1 Experiment A: Assessment of the detected Anomalies

In this section, we analyze the detected anomalies. To this end, three datasets have been exploited. Engie-accident dataset, 1 million, and 10 million triple samples of DBpedia[6]. Engie SA[7] is a French multinational electric utility company that operates in the fields of energy transition, electricity generation and distribution, natural gas, nuclear, renewable energy, and petroleum. The accident dataset contains data about accidents that occurred in France in 2018[8]. DBpedia dataset is a sample of the infobox part which contains most of the literal predicates in DBpedia. An overview of the datasets is given in Table 5.2.

Without loss of generality, we define the following setups to show the ability of the framework:

Anomaly Detection on Numerical Literals

For this case, we use the DBpedia1 dataset. We selected the BisectingKmeans algorithm for clustering, IQR for anomaly detection, *Full* mode for the semantic similarity feature extractor, and pivoting for vectorization. We also performed the silhouette method and set the number of clusters to 4. As a result, DistAD could detect 19,778 triples that contain anomalous values from 1,232 distinct properties. Some prominent anomalous properties found by DistAD are *dbp:year*, *dbo:postalCode*, *dbo:year2start*, etc., and as it can be seen most of the outliers are found in the date/time-related predicates. Manual inspection of detected outliers revealed that the DBpedia extraction tool can

[6] https://databus.dbpedia.org/dbpedia/collections/latest-core

[7] https://www.engie.com

[8] can not be publicly published due to intellectual property concerns

Table 5.3: DistAD: Example of real outliers in DBpedia

Entity	Predicate	Wikipedia Value	DBpedia Value	Reason
Ian_Turbott	dbp:year	1989-2000	19892000^^xsd:integer	can not handle hyphen
Bidhan_Saran	dbo:postalCode	700004, 700006, 700007	700004700006700007^^xsd:integer	can not handle commas
Steve_Walters	dbp:year2start	198?–85	198^^xsd:integer	wrong value in Wikipedia

not always extract correctly the information related to date/time and that leads to the wrong values. Table 5.3 shows a few detected outliers and their corresponding values from Wikipedia.

Anomaly Detection on Predicates

For this case, we used the same configuration as the previous experiment but used the larger DBpedia2 dataset. By running DistAD, we could detect some interesting anomalies in the predicates. For example, *Nasir_al-Din_al-Tusi* has four *dbo:birthDate* in the DBpedia dump, however, by manually checking it we realized that three out of four are locations, not the dates. This error is propagated to DBpedia because of putting locations under the *Born* property in Wikipedia infobox. Moreover, for example, *Alia_Toukan* has two *dbp:father* which by checking Wikipedia, we realized that there is one name as her father but with two different links which caused this issue.

Multi-Feature Anomaly Detection

In this case, we used the Engie dataset. As this dataset contains only accident entities, we set the number of clusters to 1 and skipped the clustering part. Regarding anomaly detection, we used Isolation Forest and used Literal2Feature for feature extraction.

Figure 5.2 shows the detected anomalies. As it can be seen, there are a few accidents that happened outside France and the algorithm could correctly detect them as anomalies by considering their geo-coordinates as a feature set and isolating them from the normal data. These types of anomalies will not be detected if one considers only latitude or longitude separately. However, considering them together makes it possible to detect contextual outliers. The dataset contains data of 57,783 accidents, out of those 1,962 are outside France, and the framework could correctly detect all outliers.

5.3.2 Experiment B: Scalability

In this experiment, we evaluate the scalability of DistAD by using different data sizes and varying cluster computing setups. To be able to have different sizes of datasets, we have sampled the DBpedia dataset. Table 5.4 lists the datasets and their characteristic. As the scalability of Literal2Feature and MinHashLSH has been investigated in [23] and [149] respectively, for these experiments, we use BisectingKmeans for clustering, Z-Score for anomaly detection, *Full* mode for the semantic similarity feature extractor, and pivoting/grouping for feature extraction. Moreover, we run two algorithms for detecting anomalies over numeric literals and predicates.

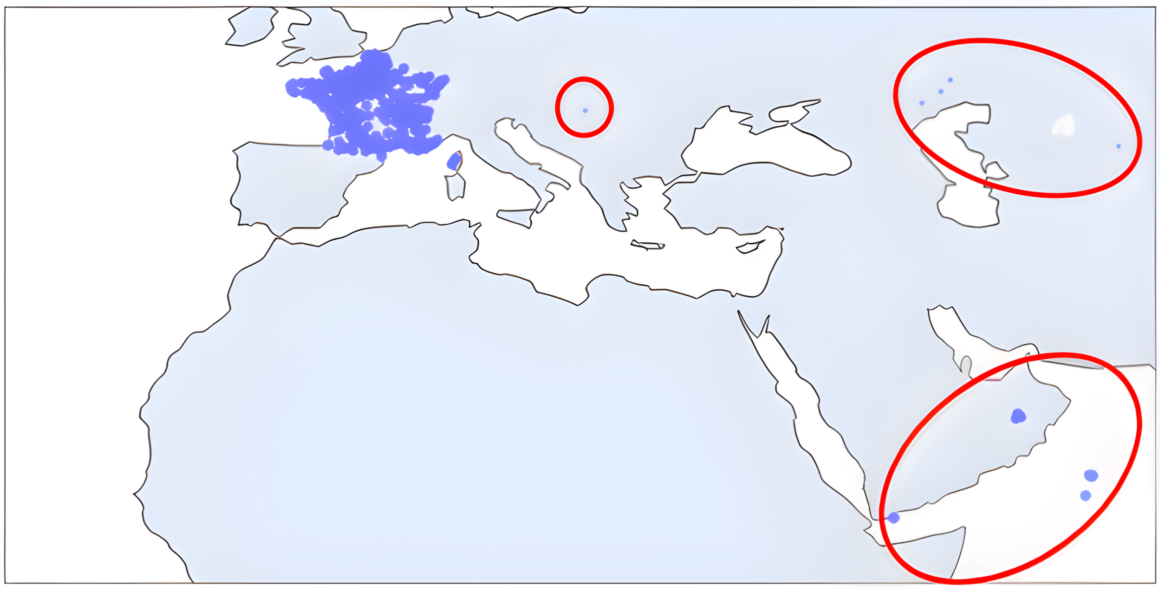


Figure 5.2: DistAD: accidents which are detected as anomalies

Table 5.4: DistAD: Dataset description

Dataset	#Triples	Size
DBpedia-S	1 M	136 MB
DBpedia-M	10 M	1.4 GB
DBpedia-L	50 M	6.8 GB
DBpedia	97.5 M	13.3 GB
DBpedia × 2	195 M	26.6 GB
DBpedia × 4	390 M	53.2 GB

Scalability over the number of cores

To adjust the distributed processing power, the number of available cores was regulated. In this experiment, we selected DBpedia (13.3 GB) as a pilot dataset and the number of cores was increased starting from $2^3 = 8$ up to $2^7 = 128$. The experiments were carried out on a small cluster of 4 nodes (1 master, 3 workers): AMD Opteron(TM) CPU Processor 6376 @ 2300MHz (64 Cores), 256 GB RAM. Moreover, the machines were connected via a Gigabit network. All experiments are executed three times and the average value is reported in the results. Figure 5.3 shows the scalability over different computing cluster setups. It is clear that increasing the computational power horizontally decreases the execution time. Initially, doubling the number of cores reduces the execution time by nearly a factor of two. However, increasing the number of cores only minimally reduces execution time. This phenomenon is caused by the overhead of moving data between nodes as well as network delay.

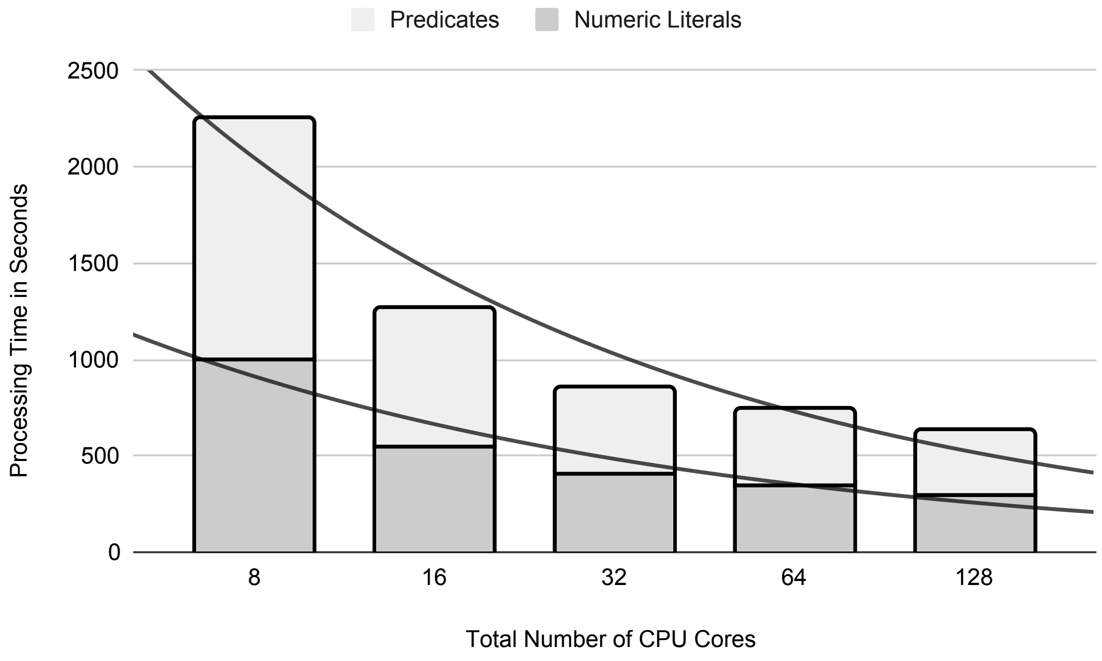


Figure 5.3: DistAD evaluation: processing power vs processing time

Scalability over dataset size

To analyze the scalability over different datasets, we fix the computational power to 64 cores and run the experiments for all datasets introduced in Table 5.4. By comparing the runtime as shown in Figure 5.4, we note that the execution time does not increase exponentially. Hence, increasing the size of the dataset with a factor of 10 does not necessarily increase the execution time by a factor of 10. This behavior is due to the distribution among available resources e.g. (memory) and partition size. It can be seen that by increasing the dataset size from 1.4 GB to 13.3 GB (~ 9.5 times bigger), the execution time almost only doubles.

5.4 Summary

In this chapter, we introduced DistAD, a generic, distributed, and scalable framework for anomaly detection in KGs. DistAD is open-source, available on GitHub, and integrated into SANSA Stack. By providing full control over different algorithms, methods, and (hyper-)parameters, DistAD enables users to have a substantial level of flexibility in using the framework. Our experiments show that the framework can correctly detect different types of anomalies in KGs and it can help to improve the data quality in KGs. Moreover, our experiments show that DistAD can be successfully scaled over a cluster of nodes for a large size of data.

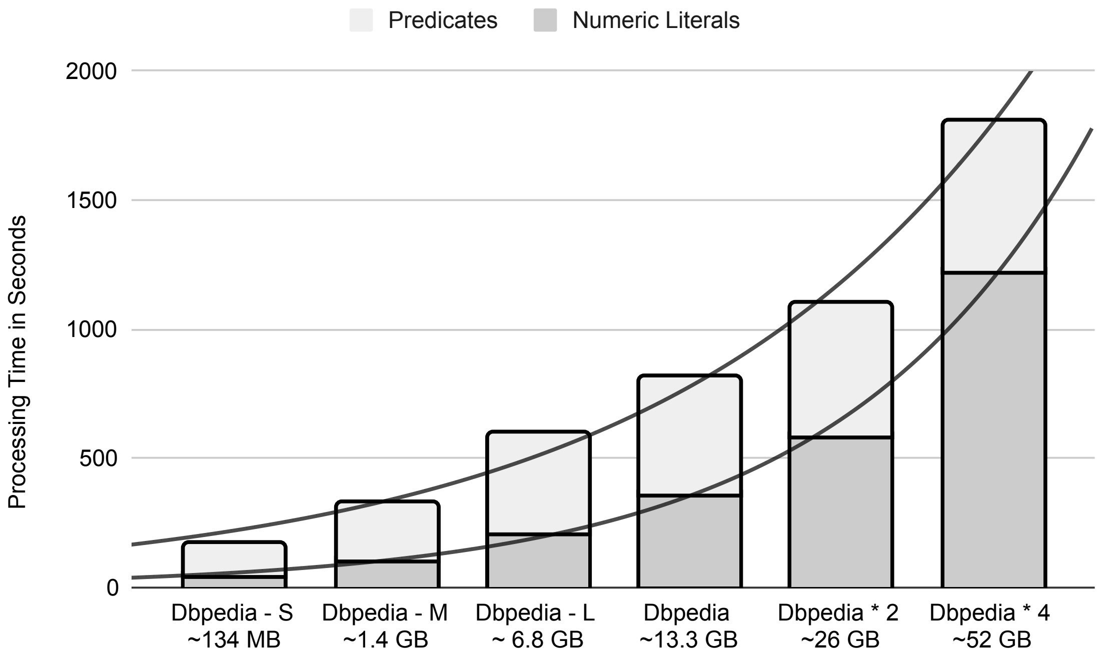


Figure 5.4: DistAD evaluation: size-up performance evaluation over 64 Cores

CHAPTER 6

Explainable Anomaly Detection on KGs

Acknowledgement This chapter is based on our scientific publication: **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs", IEEE 17th International Conference on Semantic Computing (ICSC), 2023, pp. 204-211, `https://doi.org/10.1109/ICSC56153.2023.00040`.

6.1 Motivation

Although AD is a well-studied field in the area of machine learning and statistics, there is a significant gap of AD in the area of Semantic Web and large-scale heterogeneous Knowledge Graphs (KGs). Especially when it comes to interpretability, there is no research work in the area of explainable anomaly detection in KGs.

As already explained, because the entered data are generally neither constrained nor cross-validated, KGs are prone to various types of errors because of the range of approaches and freedom in inserting the input data. These errors can happen at `Subject`, `Predicate`, or `Object` part in the RDF format. For example, there can be errors in the `predicate` part, such as a person has n death dates[1] which $n > 1$ (a person has only one death date). Or there can be extraction errors like parsing errors, e.g. in some cases the extraction tool can not interpret "-" and converts "1989-2000" in Wikipedia[2] to "19892000" in DBpedia[3].

Although numerous techniques for detecting outliers and anomalies in unstructured collections of multiple dimension points have been developed in recent years, with the current interest in large-scale heterogeneous data in knowledge graphs, most of the traditional algorithms are no longer directly applicable to KGs due to scalability and RDF complex data structure. Furthermore, uncovering why a reported anomaly has occurred (explanation discovery), forms a crucial capability for any anomaly detection system.

In this chapter, we propose ExPAD, a generic, distributed, and scalable software framework that can automatically detect numeric anomalies in KGs and produce human-readable explanations for

[1] https://dbpedia.org/page/Nasir_al-Din_al-Tusi
[2] https://en.wikipedia.org/wiki/Ian_Turbott
[3] https://dbpedia.org/page/Ian_Turbott

why a given value of a variable in an observation can be considered as outlier. ExPAD is inspired by OutlierTree [137] and works by evaluating and following distributed supervised decision tree splits on variables. This helps to detect and explain anomalous cases which can not be detected without considering other features. For example, it may be reasonable to assume that a female person's age is 4. However, if we know that this person is pregnant, then the age of 4 would not be possible. Under this logic, it is possible to produce human-readable explanations for why a given value of a variable in an observation can be considered as outlier, by considering the decision tree branch conditions.

Furthermore, given our main focus on distributed computing and the ability to perform on large KGs, ExPAD is integrated into the SANSA stack [19].

In this chapter, we address the following research question:

RQ3: Can explainable anomaly detection be performed efficiently and effectively on knowledge graphs?

To summarize, the main contributions of this chapter are as follows:

- A scalable distributed anomaly detection framework that can automatically detect numeric anomalies in a large RDF graph and provide clear explanations for why certain values are identified as anomalies.
- Integration of ExPAD into the holistic SANSA stack and making it open-source and publicly available on GitHub[4]
- Covering the code by unit and integration tests, documenting it, and providing a tutorial within Databricks
- Evaluation of the results over multiple datasets and empirical evaluation of scalability

The remainder of the chapter is organized as follows: Section 6.2 describes the architecture and implementation of ExPAD. Section 6.3 is devoted to the experimental setup, the datasets, and scalability evaluation. Finally, Section 6.4 summarizes the chapter.

6.2 ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs

ExPAD is a generic distributed framework for explainable anomaly detection in KGs. The main goal of the framework is to detect numeric anomalies in KGs and to provide human-readable explanations for why a given value can be considered as an outlier. ExPAD vectorizes RDF data (extracting literals from RDF which is explained in Chapter 4), then to detect anomalies on a given numerical variable v, it fits a Distributed Decision Tree[5], choosing v as the target and the rest of attributes as the features. Based on the depth of the tree, v is divided into two or more partitions. By applying any anomaly detection method (eg. IQR) over the partitioned data and finding anomalies, it is possible to produce explanations for why a given value of the target variable is identified as an outlier, by considering the

[4] https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.5_ExPAD

[5] https://spark.apache.org/docs/latest/mllib-decision-tree.html

decision tree branches and their corresponding variables. An example is sketched here to shed light on the working of ExPAD. Consider an RDF dataset containing information about the class *Person* with sample predicates *age (numerical)*, *job (URI)*, and *gender (boolean)* with 100 triples. After vectorizing and indexing this data, ExPAD considers one predicate (feature) as a target variable and the rest of the predicates as features for training a decision tree. Figure 6.1 depicts a fitted decision tree model with target feature *age*. As can be seen each tree node (internal or leaf) partitions data. For example, the root node partitions data based on *job* or the second leaf node from the right partitions data based on *job* and *gender*. Based on the second leaf node from the right, the estimated age for a person with job type 1.0 (this is a mapping index) and gender 0.0 is 16.30, however, based on this partitioning there is a person with the age of >= 100 which is an anomaly in this scenario. Now this anomaly can be explained as "*the age value* >= 100 *is suspicious given job* = 1.0 *and gender* = 0.0" or after mapping indices to the original values "*the age value* >= 100 *is suspicious given job=student and gender=female*". By this logic, the fitted decision tree can be parsed and for each node, an SQL query can be extracted (e.g. `SELECT * FROM data WHERE job=1.0 AND gender=0.0`). We consider these queries as rules. Finally applying these extracted rules on the dataset and applying anomaly detection techniques on the target values can yield explanations for the possible detected outliers.

In the development process of the ExPAD framework, we considered the factors of availability of the framework, its impact on the community, and usability.

(i) Availability: To make the framework available for the community, all the components of it have been integrated into the SANSA ecosystem. The framework is fully available for the community as an open-source GitHub repository [150].

(ii) Impact: ExPAD offers Semantic Web practitioners a tool to not only detect anomalies in their RDF dataset but also provide an explanation and improve the quality of the existing dataset. Besides this, as it can handle huge RDF data, it can be a suitable option for enterprise companies in the field of energy and IoT to detect anomalous events in the RDFized sensor data[6].

(iii) Usability: Although the framework is fully documented we also provided a tutorial and a sample in Databricks to assist end-users and significantly decrease the try-out barrier.

Figure 6.2 depicts the high-level system architecture overview of ExPAD. Moreover, Table 6.1 lists configurable parts of the framework. In the following, each step of the framework is explained in detail based on Figure 6.2.

6.2.1 Components

Step 1: Reading Data

The first step of the pipeline is reading and loading the RDF data. The output of this step will be a Spark dataframe. ExPAD supports *N-Triple* and *Turtle* file formats and the RDF data can reside in normal file systems or Hadoop File System (HDFS).

Step 2: Feature Extractor

The output of the first step is a dataframe with 3 columns that store ⟨`subject,predicate,object`⟩, however, to be able to apply any anomaly detection algorithm on the `object` part, the dataframe

[6] https://platoon-project.eu

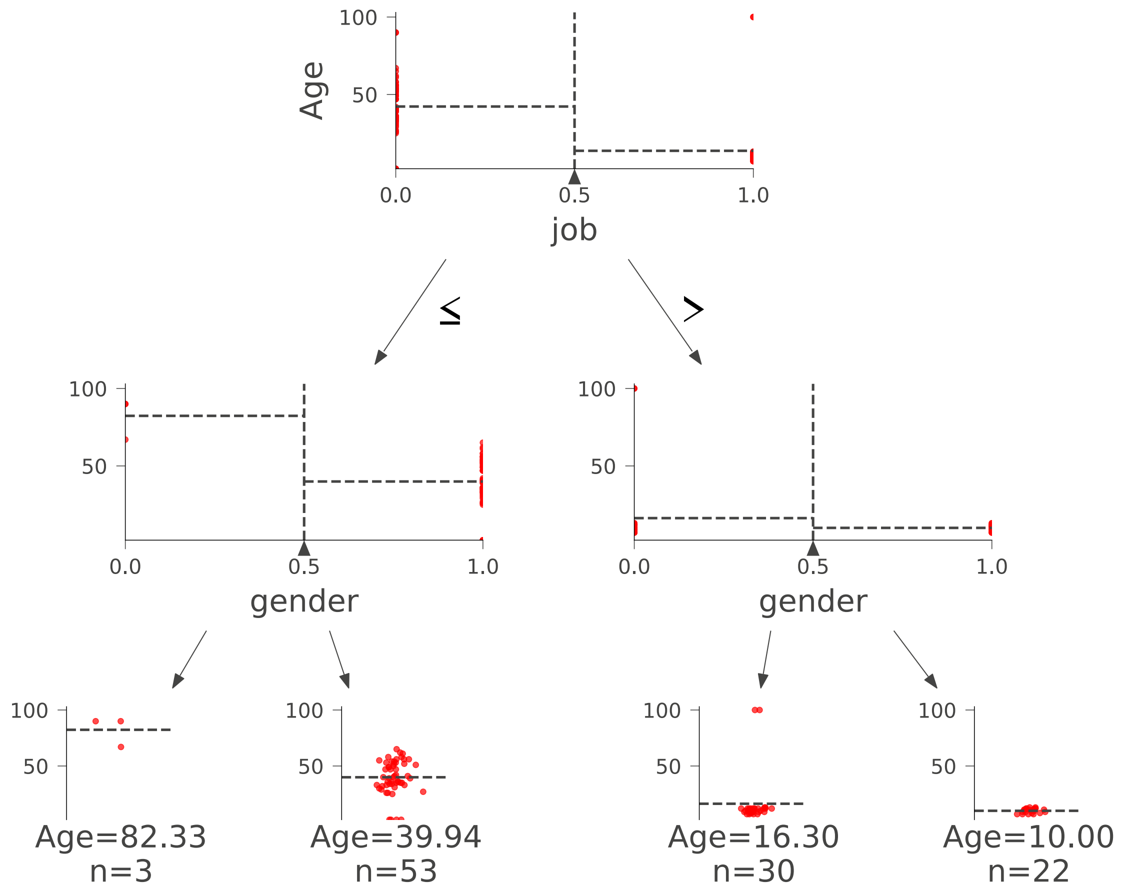


Figure 6.1: ExPAD: trained decision tree with the target variable *age*

should be vectorized (Prepositionalized). This step moves each `predicate` to a separate column and reshapes the dataframe accordingly. In this step, we borrowed two integrated approaches from DistAD (Chapter 5) to apply vectorization.

Pivoting/Grouping

A pivot is an aggregation where one (or more in the general case) of the grouping columns has its distinct values transposed into individual columns. Spark provides a pivoting mechanism over dataframes. The detailed explanation can be found in Section 5.2.1

Literal2Feature

Literal2Feature [23] transforms a given RDF dataset to a standard feature matrix by deep traversing the RDF graph and extracting literals to a given depth. It generates a SPARQL query which is executed by the SANSA built-in SPARQL engine. The result of the SPARQL query execution is the vectorized RDF dataframe. This option helps the user to extract features that are not in the direct vicinity of an

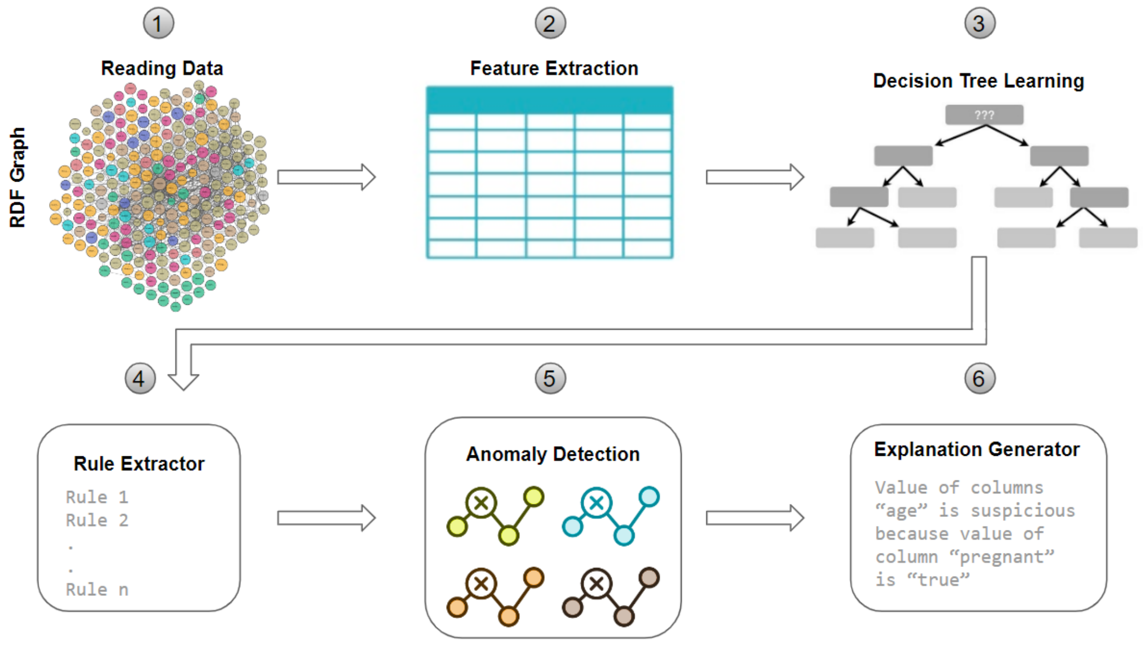


Figure 6.2: ExPAD system architecture high-level overview

entity. The detailed explanation can be found in Chapter 4.

SmartDataFrame

Besides above mentioned approaches, in ExPAD we implemented a new transformation pipeline entitled "*SmartDataFrame*". *SmartDataFrame* not only reads the RDF data into a dataframe but also extracts features based on objects and after detection of data types, casts the literal to the primitive data types. This transformer converts all features corresponding to their feature type into numeric representations. For example, non-categorical strings (e.g. URIs) are transformed into the label indexer mapping or boolean-type will be converted to $\{0, 1\}$. This transformation is necessary to make the corresponding decision tree learning possible. Listing 6.1 and Listing 6.2 depict a native RDF dataset featurized via *SmartDataFrame* transformer:

```
dbr:Person0 dbo:age "13"^^http://www.w3.org/2001/XMLSchema#integer
dbr:Person0 dbo:gender "false"^^http://www.w3.org/2001/XMLSchema#boolean
dbr:Person0 dbo:job dbr:Student
dbr:Person1 dbo:age "8"^^http://www.w3.org/2001/XMLSchema#integer
dbr:Person1 dbo:gender "true"^^http://www.w3.org/2001/XMLSchema#boolean
dbr:Person1 dbo:job dbr:Teacher
```

Listing 6.1: Sample RDF data

Step 3: Decision Tree Learning

The third step of the framework is fitting a distributed decision tree. For this reason, one of the originally numeric columns (features) in the dataframe will be considered as a *target* variable and the

```
+-----------+------------+---------------+-----------+--------------+
|s          |age__integer|gender__boolean|job__url   |job__url_index|
+-----------+------------+---------------+-----------+--------------+
|dbo:Person0|13          |0              |dbr:Student|0             |
|dbo:Person1|8           |1              |dbr:Teacher|1             |
+-----------+------------+---------------+--------------------------+
```

Listing 6.2: Transformed dataframe via SmartDataFrame

rest of the columns as features. We only consider the original numeric features as target variables, because anomaly detection will happen on only these values in the next steps, however, the categorical ones will be involved as a feature in the decision tree training process. The maximum depth of the tree is a hyper-parameter, the deeper the tree goes, the more complex explanation will be produced due to the number of reported variables. This value is configurable by the end user. After the training is done, the decision tree model will be ready to be parsed in the next step.

Step 4: Rule Extractor

This component is responsible for parsing the fitted decision tree model and for each tree node (either internal node or leaf node) extracting an SQL rule. The parsing process uses DFS (Depth First Search) [151] to traverse the whole tree. The process keeps track of each variable in every split and generates an SQL query for each path. Consider the example at Figure 6.1, the generated rules are shown in Listing 6.3.

```
SELECT * from df WHERE job <= 0.5
SELECT * from df WHERE job > 0.5
SELECT * from df WHERE job <= 0.5 AND gender <= 0.5
SELECT * from df WHERE job <= 0.5 AND gender > 0.5
SELECT * from df WHERE job > 0.5 AND gender <= 0.5
SELECT * from df WHERE job > 0.5 AND gender > 0.5
```

Listing 6.3: Generated SQL rules

Worth noting that, instead of SQL, SPARQL rules could also be generated, however, as SQL is a native query language implemented over Spark dataframe, the execution time of SQL over dataframe is much shorter than running SPARQL on the RDF data in SANSA.

Step 5: Anomaly Detection

In ExPAD, we have provided anomaly detection methods for only numerical literals. For detecting anomalies in the numerical literals, we have integrated IQR, MAD, and Z-Score. The technical definition of these algorithms has been presented in Section 2.4.1.

Step 6: Explanation Generator

Now that rules have been generated, one can apply each rule to the dataset to filter the data (fetch cluster). This is possible due to Spark SQL[7] which is a Spark module for running SQL over dataframes.

[7] https://spark.apache.org/sql

The result will be a dataframe as well. The target column will be selected to be checked for anomalies with an anomaly detection algorithm. In case any anomaly is detected, it will be reported alongside the variables and their corresponding values in the SQL rule. For clarification, consider the example in Figure 6.1. By applying `SELECT * from df WHERE job > 0.5 AND gender <= 0.5` and afterward running IQR over the filtered age values, one can see that the value 100 will be detected as an anomaly. This value can be reported as "*the age value* 100 *is suspicious given job* = 1.0 *and gender* = 0.0" or after mapping indices to the original values "*the age value* = 100 *is suspicious given job=student and gender=female*".

The final output of the framework is the list of anomalous RDF triples and corresponding explanations that can be saved as a normal file on a file system or on HDFS.

6.2.2 Implementation

As Scala[8] is the programming language of SANSA, we chose the same and its APIs in Apache Spark[9] to provide the distributed implementation of ExPAD. Moreover, we benefit from the SANSA IO layer for reading/writing RDF data. Technically, ExPAD can be divided into the following steps 1) Reading RDF data as a data frame with *SmartDateFrame*, 2) Generating feature matrix, 3) Training decision tree, 4) Traversing and parsing fitted decision tree and generating rules, 5) Performing anomaly detection, and 6) Generating explanation. Algorithm 3 depicts the framework execution pipeline.

Algorithmus 3 : ExPAD Workflow

```
Data : $G(E,V)$ Knowledge Graph as a RDF serialization
Result : possible anomalies and their explanation
18 $df = read(G)$;
19 $df = featureExtractor(df)$;
20 $df = SmartDataFrame.read(df)$;
21 foreach column $c \in df.cols$ do
22     if $c$ is a numeric literal then
23         $target = c$
24         $features = df.cols \setminus c$
25         $dt = DecisionTree(df, target, features)$
26         $rules = parse(dt)$
27         foreach rule $r \in rules$ do
28             $filteredDF = df.select(r)$
29             $targetData = filteredDF(col = target)$
30             $anomalies = detectAnomalies(targetData)$
31             if anomalies is not empty then
32                 $report(anomalies, r)$
```

[8] https://www.scala-lang.org
[9] https://spark.apache.org

Table 6.1: ExPAD configurable components

Feature	**Options**	**Comments**
Feature Extraction	`Pivoting/Grouping`	Basic operation for extracting feature from RDF data
	`Literal2Feature`[23]	Sophisticated method for extracting features from RDF data
Decision Tree	`MaxDepth`	Maximum depth of the tree (nonnegative)
	`MaxBins`	Maximum number of bins used for discretizing continuous features
Anomaly Detection	`Interquartile Range`[60]	Used for numeric values
	`Median Absolute Deviation`[61]	Used for numeric values
	`Z-score`	Used for numeric values
General	`verbose`	Boolean to generate logs
	`AnomalyListSize`	The minimum number of samples required for anomaly detection method

6.3 Experimental Results

In this section, two sets of experiments will be conducted to analyze two particular aspects of ExPAD, effectiveness and computational performance. In the first experiment, the correctness of the extracted anomalies explanation will be analyzed over different datasets, and in the second experiment, we will investigate the scalability of the framework. To the best of our knowledge, there is no other work to simultaneously detect and explain anomalies on KGs via a distributed computing stack. Therefore, in this section, we introduce multiple scenarios to examine the quality of the ExPAD solely.

6.3.1 Dataset

To the best of our knowledge, there is no RDF dataset benchmark for anomaly detection and especially for explainability. Moreover, in Chapter 5, we figured out that most of the detected numeric outliers in DBpedia are found in the date/time related values and the reason is that the extraction tool can not always extract correctly the information related to date/time. For example `198?-85` in Wikipedia[10], parsed to `198^^xsd:integer` in DBpedia[11]. We should note that, although this type of anomaly can be successfully detected by ExPAD, however explaining them with respect to other variables may not be informative due to independentness. Therefore DBpedia can not be an adequate option for investigation because it contains many independent predicates that do not correlate with each other. To this end, and as there is no adequate RDF dataset benchmark, three datasets have been exploited. Engie-accident dataset[12], Titanic dataset[13], and our generated synthetic dataset. Engie SA[14] is a French multinational electric utility company that operates in the fields of energy transition, generation, and distribution, natural gas, nuclear, renewable energy, and petroleum. The accident RDF dataset contains data about ~ 57K car accidents that occurred in France in 2018 (such as geo-coordinate, weather conditions, ...). The Titanic dataset contains information about passengers of the Titanic shipwreck. The dataset contains 12 features for 891 passengers. Besides this and to be able to have different sizes of datasets, we implemented an RDF data simulator that generates synthetic RDF graphs. For the RDF data generator, we consider *Person* class with 5 synthetic properties and the respective distribution listed in Table 6.2.

6.3.2 Experiment A: Assessment of the detected Anomalies Explanation

In this section, we analyze the detected anomalies and explanations for all three mentioned datasets. A synthetic data set with manually added anomalies, an Engie accident dataset, and an RDFized version of the Titanic dataset. For the synthetic data, we generated $100K$ triples with the mentioned distribution in Table 6.2. For the anomalies, we added 0.02% (= 20 cases) anomalous data; 10 pregnant women with age $\in [90, 100]$, 5 presidents with age $\in [4, 10]$, and 5 students with age $\in [70, 90]$. The reason why we only added a few anomalous cases is that anomalies should rarely happen, and in case they occur frequently they can not be detected. It is worth noting that the 5 generated predicates in the simulated data are deliberately defined as dependent (e.g. pregnancy is related to gender).

[10] https://en.wikipedia.org/wiki/Steve_Walters
[11] https://dbpedia.org/page/Steve_Walters
[12] can not be publicly published due to intellectual property concerns
[13] https://www.kaggle.com/c/titanic
[14] https://www.engie.com

Table 6.2: ExPAD: Predicates used for generating synthetic RDF graph for class *Person*

Predicate	**Value Type**	**Example**	**Distribution**
id	non negative integer	`{0,1,2,...}`	incremental starting from 0
gender	boolean	`{male,female}`	50% male, 50% female
job	URI	`{Student,President}`	50% student, 50% president
pregnant	boolean	`{true,false}`	if male then false, if job=student then false, if job=president and age>55 then false, if job=president and age<=55 then 40%
age	positive integer	`{1,2,3,...}`	if job=student in [7,14], if job=president in [25,70]

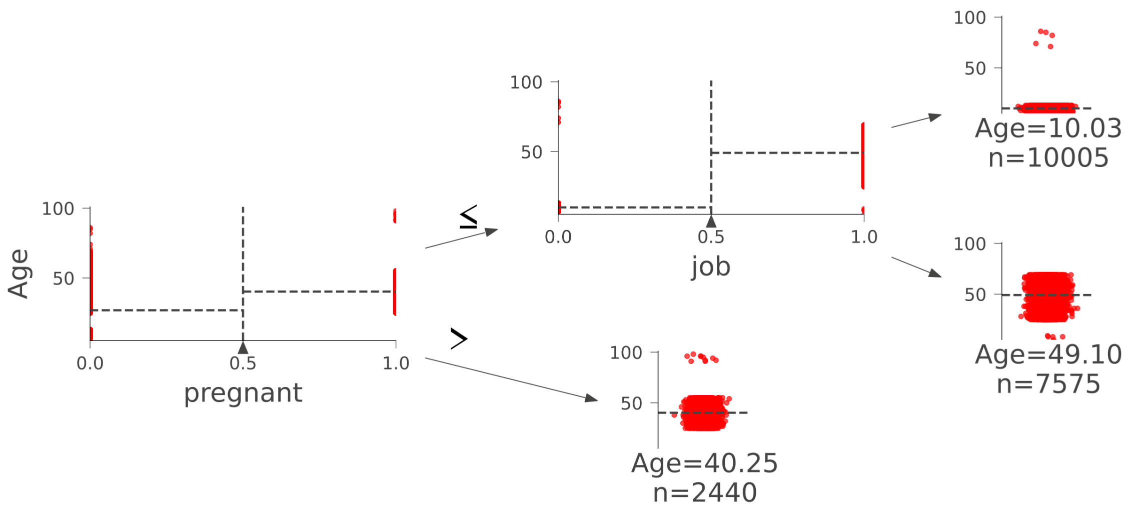


Figure 6.3: ExPAD: trained decision tree with the target variable *age*

Although adding other independent predicates (such as weather conditions) won't cause any issues for the workflow, however, our approach naturally (due to the nature of the decision tree algorithm) can not generate meaningful explanations for the independent variables (age of a person is not correlated to the weather condition and therefore can not be explained meaningfully via it).

In this scenario, we set the maximum depth of the tree to two, used Literal2Feature for extracting features, and used IQR for anomaly detection. Note that all of these values are configurable for the end user. Moreover, in this scenario, we only focused on the *age* predicate, although ExPAD performs on all the numeric variables. Figure 6.3 depicts the fitted decision tree on the 100K synthetic data with *age* as the target variable.

For the accident dataset, we used Literal2Feature for extracting features (due to the nested architecture of RDF in this dataset), used IQR for anomaly detection, and set the maximum depth of the tree to two. In this case, we executed ExPAD on the geo-coordinates of the accidents.

For the Titanic dataset, we used RDFizer [152] to transform Comma Separated Value (CSV) data to RDF. It is beyond the scope of this chapter to explain how RDFizer works, however, it uses RDF Mapping Language (RML)[15] rules for the transformation of (un)structured data into RDF knowledge graphs. For this scenario, we only used the training dataset provided by Kaggle[16]. For feature extraction we used pivoting approach, for anomaly detection IQR has been used, and set the maximum depth of the tree to three.

After applying ExPAD on all three datasets, all 20 cases from synthetic data, 9 cases from the Titanic dataset, and 5 cases from the accident dataset have been detected. Table 6.3 shows some sample-detected anomalies with the corresponding explanation. It is worth noting that there is a positive correlation between decision tree depth and the complication of explanation (number of variables reported in the explanation). Obviously, as we go deeper, the explanation may contain more variables. Comparing the detected anomalies from the Titanic dataset with OutlierTree [137] results on

[15] https://rml.io/specs/rml

[16] https://www.kaggle.com/competitions/titanic/data?select=train.csv

Table 6.3: ExPAD: Sample of detected anomalies and their explanation

Dataset	Anomaly	Explanation
$100k$ Synthetic	`age = 100`	age = 100 is anomalous given pregnant = $true$
	`age = 80`	age = 80 is anomalous given pregnant = $false$ and job = $student$
	`age = 5`	age = 5 is anomalous given pregnant = $false$ and job = $president$
Titanic	`Fare = 0`	$Fare$ = 0 is anomalous given Pclass = 3 and SibSp = 0
Accident	`longitude = 50`	$longitude$ = 50 is anomalous given location='$France$'

Table 6.4: ExPAD: Synthetic dataset description

Dataset	File Size	#Triples
$DS1$	$500MB$	$1M$
$DS2$	$2.6GB$	$5M$
$DS3$	$5.2GB$	$10M$
$DS4$	$26GB$	$50M$
$DS5$	$52GB$	$100M$

the same dataset, revealed that ExPAD detected the anomalies correctly. Moreover, manual inspection of the detected anomalies in the accident dataset, reveals that although the accidents' location in the dataset is 'France Métropole', however, their geo-coordinates show that they lay outside France, in Asia and eastern Europe (Figure 6.4).

6.3.3 Experiment B: Scalability

In this experiment, we evaluate the scalability of ExPAD by different data sizes and varying cluster processing setups. To be able to have different sizes of datasets, we use the same RDF data simulator explained earlier to generate big synthetic RDF graphs. Table 6.4 listed the generated datasets and their characteristics (worth mentioning that the German DBpedia size is 48GB). As the running time of reading data from HDFS is the same for all the configurations, and as the scalability of Literal2Feature has been investigated in [23], for these experiments, we neglect the execution time of these components.

Scalability over the number of cores

To adjust the distributed processing power, the number of available cores was regulated. In this experiment, we selected *DS3* (5.2GB) as a pilot dataset and the number of cores was increased starting from $2^2 = 4$ up to $2^7 = 128$. The experiments were carried out on a small cluster of 4 nodes (1 master, 3 workers): AMD Opteron(TM) CPU Processor 6376 @ 2300MHz (64 Cores), 256 GB RAM.

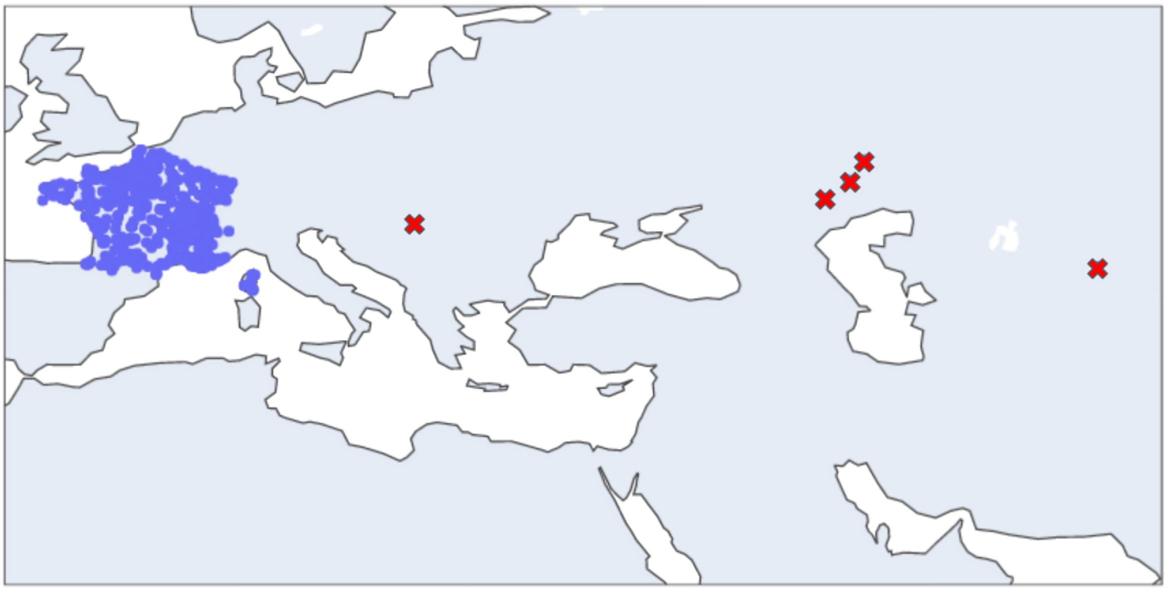


Figure 6.4: ExPAD: detected anomalies on the Accident Dataset

Moreover, the machines were connected via a Gigabit network. All experiments are executed three times and the average value is reported in the results. Moreover, in these experiments, we set the maximum depth of the tree to 2, used pivoting for extracting features, and used MAD for anomaly detection. Figure 6.5 shows the scalability over different computing cluster setups. It is clear that increasing the computational power horizontally decreases the execution time. Initially, doubling the number of cores reduces the execution time by nearly a factor of two. However, by adding more cores, the execution time only slightly decreases. This phenomenon is caused by the overhead of moving data between nodes as well as network latency. The maximum speed-up is 5.1x.

Scalability over dataset size

To analyze the scalability over different datasets, we fix the computational power to 32 cores and run the experiments for all datasets introduced in Table 6.4. By comparing the run-time as shown in Figure 6.6, we note that the execution time does not increase neither linearly nor exponentially. This behavior is due to the distribution among available resources e.g. (memory) and partition size. It can be seen that by increasing the dataset size from 500 MB to 52 GB (~ 100 times bigger), the execution time is almost only 19 times higher.

Scalability over the number of features

For checking anomalies in each feature, one decision tree should be trained, therefore the complexity of ExPAD with respect to the number of features is linear. However, to show how decision tree training of ExPAD behaves when the data set dimensionality increases, we added 1000 uninformative numeric features to the *DS3* dataset which are drawn from a Gaussian distribution. Dealing with these random numbers can be problematic for decision trees. Also, we fixed the computational power to 32 cores, max decision tree depth to 3, and set *age* as the target variable. Figure 6.7 shows that increasing

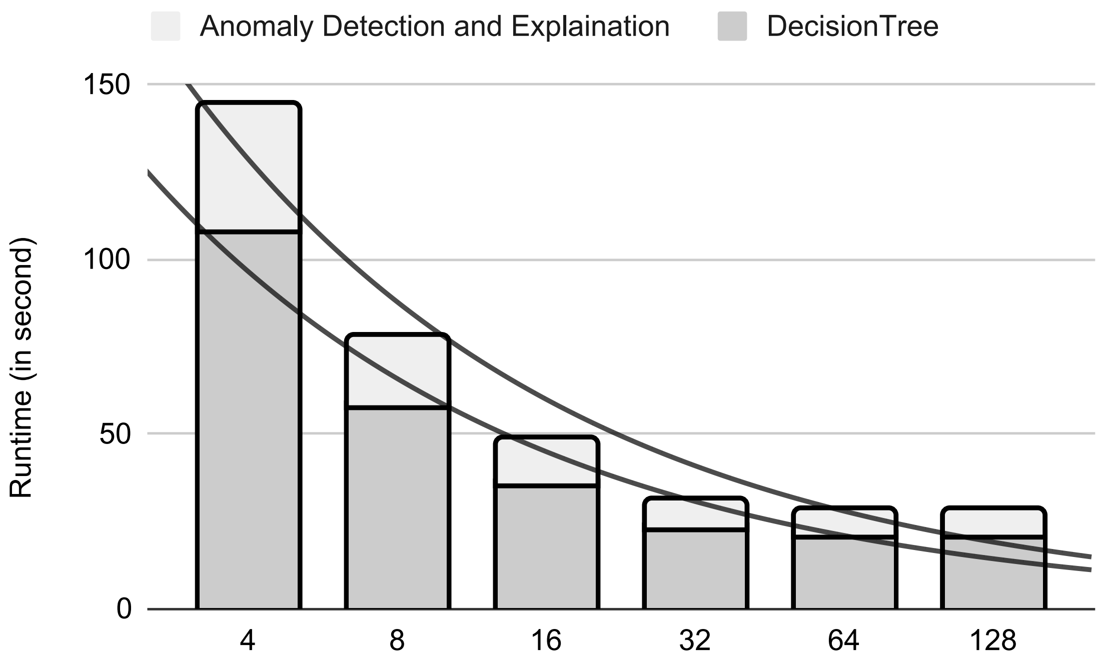


Figure 6.5: ExPAD evaluation: processing power scalability on *DS3*

the dimensionality increases the running time almost linearly. This is happening due to the efficient implementation of the decision tree in Spark. So it can be seen the framework can handle even a large number of features.

6.4 Summary

In this chapter, we introduced ExPAD, a generic, distributed, and scalable framework for explainable numeric anomaly detection in KGs. ExPAD is open-source, available on GitHub, and integrated into SANSA Stack. Inspired by OutlierTree, ExPAD generates human-readable explanations for outlier identification, by traversing supervised decision tree splits.

As ExPAD operates on univariate data, it may not be able to detect all multi-dimensional outliers (as opposed to other methods such as Isolation Forest that consider multiple variables simultaneously), however, it generates meaningful explanations for the dependent features. Moreover, our experiments show that the framework by detecting and explaining abnormalities can help in enhancing the data quality in KGs. Furthermore, our results indicate that ExPAD can be successfully scaled across a cluster of nodes for very large data sets.

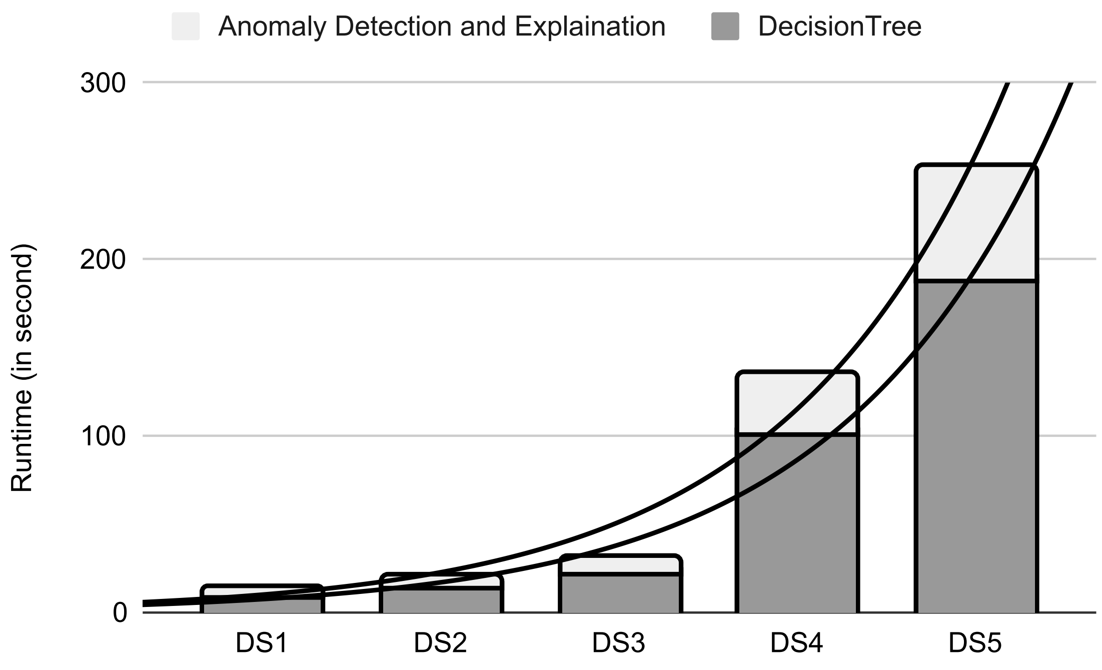


Figure 6.6: ExPAD evaluation: size-up performance evaluation over 32 Cores

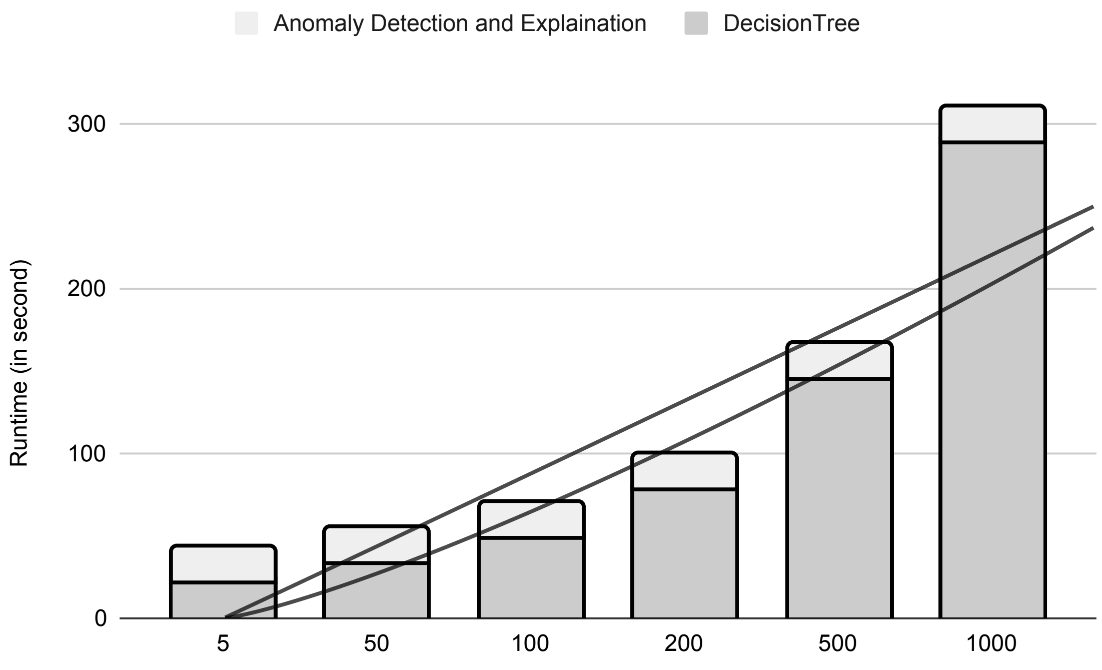


Figure 6.7: ExPAD evaluation: running time of ExPAD over a large number of features

CHAPTER 7

Implementation and Use-Case

Acknowledgement This chapter is based on our scientific publications:

- **Farshad Bakhshandegan Moghaddam**, Carsten Felix Draschner, Jens Lehmann, Hajira Jabeen, "Semantic Web Analysis with Flavor of Micro-Services", LAMBDA Doctoral Workshop 2021, http://ceur-ws.org/Vol-3195/paper1.pdf

- Carsten Felix Draschner, **Farshad Bakhshandegan Moghaddam**, Jens Lehmann, Hajira Jabeen, "Semantic Analytics in the Palm of Your Browser", LAMBDA Doctoral Workshop 2021, http://ceur-ws.org/Vol-3195/paper2.pdf

The preceding chapters have outlined the algorithmic and technical approaches employed by Literal2Feature, DistAD, and ExPAD. However, this chapter delves into the standardization and accessibility of these approaches, transforming them into readily available resources. Furthermore, it explores the practical implementation of these newly developed tools and frameworks, examining their testing capabilities within Databricks and Zeppelin notebooks, as well as their integration potential through REST API.

Section 7.1 provides a detailed account of the technical deployment. It highlights the steps taken to effectively bring these solutions into operation. Additionally, Section 7.2 illustrates how these technologies can be thoroughly tested and utilized within the browser environment, leveraging the capabilities of the Platform-as-a-Service provider, Databricks. Moreover, Section 7.3 presents an alternative deployment scenario wherein the technology can be employed on-site, utilizing Zeppelin notebooks or adopting a REST API architecture. This section explores the practical aspects and considerations involved in implementing these approaches, catering to the specific needs and preferences of users.

By expanding on these topics, this chapter offers a comprehensive overview of the standardized resources, testing methodologies, and deployment options associated with proposed frameworks in this thesis. It equips readers with the necessary knowledge to leverage these innovative technologies effectively in various settings and configurations.

7.1 Resources

Every project created as part of this thesis was carefully developed and released in accordance with a well-defined structure. The technical components driving these projects have been made fully open source and are readily accessible through the dedicated GitHub repository. To aid users in understanding and utilizing these components effectively, comprehensive documentation is available on the GitHub page, as well as within the repository itself.

The repository documentation has been thoughtfully organized and structured, making it easily navigable for users. It can be accessed through GitHub pages, ReadMe files, and corresponding release notes. These resources provide detailed insights into the functionality, implementation, and usage guidelines of the projects. The availability of explicit schemas, open-source code, and comprehensive documentation empowers users to explore and contribute to the ongoing evolution of these projects, promoting further innovation and advancement.

7.1.1 GitHub Repository

The complete body of work can be readily accessed through the dedicated GitHub repository of the SANSA stack[1]. Within this repository, multiple projects are available, with the primary focus centered around the central SANSA stack. This central repository serves as a unifying platform, consolidating the historical projects that were previously segregated by layers. In addition to the central SANSA stack, the repository also contains separate projects that offer technical implementations for Zeppelin Notebook, Databricks, and REST API functionalities. These projects are designed for particular situations and offer unique features and integrations. To facilitate ease of use and comprehension, the central repository offers comprehensive documentation through detailed ReadMe files and GitHub pages. These resources serve as valuable references, guiding users in understanding the various aspects of the projects and assisting in their effective implementation. Moreover, the GitHub repository includes a strong testing system that uses GitHub actions to run unit tests. This helps make sure that the projects are of high quality and dependable, ultimately improving their performance. By bringing together the projects, offering detailed documentation, and using effective testing methods, the GitHub repository for the SANSA stack becomes a complete platform for accessing and using the different parts of the SANSA stack. It allows users to explore what's available, understand how they work, and easily incorporate them into their own work processes and systems.

Repository Structure

The central SANSA project repository offers a well-structured organization of both code and documentation, categorized into distinct layers. These layers encompass the RDF layer for efficient management of input and output, the Query layer designed for executing SPARQL queries, the OWL layer dedicated to ontology analytics, the Inference layer for reasoning capabilities, and the ML layer specifically focused on data analytics and machine learning approaches (Figure 7.1).

Within the scope of this thesis, contributions were made to the ML layer of the SANSA stack. The emphasis was placed on developing and enhancing anomaly detection functionalities, allowing users to leverage AD techniques on their data. By structuring the repository based on these layers, it becomes easier for users to navigate and locate the relevant code and documentation pertaining to their

[1] https://github.com/SANSA-Stack

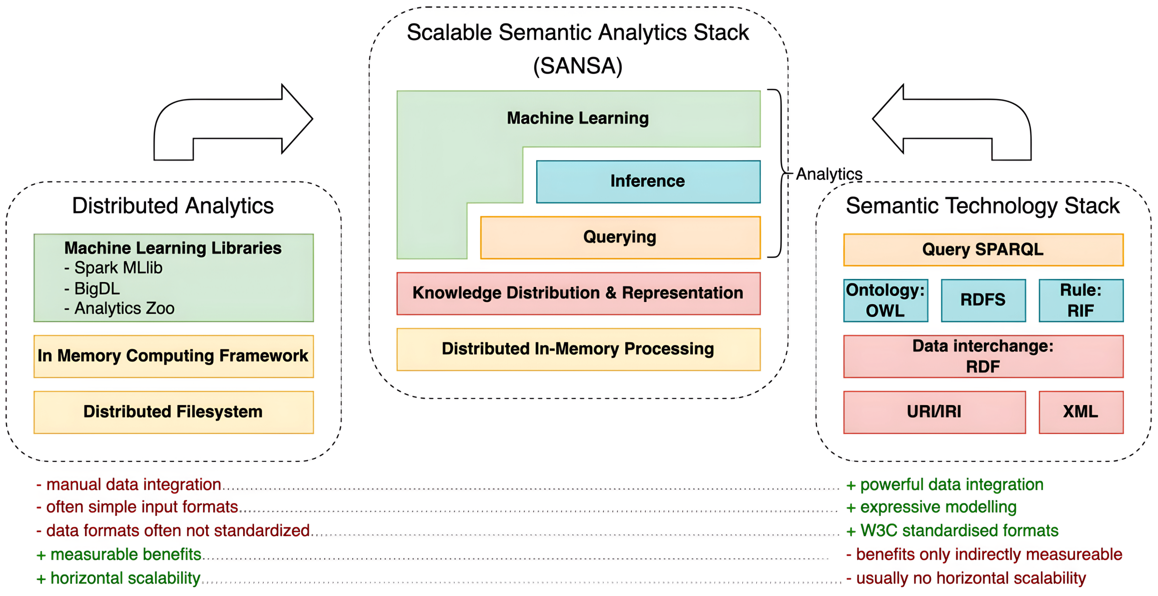

Scalable Semantic Analytics Stack (SANSA)
Machine Learning
Inference
Querying
Analytics
Knowledge Distribution & Representation
Distributed In-Memory Processing
Distributed Analytics
Machine Learning Libraries
- Spark MLlib
- BigDL
- Analytics Zoo
In Memory Computing Framework
Distributed Filesystem
Semantic Technology Stack
Query SPARQL
Ontology: OWL
RDFS
Rule: RIF
Data interchange: RDF
URI/IRI
XML
- manual data integration
- often simple input formats
- data formats often not standardized
+ measurable benefits
+ horizontal scalability
+ powerful data integration
+ expressive modelling
+ W3C standardised formats
- benefits only indirectly measureable
- usually no horizontal scalability

Figure 7.1: SANSA stack structure

specific requirements. This organized approach ensures a coherent and comprehensive understanding of the SANSA stack, enabling users to effectively utilize the functionalities provided by each layer.

7.1.2 Documentation

The documentation accompanying the new modules offers a comprehensive overview of each module's concept, providing practical insights and helpful tutorials. To facilitate implementation, the documentation includes exemplary code snippets, Scala docs, and sample Databricks notebooks. Users can effortlessly navigate between the documentation and the codebase, allowing for quick and convenient reference. By providing a range of resources and linking them closely, the documentation enables users to easily locate the necessary information, empowering them to understand and utilize the modules effectively.

Sample Code and Collaborative Environments

To facilitate the understanding and adoption of the technical approaches, namely Literal2Feature, DistAD, and ExPAD, a collection of example code snippets, classes, and notebooks have been developed. These resources are designed to minimize the initial barriers of trial and error, providing users with practical demonstrations of feature extraction, anomaly detection, and explainable anomaly detection. The code snippets serve as concise and illustrative examples, showcasing the implementation details of key functionalities. Additionally, the development of dedicated classes offers reusable components that can be seamlessly integrated into users' own projects.

Moreover, to enhance the accessibility of the example pipelines showcased, we have created notebook versions of these examples. This approach ensures that the example code snippets are accompanied by comprehensive documentation and generated data. Given that the SANSA stack relies on Apache Spark, we utilize Apache Zeppelin for executing local notebooks and Databricks as a Platform-as-a-Service (PaaS) provider for cloud-based notebooks.

In Section 7.2, we provide a detailed guide on how to instantiate these notebooks within the Databricks environment. This section outlines the necessary steps and configurations required to set up and utilize the example pipelines effectively. Furthermore, in Section 7.3, we introduce the execution of the example pipelines in both the REST API and Zeppelin environments, offering users the flexibility to choose the most suitable execution approach based on their specific requirements and preferences.

By offering notebook versions of the example pipelines and providing guidance on utilizing them in both local and cloud-based environments, we aim to ensure a seamless and accessible experience for users. This approach allows users to experiment, execute, and analyze the example pipelines using familiar tools and platforms, enabling a smoother learning and implementation process.

7.1.3 Releases

New releases have been generated for the developed frameworks Literal2Feature, DistAD, and ExPAD [23–25]. These releases encompass detailed explanations of the latest features and provide direct links to relevant documentation, examples, Scala docs, example data, and associated publications. By aligning with the current code state, these releases enhance the reproducibility of the experiments and evaluations conducted in the corresponding publications.

1. Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor & DistRDF2ML: Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs
   - *Release Name*: DistRDF2ML & Literal2Feature Release
   - *Version Number*: v0.8.1_DistRDF2ML
   - *Link*: `https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1_DistRDF2ML`
2. DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs
   - *Release Name*: DistAD Release
   - *Version Number*: v0.8.3_DistAD
   - *Link*: `https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.3_DistAD`
3. ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs
   - *Release Name*: DistAD Release
   - *Version Number*: v0.8.5_ExPAD
   - *Link*: `https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.5_ExPAD`

7.2 Scalable Semantic Analytics within PaaS

7.2.1 Motivation

All the introduced scalable and distributed frameworks in Chapters 4, 5, and 6 allow scalable processing of knowledge graph data across multiple machines using Apache Spark. Moreover, all of these frameworks have been implemented and integrated into the Scalable Semantic Analytics Stack SANSA [19]. SANSA leverages Apache Spark and Apache Jena to provide an open-source framework for start-to-end data analytic pipelines for large-scale RDF knowledge graphs [23–25, 148, 153]. However, one significant challenge when working with the SANSA stack is setting up cluster computing. It can be technically demanding and requires access to suitable hardware, as well as expertise in installing and connecting the necessary components. To reduce this hurdle, we utilize Platform as a Service (PaaS) providers and show how users who want to build large-scale RDF Knowledge graph data analytic pipelines can use SANSA with minimal technical requirements and entry hurdles. The main contributions of this section are the following:

- Introduction of scalable semantic analytics stack SANSA through Databricks
- Sample coding notebooks for hands-on ML on RDF KGs
- Guideline on how to set up third-party Apache Spark frameworks in PaaS

7.2.2 SANSA through Databricks

A complex and heterogeneous framework like SANSA requires several technical prerequisites to run initial experiments. On the one hand, the computation is done in memory and it is crucial to have enough memory to manage the data, On the other hand, the computation is done on the CPU side. Apache Spark is designed for multi-core and cluster computation. To use the framework, Apache Spark must be available in the required version (for now, Spark 3.x and also Scala in version 2.12). Setting up this hardware and software can be eased by using Databricks since even in the community edition (free plan), a two-core system with 15GB of memory is already available. Furthermore, there are predefined images like the combination of different Apache Spark versions and Scala versions. The following sections will guide the users through the setup, and explain working with SANSA on RDF data.

Get Access to Platform

Databricks [22] is one of several Platform as a Service (PaaS) providers. It offers the simplicity of setting up an Apache Spark instance for use on notebooks and making it accessible through notebooks in a user-friendly way. For registering Databricks in the free plan, the Community Edition is suitable. More information can be found on the Databricks FAQ[2].

Uploading Data

Once logged in to the platform, an opportunity is given to import libraries. The SANSA stack needs to be uploaded as a *jar* file. The jar file can be fetched from the most recent release in the SANSA

[2] https://www.databricks.com/product/faq

stack GitHub page[3]. The name will be given automatically according to the filename of the jar. Due to the size of the jar, the upload process may take a few minutes. After the upload is done, the process can be confirmed with the *Create* button. Next, we need to make our desired data available. In the next step, we add the knowledge graph data to our Databricks file system. We will introduce the usage of the SANSA stack based on the Linked Movie Database dataset [154] which is a LOD RDF dataset containing 40 thousand movies and their title, runtime, list of actors, genres, and publish date. This dataset represents a multimodal knowledge graph for several example pipelines. The import can be started from the main pages *Import and Explore data*. In the overlay menu, one could drag and drop the file. Once the data is uploaded, the menu shows the path where it was stored. An example might be: "*FileStore/tables/linkedmdb-18-05-2009-dump.nt*".

Setup Cluster

One must set up a cluster in the platform used for executing the notebooks. First, create the cluster as a new cluster and give it a unique name like the *SANSA-tryout cluster*. Next, select the fitting image named Spark Runtime Version to the pair Scala 2.12 and Apache Spark 3.x. Then specify the spark config by pasting the following three key-value pairs shown in Figure 7.2 and Listing 7.1. They correspond to the default Databricks and SANSA Spark setup. This Cluster configuration has to be confirmed with *Create Cluster*. Confirmation over *create a cluster* opens up the overview of the respective created cluster. In the *Libraries* tab, it is needed to *install new* the previously uploaded SANSA jar. The uploaded jar is within the user's workspace. This process has to be confirmed with *install*. After some seconds the SANSA library will change status from *installing* to *installed*.

Setup Notebook

Now, the user has to open up or create a desired notebook. Either one can start with a blank notebook, but it is easier to use the provided sample notebooks[4]. These sample notebooks can be imported by using the import option from the users' workspace. In the pop-up window, one can import the notebook over the notebook URL. The import will directly add the notebook to the workspace and open it up.

Listing 7.1: Spark Configuration modules

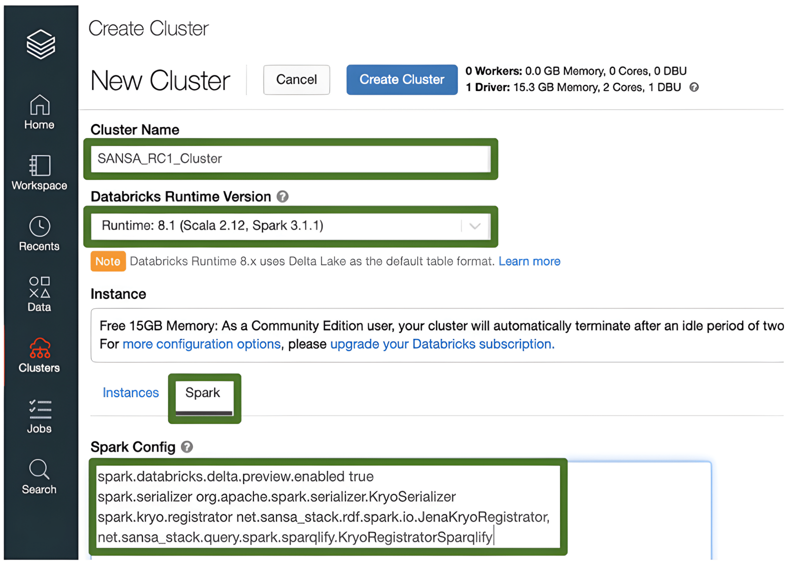
```
spark.databricks.delta.preview.enabled true
spark.serializer org.apache.spark.serializer.KryoSerializer
spark.kryo.registrator net.sansastack.rdf.spark.io.JenaKryoRegistrator,net.
    sansastack.query.spark.sparqlify.KryoRegistratorSparqlify
```

Execution of Sample Notebooks

The notebook needs to be assigned to a cluster. The cluster should be present as previously configured (see Figure 7.3) and contain the SANSA framework as a library. After selecting the cluster, it gets attached and will be ready after some seconds. This enables the execution of notebook cells with SANSA module functionalities.

[3] https://github.com/SANSA-Stack/SANSA-Stack/releases

[4] https://github.com/SANSA-Stack/SANSA-Databricks



0 Workers: 0.0 GB Memory, 0 Cores, 0 DBU
1 Driver: 15.3 GB Memory, 2 Cores, 1 DBU

Cluster Name
SANSA_RC1_Cluster

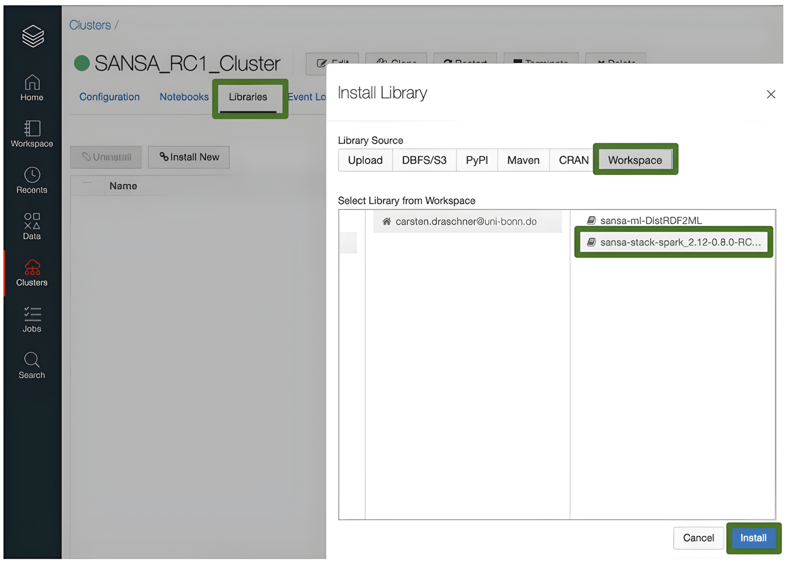
Databricks Runtime Version
Runtime: 8.1 (Scala 2.12, Spark 3.1.1)

Note Databricks Runtime 8.x uses Delta Lake as the default table format. Learn more

Instance
Free 15GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two
For more configuration options, please upgrade your Databricks subscription.

Spark Config
```
spark.databricks.delta.preview.enabled true
spark.serializer org.apache.spark.serializer.KryoSerializer
spark.kryo.registrator net.sansa_stack.rdf.spark.io.JenaKryoRegistrator,
net.sansa_stack.query.spark.sparqlify.KryoRegistratorSparqlify
```

Figure 7.2: Configuration of cluster



Library Source
Upload | DBFS/S3 | PyPI | Maven | CRAN | Workspace

Select Library from Workspace
carsten.draschner@uni-bonn.de
sansa-ml-DistRDF2ML
sansa-stack-spark_2.12-0.8.0-RC...

Figure 7.3: Installation of SANSA library

7.2.3 Summary

This section illustrates the ease of accessibility of a comprehensive and integrated framework for scalable semantic analytics through the demonstration of sample notebooks hosted and running within the Databricks platform provider service. This guide serves as a starting point for users to explore and adapt their own semantic data analytical pipelines.

By leveraging the infrastructure of Databricks, users are relieved from the need for a dedicated hardware setup with sufficient computational power and main memory, thus simplifying the initial steps of the process. Furthermore, the installation and management of the necessary Scala and Spark versions are automatically handled, eliminating potential compatibility issues. The provided code within the sample notebooks is designed to seamlessly run and scale across distributed Spark clusters.

7.3 Micro-Service based Semantic Analytics integration

7.3.1 Motivation

To be able to use SANSA effectively, a cluster of Spark nodes with an HDFS file system is required. Even by having such a cluster, one can only interact with SANSA via a terminal. Establishing a cluster of different nodes containing Spark and Hadoop[5] and configuring them is by nature a cumbersome task and needs a lot of knowledge and experience. Moreover, in the case of having technical knowledge, establishing such a cluster is a time-consuming task. In addition, end-users prefer to have a user-friendlier way to interact with SANSA without having any programming and scripting knowledge. Therefore, in this section, we introduce a micro-service architecture of a SANSA-enabled Spark and Hadoop cluster with 2 user-friendly interactive communication mechanisms aka. REST API and Zeppelin Notebook[6]. Our introduced architecture is based on Docker technologies[7]. Moreover, our sample explanatory tutorial enables non-technical users to easily use SANSA without having any specific knowledge and skills.

The main contributions of this section are the following:

- Introducing a micro-service architecture of Big Data tools such as Apache Spark, Apache Hadoop, Apache Zeppelin, HDFS File Browser, and Apache Livy
- Introducing for the first time REST APIs for the SANSA stack
- Introducing an interactive Notebook (i.e. Apache Zeppelin) for interacting with SANSA

7.3.2 Architecture

In this section, we present the micro-service system architecture for using SANSA. Worth mentioning that the framework is open-source and hosted on GitHub[8]. The main goal of the framework is to bring a simple and effortless approach to setting up a Spark cluster with all the requirements without having extensive computer science knowledge.

7.3.3 Components

We provided two interaction mechanisms for the end-users. (i) Zeppelin Notebooks, and (ii) REST APIs. Depending on the scenario users may select one of the mentioned mechanisms. These mechanisms cover the full spectrum from simplicity to flexibility. By using the REST APIs, users will be able to call predefined functionalities from SANSA without any effort. However, in case a user is interested

[5] https://hadoop.apache.org/
[6] https://zeppelin.apache.org/
[7] https://www.docker.com/
[8] https://github.com/SANSA-Stack/SANSA-Rest-API

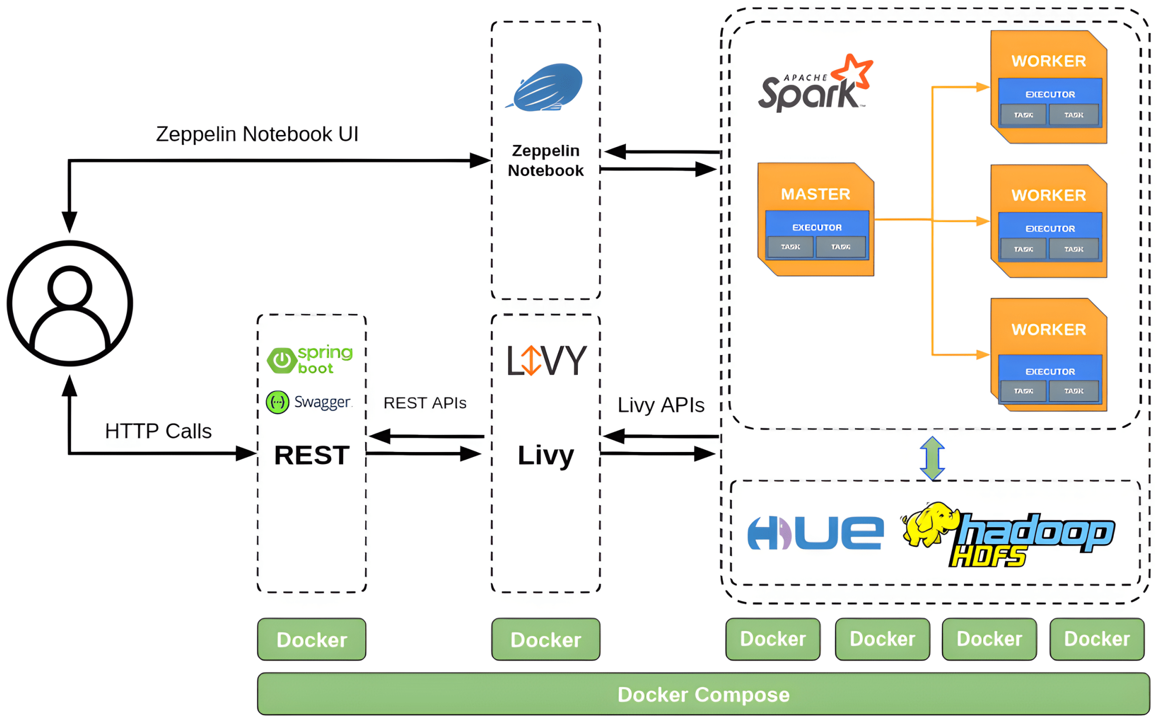


Figure 7.4: High-level micro-service architecture overview

in the new functionalities, they can write code and stack their code via Zeppelin Notebook and submit their task to the Spark cluster. Figure 7.4 depicts the high-level system overview. The architecture contains 4 main components i.e. (i) Java-based REST APIs, (ii) Apache Zeppelin Notebook, (iii) Apache Livy, and (iv) Spark-Hadoop Cluster, which are all utilized via Docker-Compose.

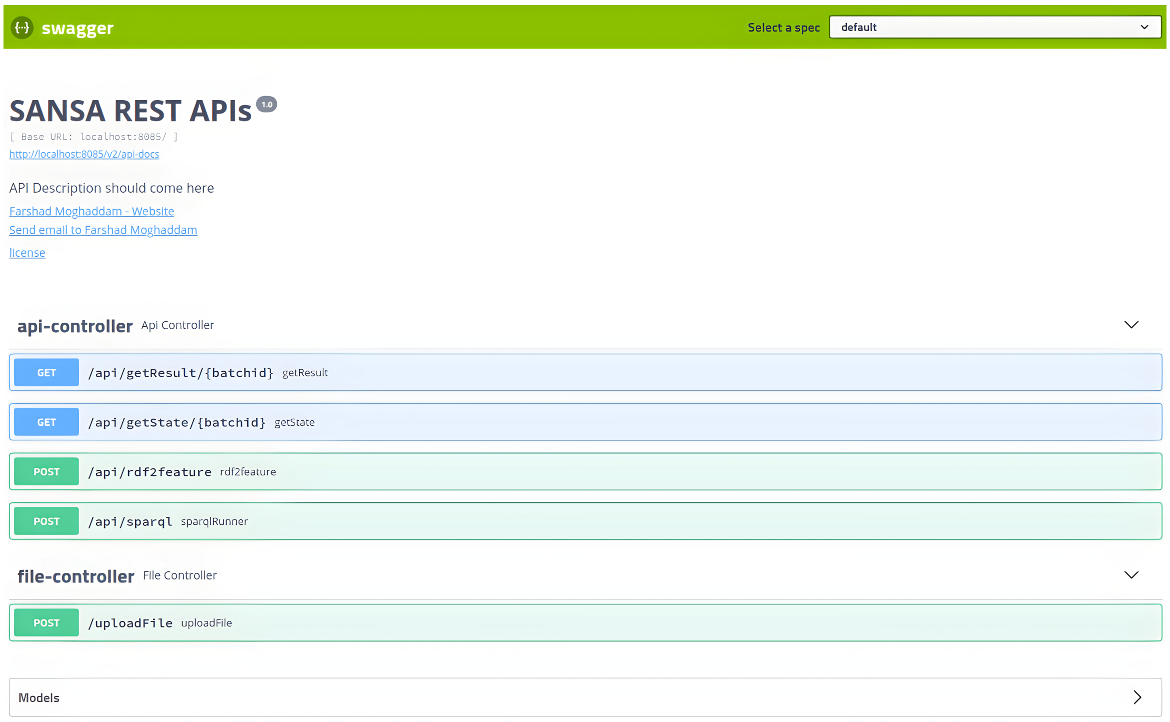
Java-based REST API

This layer provides functionality for the end-user to interact with SANSA via REST APIs. It is Java-based and is powered by Spring Boot[9] technology and contains a Swagger[10] UI which enables users to easily call any provided functions via a browser. Figure 7.5 shows the SwaggerUI and provided APIs. A sample scenario of how to use these APIs is provided in Section 7.3.4.

Apache Livy REST APIs

As the Spark tasks may be long-running (up to a few days) and there is a chance that the node that is running the task crashes and loses the calculations, therefore, directly connecting the REST APIs to the Spark cluster is not feasible due to asynchronous nature of such computations. To tackle this, another layer has been added by using Apache Livy[11], which is able to keep track of Spark sessions and calculation states. Livy enables programmatic, fault-tolerant, multi-tenant submission of Spark

[9] https://spring.io/projects/spring-boot
[10] https://swagger.io/
[11] https://livy.apache.org/



swagger Select a spec: default

SANSA REST APIs 1.0

[Base URL: localhost:8085/]

http://localhost:8085/v2/api-docs

API Description should come here

Farshad Moghaddam - Website

Send email to Farshad Moghaddam

license

api-controller Api Controller

```
GET   /api/getResult/{batchid}  getResult
GET   /api/getState/{batchid}   getState
POST  /api/rdf2feature          rdf2feature
POST  /api/sparql               sparqlRunner
```

file-controller File Controller

```
POST  /uploadFile  uploadFile
```

Models

Figure 7.5: REST APIs Swagger UI

jobs from web/mobile apps. So, multiple users can interact with the Spark cluster concurrently and reliably. Although the Livy interface is available for the user, this layer works as a background process and the user does not need to interact with it directly, because all the functionalities will be handled by the REST API layer.

Spark-Hadoop Cluster

To be able to run a SANSA functionality, having a Spark cluster with a Hadoop file system is inevitable. To do so we containerized Spark, Hadoop Namenode, Hadoop Datanode, and Hue HDFS file browser[12] in publicly available docker images. Moreover, we configured the containers to interact with each other seamlessly via Docker-Compose. All the other layers have been containerized and exposed in the same docker-compose file. For the sake of reproducibility, Appendix A presents the `docker-compose.yml` file.

7.3.4 Usage

To be able to run the cluster, the user needs to clone the SANSA stack from GitHub and navigate to *sansa-rest* sub-folder. The following codes in the terminal will bring up the cluster.

[12] https://gethue.com/

```
$ git clone https://github.com/SANSA-Stack/SANSA-Stack.git
$ cd SANSA-Stack/sansa-rest
$ make
$ make up
```

To stop the cluster, the user only needs to run the following command.

```
$ make down
```

Table 7.1 lists all the available endpoints and their functionalities.

Table 7.1: Available endpoints and their functionalities

Endpoint	**Functionality**
http://localhost	Zeppelin Notebook
http://localhost:8085	REST Swagger UI
http://localhost:8998	Livy UI
http://localhost:8080	Spark Master UI
http://localhost:8088/home	Hue file browser UI

As already mentioned, users will have two interaction mechanisms to connect to SANSA, either using Zeppelin Notebook or REST APIs. Using Zeppelin Notebook is straightforward same as all the other notebook technologies such as Jupyter Notebook[13]. Therefore, without losing the generality and due to the space issue, we ignore the explanation in this section. However, in the following, we explain a sample scenario that shows how to use the REST APIs. Besides its many functionalities, SANSA provides a distributed SPARQL engine (i.e. Sparklify [141]) which can execute a SPARQL query in a distributed fashion. As a scenario, imagine the user has an RDF file (any format) and would like to run a SPARQL query over it. To be able to use a REST API for this purpose, the user first needs to upload her file into the HDFS because SANSA needs to access the file in a distributed manner. For this reason, we have provided an API in Swagger that enables the user to upload files to the HDFS. The result of the API call will be the address to which the file will be stored in the HDFS. The user will need to provide this address for any subsequent API call. To call the SPARQL engine API (i.e. `/api/sparql`), the user simply needs to provide the SPARQL query and the address of the file which he retrieved from the file upload API. Keep in mind that the result of any API call will be a *livy batch id*. This *id* is a unique number that identifies a Livy session that is responsible for the task computation. We provided two APIs that receive this *id* and provide more information about the execution process and the result of the executions. The result of the `/api/getState` API will be either `running`, `success`, or `dead`. Only in case of `success`, the user can use `/api/getResult` API to see the result of the call. The other two states either show the process is ongoing, or the process is unexpectedly stopped.

[13] https://jupyter.org/

7.3.5 Summary

In this section, our efforts revolve around providing interactive opportunities that can be used for the newly developed distributed scalable in-memory data analytics and approaches introduced in Chapters 4, 5, and 6, as well as for the Semantic Analytics Stack as a whole. By adopting a containerized approach and distributing resources as open source, we aim to facilitate easier and more accessible utilization of our proposed developments, models, and ML pipelines.

This technology empowers users to host SANSA as a service, enabling scalable processing of RDF KG data. Additionally, a hosted version of SANSA as a service caters to less technically proficient users, allowing them to execute RDF KG data analytics through a web-based interface without the complexities and time-consuming setup overhead typically associated with such tasks.

CHAPTER 8

Conclusion and Future Directions

In this thesis, we have focused on the problem of scalable distributed anomaly detection on knowledge graphs. To address this problem, we have made several contributions in the areas of scalable KG feature extraction, anomaly detection on KGs, and explainable anomaly detection on KGs. These contributions have allowed us to address the challenges initially stated in our research, and have led to the development of novel methods and techniques for detecting anomalies in KGs. In the following sections of this chapter, we will provide a summary of our contributions and explain the main results that support the validity of our research questions. By doing so, we aim to provide a comprehensive overview of the work we have undertaken in this thesis and its impact on the field of anomaly detection on KGs.

8.1 Review of the Contributions

In this section, we will thoroughly explain the contributions of the current Ph.D. thesis. We'll clarify how it addresses the challenges we encountered and provide evidence-based solutions to the research questions. The main goal of this thesis is to enhance the ability to detect anomalies in knowledge graphs at a scalable level. We have made contributions that successfully answer three key research questions. Let's go back and review the research questions outlined in this thesis.

First, we tackled the problem of extracting features from the large-scale RDF datasets and answering the following research question:

RQ1: Can we vectorize knowledge graphs in a scalable and distributed manner?

Our work encompasses the development of the Literal2Feature module (detailed in Chapter 4), which plays a crucial role in facilitating the creation of interpretable feature vectors from RDF knowledge graphs. One of the key strengths of this module lies in its ability to automatically generate SPARQL queries for extracting essential features. This automation is achieved through dynamic traversal of the KG, allowing us to derive the optimal SPARQL query structure based on the underlying graph's rich semantic relationships. By leveraging URI information, particularly relation patterns, we can derive meaningful projection variables in the SPARQL query. This strategic approach ensures that the resulting feature matrices are not only informative but also interpretable, understandable, and explainable. These interpretable feature matrices can be seamlessly integrated into traditional downstream machine learning pipelines, including anomaly detection tasks, enhancing the overall transparency and trustworthiness of the ML process. The evaluation of our approach involved testing on a multi-node cluster with extensive KG data. The results were promising, showcasing the efficacy of literal-based features in predicting and improving ML pipelines. This success is particularly noteworthy in scenarios where KGs lack pre-defined ontologies or when users are relatively new to the domain. A significant advantage of our approach is its ability to streamline the process of acquiring ML-relevant data from source RDF-KGs. By automating the process of generating SPARQL queries that extract features, we make it easier for users and reduce the risk of errors that can occur when creating queries manually. This not only saves time and effort but also enhances the overall data retrieval and utilization process, leading to more efficient and reliable ML outcomes. Moreover, the interpretable nature of the feature matrices empowers users to gain deeper insights into the data and the anomaly detection process. This level of transparency boosts confidence in the results and makes it easier to make informed decisions when dealing with anomalies detected in knowledge graphs.

Our second main research focus was aimed at finding efficient ways to apply anomaly detection techniques to large-scale knowledge graphs in a distributed manner. This effort aimed to answer the following research question:

RQ2: How can we apply anomaly detection on knowledge graphs in a scalable and distributed manner?

To address this question comprehensively, our focus was on anomalies present within literals, and we devised a novel, distributed, and scalable software framework called DistAD (detailed in Chapter 5). This framework is designed to automatically detect anomalies within knowledge graphs by extracting semantic features from RDF data, clustering entities, and then applying anomaly detection algorithms at different levels: *numeric objects*, *predicates*, and *multi-feature* scenarios. The DistAD framework provides high flexibility throughout the entire workflow, allowing end-users to customize the anomaly detection process according to their specific needs and use cases. By providing the ability to select various approaches and granularities, users can customize the detection process to suit their unique KG structures and anomaly detection requirements. With the outcomes of our research question RQ2, we are well-equipped to tackle the challenge we defined earlier (cf. Section 1.2.2). By leveraging DistAD's capabilities, we are able to efficiently and effectively detect anomalies within numeric literals, which significantly contributes to improving the overall quality and reliability of KG data. The extraction of semantic features from RDF data proves to be a pivotal step in the anomaly detection process, as it enables us to gain valuable insights into the underlying data distribution and relationships. The subsequent clustering of entities further refines the detection process by grouping similar entities, facilitating more accurate anomaly detection. Through DistAD's distributed and scalable nature, we can handle KGs of varying sizes and complexities, making it suitable for real-world applications with large-scale datasets. This scalability ensures that the anomaly detection process remains efficient even when dealing with massive KGs. The ability to apply anomaly detection algorithms on different levels, such as numeric objects and predicates, allows us to cast a wide net in detecting anomalies across various aspects of the KG. Additionally, addressing multi-feature scenarios enables the detection of anomalies that span multiple interconnected features, providing a more comprehensive understanding of potential irregularities within the KG.

The third contribution of our thesis tackled the third challenge, which was about efficiently and effectively implementing explainable anomaly detection on large knowledge graphs. In doing so, we aimed to answer the following research question:

RQ3: Can explainable anomaly detection be performed efficiently and effectively on knowledge graphs?

To address this research question comprehensively, we introduced a robust, generic, distributed, and scalable software framework known as ExPAD (elaborated in Chapter 6). ExPAD not only possesses the capability to automatically detect numeric anomalies within knowledge graphs but also provides human-readable explanations for why a specific value of a variable in an observation is deemed an outlier. This interpretability aspect of ExPAD adds a significant advantage as it aids users in understanding the reasons behind the anomalies detected. The core methodology employed by ExPAD involves evaluating and following distributed supervised decision tree splits on variables. By leveraging this approach, ExPAD becomes proficient in detecting and explaining anomalous cases that would have remained unnoticed if individual features were considered in isolation. Analyzing the relationships between different features is essential because anomalies can often appear as intricate interactions among various features. This requires a comprehensive and multi-dimensional approach to ensure accurate detection. The results of our research question RQ3 offer profound insights and contribute to addressing the defined challenge we identified earlier (cf. Section 1.2.3). The ability of ExPAD to automatically detect numeric anomalies in KGs, coupled with its capacity to provide human-readable explanations, significantly enhances the anomaly detection process and supports decision-making processes for users. The interpretability of the explanations generated by ExPAD ensures that users can trust and comprehend the detected anomalies. This transparency is critical in various applications, especially when dealing with sensitive or high-stakes data where understanding the reasoning behind anomaly detection is of paramount importance. Moreover, ExPAD's distributed and scalable nature empowers it to handle KGs of varying sizes and complexities. By efficiently processing large-scale datasets, ExPAD remains practical and applicable in real-world scenarios where KGs can be extensive and constantly evolving. Using decision trees for anomaly detection empowers ExPAD to effectively navigate and explore the structure of the knowledge graph. This helps identify patterns and correlations that could indicate abnormal behavior. This data-driven approach boosts the reliability of anomaly detection, reinforcing ExPAD's status as a valuable tool for KG-based anomaly detection.

Lastly, our thesis tackled the final challenge, which was about having a user-friendly environment to run our proposed frameworks (cf. Section 1.2.4). Since SANSA utilizes Apache Spark and Apache Jena to offer an open-source framework for comprehensive data analytics on large-scale RDF knowledge graphs, a notable hurdle in working with the SANSA stack is the establishment of Hadoop and Spark cluster computing. This task can be technically complex, demanding access to appropriate hardware, and expertise in the installation and integration of essential components. In this regard, we introduced a micro-service architecture that combines a SANSA-enabled Spark and Hadoop cluster, offering two user-friendly ways to interact with it: REST API and Zeppelin Notebook. This architecture relied on Docker technologies. Additionally, we provided a straightforward tutorial that allows even non-technical users to utilize SANSA without requiring any specialized knowledge or skills. Moreover, to reduce the burden of on-premise deployment, we utilized Databricks as a Platform as a Service (PaaS) and showed how users who want to build large-scale RDF Knowledge graph data analytic pipelines can use SANSA with minimal technical requirements and entry hurdles.

8.2 Future Work

Alongside the current research, advancements, and responses to the research questions, in this section, we turn our attention to presenting future directions for future exploration.

- **KG's Feature Extraction:** In Chapter 4, we introduced a novel approach to extracting explainable features from knowledge graphs through deep traversing. Our experimental results demonstrated that the features we extracted possess significant meaning and are easy to interpret. However, we focused solely on considering literals as features in our study. As a promising future direction, other features such as the number of specific predicated (e.g. `foaf:knows` which can count the number of friends of each person) or existence of a type relation (e.g. a boolean value which is `True`, if a specific type relation exists such as `rdf:type dbo:Person`), can be also retrieved from an RDF graph. Moreover, it is possible to explore graph topological features such as the number of neighbors.

  Furthermore, our evaluation of the extracted features was limited to clustering and classification scenarios. To establish the broader utility and effectiveness of these features, another avenue of research could involve testing them in various other machine learning scenarios.

  An important observation is that certain literal values, such as abstracts of books or papers, may consist of a large number of characters. This can lead to an increase in the size of the data frame, thereby impacting the performance of data frame joining operations. To address this issue, a potential solution would be to employ a hashing mechanism or embeddings to transform these large strings into more compact features. This approach could result in improved performance when joining data frames, contributing to more efficient processing and analysis.

- **Anomaly Detection over KGs:** In Chapter 5, we introduced DistAD, a scalable framework designed for applying anomaly detection on knowledge graphs. Within this framework, we primarily utilized statistical and univariate algorithms to identify anomalies (except Isolation Forest). However, in many real-world scenarios, there exist correlations between the features within the dataset. As a result, exploring the use of more multi-variate algorithms, such as OnClass-SVM, presents itself as an intriguing avenue for future research. These algorithms could better capture the interdependencies among features, leading to more accurate anomaly detection. Moreover, as KGs are heterogeneous complex graphs with labeled edges (predicates), applying graph-based anomaly detection techniques [155] to KGs could be an interesting research area.

  Additionally, an exciting prospect for further development lies in the creation of inductive rules to automatically generate SPARQL queries for detecting anomalies. By devising such rules, anomalies can be detected by simply executing the SPARQL queries over the KG. This approach could streamline the anomaly detection process and reduce the manual effort required to identify and address potential anomalies within the KG.

  Moreover, our focus was only on numeric literals. However, literals may have other data types such as categories, dates, strings, etc. Therefore having a unified framework that can detect anomalies within different data types could be a promising future research area.

One of the main reasons why anomaly detection over KGs has not gained adequate attention is the lack of benchmarks and ground truths. To the best of our knowledge, there is no publicly available labeled dataset for anomaly detection in the RDF format. Introducing such a public benchmark can boost this research field.

In addition, a promising avenue for future exploration lies in the synergistic connection between anomaly detection and fact verification within knowledge graphs. Anomaly detection, as addressed in this thesis, focuses on identifying irregular data within the KG. Concurrently, fact verification leverages Language Model Models (LLMs), web data, or other KGs as corpora to substantiate the accuracy of facts. Combining these two approaches could enhance the robustness of anomaly detection by validating detected anomalies through fact verification mechanisms. While anomaly detection pinpoints deviations from expected patterns, fact verification provides an additional layer of validation by cross-referencing against external knowledge sources. This integration could lead to a more comprehensive anomaly detection framework, incorporating not only the identification of anomalies but also the verification of their accuracy through external evidence. The synergy between these methodologies has the potential to elevate the reliability and interpretability of anomaly detection results, offering a more nuanced understanding of irregularities within KGs.

- **Explainable Anomaly Detection over KGs:** In Chapter 6, we presented ExPAD, a distributed and scalable framework specifically designed for explainable anomaly detection. In our approach, we leveraged interpretable models, specifically decision trees, to produce human-readable explanations for detected anomalies. While this method has proven effective, there are other avenues worth exploring in the realm of explanation approaches, such as model-agnostic methods. Embracing model-agnostic techniques could potentially yield more diverse and comprehensive explanations, appealing to a broader range of users and use cases.

  It is important to acknowledge that ExPAD does have a limitation in its current form, primarily stemming from its reliance on univariate data analysis. As a consequence, the framework may not be able to detect all multi-dimensional outliers, unlike alternative methods such as Isolation Forest, which can simultaneously consider multiple variables. However, one of ExPAD's noteworthy strengths lies in its ability to generate meaningful explanations for features that exhibit dependencies on one another. To build upon this foundation, future work could focus on enhancing ExPAD to provide explanations even in scenarios where features lack correlations with each other.

8.3 Closing Remarks

The widespread availability of Semantic Linked Data sources and RDF knowledge graphs has become an invaluable asset for AI, data analytics, and machine learning processes. However, these RDF graphs can occasionally contain incorrect data, which makes the quality of knowledge graphs crucial for the success of subsequent approaches. In this thesis, we have tackled this challenge by creating scalable frameworks that employ anomaly detection techniques to spot these inaccuracies.

Our research has resulted in three major contributions. First, we developed an automated feature extraction process that generates SPARQL queries, simplifying the identification of relevant features from knowledge graphs. This automation significantly eases the anomaly detection process and improves its efficiency.

Secondly, we introduced a distributed anomaly detection framework tailored specifically for KGs. By leveraging distributed computing capabilities, we have optimized the detection of anomalies in large-scale KGs. This approach not only improves the detection accuracy but also significantly reduces the processing time, making it practical for real-world applications.

The third and equally vital contribution is our creation of an explainable anomaly detection method specifically designed for knowledge graphs. The interpretability of our approach enables users to grasp the detected anomalies and comprehend the logic behind them. Through the use of interpretable models, like decision trees, we guarantee that the explanations given are easy for humans to read and insightful, which ultimately improves the overall trustworthiness of the anomaly detection process.

Importantly, these contributions have been integrated into the open-source SANSA project, contributing to its growth and making these advancements accessible to the broader Semantic Web community. Furthermore, the SANSA stack's successful application in various European projects highlights its positive influence on the field of Semantic Web research.

Our work doesn't just offer technical solutions; it also explores broader theoretical aspects, which could serve as a basis for new developments in knowledge graph-based machine learning. By blending technical expertise with theoretical insights, our research has the potential to lead to substantial progress in the field and open up exciting possibilities for future KG-based anomaly detection applications.

APPENDIX A

SANSA REST docker-compose.yml File

We introduced a microservice architecture that incorporates all the necessary distributed computing components to execute SANSA (cf. Chapter 7). To achieve this, we utilized Docker technology and used Docker Swarm. Subsequently, the content of the `docker-compose.yml` file is presented to facilitate reproducibility.

```
version: '3.9'

services:
  spark:
    container_name: spark
    image: fmoghaddam/sansaspark:v1
    environment:
      - SPARK_MODE=master
      - SPARK_DEPLOY_MODE=cluter
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    volumes:
      - ./examples/jars/sansa-examples-spark.jar:
      /opt/bitnami/spark/jars/sansa.jar
    ports:
      - '8080:8080'
      - '7077:7077'
      - '4040:4040'
    networks:
      - spark-net
  spark-worker-1:
    container_name: spark-worker-1
    image: fmoghaddam/sansaspark:v1
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark:7077
      - SPARK_WORKER_MEMORY=1G
      - SPARK_WORKER_CORES=1
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    volumes:
      - ./examples/jars/sansa-examples-spark.jar:
        /opt/bitnami/spark/jars/sansa.jar
    networks:
      - spark-net
  spark-worker-2:
    container_name: spark-worker-2
    image: fmoghaddam/sansaspark:v1
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark:7077
      - SPARK_WORKER_MEMORY=1G
      - SPARK_WORKER_CORES=1
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    volumes:
      - ./examples/jars/sansa-examples-spark.jar:
         /opt/bitnami/spark/jars/sansa.jar
    networks:
      - spark-net
  zeppelin:
    container_name: zeppelin
    image: fmoghaddam/sansazeppelin:v1
    ports:
      - 80:8080
    volumes:
      - ./examples:/opt/sansa-examples
      - ./notebook:/opt/zeppelin/notebook/SANSA
    environment:
      CORE_CONF_fs_defaultFS: "hdfs://namenode:8020"
      SPARK_MASTER: "spark://spark:7077"
      MASTER: "spark://spark:7077"
      SPARK_SUBMIT_OPTIONS: "--jars /opt/sansa-examples/jars/
      sansa-examples-spark.jar --conf spark.serializer=org.apache.spark.
      serializer.KryoSerializer --conf spark.kryo.registrator=org.
      datasyslab.geospark.serde.GeoSparkKryoRegistrator --conf
      spark.kryo.registrator=net.sansa_stack.owl.spark.dataset.
      UnmodifiableCollectionKryoRegistrator"
      SPARK_HOME: "/opt/zeppelin/spark"
      ZEPPELIN_NOTEBOOK_NEW_FORMAT_CONVERT: "true"
    depends_on:
      - spark
      - namenode
      - datanode
      - hue
      - spark-worker-1
      - spark-worker-2
    networks:
      - spark-net
  hue:
    container_name: hue
    image: fmoghaddam/sansahue:v1
    ports:
      - 8088:8088
    environment:
      - NAMENODE_HOST=namenode
      - SPARK_MASTER=spark://spark:7077
    depends_on:
      - namenode
    networks:
      - spark-net
  namenode:
    image: fmoghaddam/sansanamenode:v1
    container_name: namenode
    ports:
     - 8020:8020
    environment:
      - CLUSTER_NAME=test
      - CORE_CONF_fs_defaultFS=hdfs://namenode:8020
      - CORE_CONF_hadoop_http_staticuser_user=root
      - CORE_CONF_hadoop_proxyuser_hue_hosts=*
      - CORE_CONF_hadoop_proxyuser_hue_groups=*
      - HDFS_CONF_dfs_webhdfs_enabled=true
      - HDFS_CONF_dfs_permissions_enabled=false
    healthcheck:
      interval: 5s
      retries: 100
      start_period: 10s
    volumes:
      - ./data/namenode:/hadoop/dfs/name
    networks:
      - spark-net
  datanode:
    image: fmoghaddam/sansadatanode:v1
    container_name: datanode
    volumes:
      - ./data/datanode:/hadoop/dfs/data
    environment:
      - CORE_CONF_fs_defaultFS=hdfs://namenode:8020
    depends_on:
      - namenode
    healthcheck:
      interval: 5s
      retries: 100
      start_period: 10s
    networks:
      - spark-net


networks:
  spark-net:
    external: true
    name: spark-net

```

APPENDIX B

List of Publications

- Conference Papers:
  - **Farshad Bakhshandegan Moghaddam**, Carsten Felix Draschner, Jens Lehmann and Hajira Jabeen, "Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor", SEMANTICS, 2021, IOS Press, pp. 74–88. https://doi.org/10.3233/SSW210036.
  - Carsten Felix Draschner, Claus Stadler, **Farshad Bakhshandegan Moghaddam**, Jens Lehmann, Hajira Jabeen, "DistRDF2ML - Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs", Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM). Association for Computing Machinery, New York, NY, USA, 4465–4474. https://doi.org/10.1145/3459637.3481999.
  - **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs", IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 243-250, https://doi.org/10.1109/ICSC52841.2022.00047.
  - **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs", IEEE 17th International Conference on Semantic Computing (ICSC), 2023, pp. 204-211, https://doi.org/10.1109/ICSC56153.2023.00040.
  - **Farshad Bakhshandegan Moghaddam**, Jens Lehmann and Hajira Jabeen, "Anomaly Detection for Numerical Literals in Knowledge Graphs: A Short Review of Approaches", The Sixth IEEE International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2023, pp. 46-53, https://doi.org/10.1109/AIKE59827.2023.00015.
- Workshops, Demos, and Doctoral Consortium:
  - **Farshad Bakhshandegan Moghaddam**, Carsten Felix Draschner, Jens Lehmann, Hajira Jabeen, "Semantic Web Analysis with Flavor of Micro-Services", LAMBDA Doctoral Workshop 2021, http://ceur-ws.org/Vol-3195/paper1.pdf

- Carsten Felix Draschner, **Farshad Bakhshandegan Moghaddam**, Jens Lehmann, Hajira Jabeen, "Semantic Analytics in the Palm of Your Browser", LAMBDA Doctoral Workshop 2021, `http://ceur-ws.org/Vol-3195/paper2.pdf`

Bibliography

[1] T. Berners-Lee, "A roadmap to the Semantic Web", 1998 (cit. on pp. 1, 13, 39).

[2] C. Bizer, M.-E. Vidal and H. Skaf-Molli, "Linked Open Data", *Encyclopedia of Database Systems*, Springer New York, 2018 2096, ISBN: 978-1-4614-8265-9 (cit. on pp. 2, 39).

[3] C. Xiong, R. Power and J. Callan, "Explicit Semantic Ranking for Academic Search via Knowledge Graph Embedding", *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, ed. by R. Barrett, R. Cummings, E. Agichtein and E. Gabrilovich, ACM, 2017 1271, URL: https://doi.org/10.1145/3038912.3052558 (cit. on p. 2).

[4] J. R. Pérez-Agüera, J. Arroyo, J. Greenberg, J. Pérez-Iglesias and V. Fresno, "Using BM25F for semantic search", *Proceedings of the 3rd International Semantic Search Workshop, SEMSEARCH '10, Raleigh, North Carolina, USA, April 26, 2010*, ed. by M. Grobelnik, P. Mika, D. T. Tran and H. Wang, ACM, 2010 2:1, URL: https://doi.org/10.1145/1863879.1863881 (cit. on p. 2).

[5] S. Ji, S. Pan, E. Cambria, P. Marttinen and P. S. Yu, *A Survey on Knowledge Graphs: Representation, Acquisition and Applications*, CoRR **abs/2002.00388** (2020), arXiv: 2002.00388, URL: https://arxiv.org/abs/2002.00388 (cit. on p. 2).

[6] W. Zhang et al., "XTransE: Explainable Knowledge Graph Embedding for Link Prediction with Lifestyles in e-Commerce", *Semantic Technology - 9th Joint International Conference, JIST 2019, Hangzhou, China, November 25-27, 2019, Revised Selected Papers*, ed. by X. Wang, F. A. Lisi, G. Xiao and E. Botoeva, vol. 1157, Communications in Computer and Information Science, Springer, 2019 78, URL: https://doi.org/10.1007/978-981-15-3412-6%5C_8 (cit. on p. 2).

[7] F. Li et al., *AliMe KG: Domain Knowledge Graph Construction and Application in E-commerce*, CoRR **abs/2009.11684** (2020), arXiv: 2009.11684, URL: https://arxiv.org/abs/2009.11684 (cit. on p. 2).

[8] X. Xiang, Z. Wang, Y. Jia and B. Fang, "Knowledge Graph-Based Clinical Decision Support System Reasoning: A Survey", *Fourth IEEE International Conference on Data Science in Cyberspace, DSC 2019, Hangzhou, China, June 23-25, 2019*, IEEE, 2019 373, URL: https://doi.org/10.1109/DSC.2019.00063 (cit. on p. 2).

[9] R. Lourdusamy and X. J. Mattam, "Resource description framework based semantic knowledge graph for clinical decision support systems", *Web Semantics*, Elsevier, 2021 69 (cit. on p. 2).

[10] J. M. Juran, *Juran's Quality Control Handbook*, 4th, Mcgraw-Hill (Tx), 1974, ISBN: 0070331766 (cit. on p. 2).

[11] D. Vrandecic, "Wikidata: a new platform for collaborative data collection", *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, ed. by A. Mille, F. Gandon, J. Misselis, M. Rabinovich and S. Staab, ACM, 2012 1063 (cit. on p. 2).

[12] K. Bollacker, C. Evans, P. Paritosh, T. Sturge and J. Taylor, "Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge", *ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, Vancouver, Canada: Association for Computing Machinery, 2008 1247, ISBN: 9781605581026 (cit. on p. 2).

[13] T. Mitchell et al., "Never-Ending Learning", *AAAI-15*, 2015 (cit. on p. 2).

[14] S. Auer et al., "DBpedia: A Nucleus for a Web of Open Data", *International Semantic Web Conference ISWC*, ed. by K. Aberer et al., Lecture Notes in Computer Science, Springer, 2007 722 (cit. on p. 2).

[15] F. M. Suchanek, G. Kasneci and G. Weikum, "Yago: a core of semantic knowledge", *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, ed. by C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider and P. J. Shenoy, ACM, 2007 697 (cit. on p. 2).

[16] V. Chandola, A. Banerjee and V. Kumar, *Anomaly Detection: A Survey*, ACM Comput. Surv. (2009), ISSN: 0360-0300 (cit. on pp. 2, 20, 21, 55).

[17] C. C. Aggarwal, *Outlier Ensembles: Position Paper*, SIGKDD Explor. Newsl. (2013) 49, ISSN: 1931-0145 (cit. on pp. 3, 21, 32).

[18] M. Zaharia et al., *Apache Spark: A Unified Engine for Big Data Processing*, Commun. ACM **59** (2016) 56, ISSN: 0001-0782, URL: https://doi.org/10.1145/2934664 (cit. on p. 3).

[19] J. Lehmann et al., "Distributed Semantic Analytics using the SANSA Stack", *Proceedings of 16th International Semantic Web Conference - Resources Track (ISWC'2017)*, Springer, 2017 147 (cit. on pp. 3, 7, 40, 56, 68, 87).

[20] *A. J. Foundation*, *Apache Jena*, 2023, URL: https://jena.apache.org/index.html (cit. on p. 3).

[21] R. Benjamins, J. Contreras, Ó. Corcho and A. Gómez-Pérez, "Six challenges for the Semantic Web", *International Conference on Principles of Knowledge Representation and Reasoning*, 2002 (cit. on p. 3).

[22] Databricks-Inc, *Databricks platform*, 2021, URL: https://databricks.com/product/ (cit. on pp. 6, 87).

[23] F. B. Moghaddam, C. F. Draschner, J. Lehmann and H. Jabeen, "Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor", *SEMANTICS, The Netherlands*, 2021 (cit. on pp. 6, 27, 29, 58, 60, 63, 70, 74, 78, 86, 87).

[24] F. B. Moghaddam, J. Lehmann and H. Jabeen, "DistAD: A Distributed Generic Anomaly Detection Framework over Large KGs", *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, 2022 243 (cit. on pp. 7, 27, 33, 86, 87).

[25] F. B. Moghaddam, J. Lehmann and H. Jabeen, "ExPAD: An Explainable Distributed Automatic Anomaly Detection Framework over Large KGs", *2023 IEEE 17th International Conference on Semantic Computing (ICSC)*, 2023 (cit. on pp. 7, 27, 33, 86, 87).

[26] F. B. Moghaddam and C. D. Jens, "Semantic Web Analysis with Flavor of Micro-Services", *Big Data Analytics 3rd Summer School*, 2021 (cit. on p. 7).

[27] C. Draschner, F. B. Moghaddam, J. Lehmann and H. Jabeen, "Semantic Analytics in the Palm of your Browser", *Big Data Analytics 3rd Summer School*, 2021 (cit. on p. 7).

[28] F. B. Moghaddam, J. Lehmann and H. Jabeen, "Anomaly Detection for Numerical Literals in Knowledge Graphs: A Short Review of Approaches", *2023 IEEE Sixth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, Los Alamitos, CA, USA: IEEE Computer Society, 2023 46 (cit. on pp. 7, 27).

[29] M. Nickel, K. Murphy, V. Tresp and E. Gabrilovich, *A Review of Relational Machine Learning for Knowledge Graphs*, Proc. IEEE **104** (2016) 11, URL: https://doi.org/10.1109/JPROC.2015.2483592 (cit. on pp. 11, 40).

[30] A. Hogan et al., *Knowledge graphs*, ACM Computing Surveys (Csur) **54** (2021) 1 (cit. on p. 11).

[31] L. Bellomarini, D. Fakhoury, G. Gottlob and E. Sallinger, "Knowledge Graphs and Enterprise AI: The Promise of an Enabling Technology", *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019 26 (cit. on p. 12).

[32] P. Cimiano and H. Paulheim, *Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods*, Semant. Web (2017) 489, ISSN: 1570-0844 (cit. on p. 12).

[33] L. Ehrlinger and W. Wöß, "Towards a Definition of Knowledge Graphs.", *SEMANTiCS (Posters, Demos, SuCCESS)*, 2016 (cit. on p. 12).

[34] M. Färber, F. Bartscherer, C. Menne and A. Rettinger, *Linked Data Quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO*, Semantic Web Journal (2017) 1 (cit. on p. 12).

[35] J. Pérez, M. Arenas and C. Gutierrez, *Semantics and Complexity of SPARQL*, ACM Trans. Database Syst. (2009), ISSN: 0362-5915 (cit. on p. 14).

[36] T. White, *Hadoop: The Definitive Guide*, Fourth, O'Reilly, 2015, ISBN: 9781491901632 (cit. on p. 17).

[37] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System", *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010 1 (cit. on p. 17).

[38] X. Meng et al., *MLlib: Machine Learning in Apache Spark*, J. Mach. Learn. Res. (2016) 1235, ISSN: 1532-4435 (cit. on p. 17).

[39] M. Odersky and al., *An Overview of the Scala Programming Language*, tech. rep., EPFL Lausanne, Switzerland, 2004 (cit. on p. 18).

[40] A. Zaveri et al., *Quality assessment methodologies for linked open data*, Submitted to Semantic Web Journal (2013) 1 (cit. on p. 19).

[41] B. Stvilia, L. Gasser, M. B. Twidale and L. C. Smith, *A framework for information quality assessment*, J. Assoc. Inf. Sci. Technol. (2007) 1720 (cit. on p. 19).

[42] A. Zaveri et al., *Quality assessment for Linked Data: A Survey*, Semantic Web (2016) 63 (cit. on pp. 19, 20).

[43] M. Färber, F. Bartscherer, C. Menne and A. Rettinger, *Linked data quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO*, Semantic Web (2018) 77 (cit. on p. 20).

[44] V. Jayawardene, S. Sadiq and M. Indulska, *An Analysis of Data Quality Dimensions*, ITEE Tech (2015) (cit. on p. 20).

[45] R. Y. Wang and D. M. Strong, *Beyond Accuracy: What Data Quality Means to Data Consumers*, J. Manag. Inf. Syst. (1996) 5 (cit. on p. 20).

[46] N. Mihindukulasooriya, R. García-Castro and A. Gómez-Pérez, "LD sniffer: A quality assessment tool for measuring the accessibility of linked data", *Knowledge Engineering and Knowledge Management: EKAW 2016 Satellite Events, EKM and Drift-an-LOD, Bologna, Italy, November 19–23, 2016, Revised Selected Papers*, Springer, 2017 149 (cit. on p. 20).

[47] D. Hawkins, *Identification of Outliers*, English, Chapman and Hall, 1980 (cit. on p. 21).

[48] A. Charu C., *Outlier Analysis.* Springer, 2013, ISBN: 9781461463955 (cit. on p. 21).

[49] G. Pang, C. Shen, L. Cao and A. V. D. Hengel, *Deep Learning for Anomaly Detection: A Review*, ACM Comput. Surv. (2021), ISSN: 0360-0300 (cit. on p. 21).

[50] S. Ramaswamy, R. Rastogi and K. Shim,
"Efficient Algorithms for Mining Outliers from Large Data Sets.", *SIGMOD Conference*,
ACM, 2000 427, ISBN: 1-58113-217-4 (cit. on pp. 21, 23).

[51] F. Angiulli and C. Pizzuti, "Fast Outlier Detection in High Dimensional Spaces.", *PKDD*,
Lecture Notes in Computer Science, Springer, 2002 15 (cit. on pp. 21, 23).

[52] M. M. Breunig, H.-P. Kriegel, R. T. Ng and J. Sander,
"LOF: Identifying Density-Based Local Outliers.", *SIGMOD Conference*, ACM, 2000 93,
ISBN: 1-58113-217-4 (cit. on pp. 21, 23).

[53] C. C. Aggarwal and P. S. Yu, *Outlier detection for high dimensional data*,
SIGMOD Rec. (2 2001) 37 (cit. on p. 21).

[54] A. Zimek, E. Schubert and H.-P. Kriegel,
A survey on unsupervised outlier detection in high-dimensional numerical data.,
Stat. Anal. Data Min. (2012) 363 (cit. on p. 21).

[55] J. Zhang et al., *A concept lattice based outlier mining method in low-dimensional subspaces.*,
Pattern Recognit. Lett. (2009) 1434 (cit. on p. 22).

[56] J. K. Dutta, B. Banerjee and C. K. Reddy,
RODS: Rarity based Outlier Detection in a Sparse Coding Framework.,
IEEE Trans. Knowl. Data Eng. (2016) 483 (cit. on p. 22).

[57] J. Zhang, S. Zhang, K. H. Chang and X. Qin,
An outlier mining algorithm based on constrained concept lattice.,
Int. J. Systems Science (2014) 1170 (cit. on p. 22).

[58] C. C. Aggarwal and P. S. Yu,
An effective and efficient algorithm for high-dimensional outlier detection.,
VLDB J. (2005) 211 (cit. on p. 22).

[59] A. Lazarevic and V. Kumar, "Feature bagging for outlier detection.", *KDD*, ACM, 2005 157
(cit. on p. 22).

[60] R. McGill, J. W. Tukey and W. A. Larsen, *Variations of Box Plots*,
The American Statistician (1978) 12, ISSN: 00031305 (cit. on pp. 22, 34, 58, 74).

[61] C. Leys, C. Ley, O. Klein, P. Bernard and L. Licata, *Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median*,
Journal of Experimental Social Psychology (2013) 764, ISSN: 0022-1031
(cit. on pp. 22, 58, 74).

[62] E. Parzen, *On Estimation of a Probability Density Function and Mode*, English,
The Annals of Mathematical Statistics (1962) pp. 1065, ISSN: 00034851 (cit. on pp. 23, 34).

[63] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola and R. C. Williamson,
Estimating the Support of a High-Dimensional Distribution, Neural Computation (2001) 1443
(cit. on p. 24).

[64] T. Zemicheal and T. G. Dietterich,
"Anomaly Detection in the Presence of Missing Values for Weather Data Quality Control",
Proceedings of the 2nd ACM SIGCAS Conference on Computing and Sustainable Societies,
Accra, Ghana, 2019, ISBN: 9781450367141 (cit. on pp. 24, 36, 58, 61).

[65] X. Meng et al., *MLlib: Machine Learning in Apache Spark*, Journal of Machine Learning Research **17** (2016) 1, URL: http://jmlr.org/papers/v17/15-237.html (cit. on p. 28).

[66] E. Alpaydin, *Introduction to Machine Learning*, 2nd, The MIT Press, 2010, ISBN: 026201243X (cit. on p. 28).

[67] V. N. P. Kappara, R. Ichise and O. P. Vyas, "LiDDM: A Data Mining System for Linked Data", *WWW2011 Workshop on Linked Data on the Web, Hyderabad, India, March 29, 2011*, CEUR Workshop Proceedings, CEUR-WS.org, 2011 (cit. on p. 28).

[68] W. Cheng, G. Kasneci, T. Graepel, D. H. Stern and R. Herbrich, "Automated feature generation from structured knowledge", *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, ACM, 2011 1395 (cit. on p. 28).

[69] M. Khan, G. Grimnes and A. Dengel, "Two pre-processing operators for improved learning from semanticweb data", *First RapidMiner Community Meeting And Conference (RCOMM 2010)*, 2010 (cit. on p. 28).

[70] P. Ristoski, C. Bizer and H. Paulheim, *Mining the Web of Linked Data with RapidMiner*, J. Web Semant. (2015) 142 (cit. on p. 28).

[71] A. Grover and J. Leskovec, "Node2vec: Scalable Feature Learning for Networks", *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016 855, ISBN: 9781450342322, URL: https://doi.org/10.1145/2939672.2939754 (cit. on pp. 28, 29).

[72] B. Perozzi, R. Al-Rfou and S. Skiena, "DeepWalk: Online Learning of Social Representations", *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, New York, New York, USA: Association for Computing Machinery, 2014 701, ISBN: 9781450329569, URL: https://doi.org/10.1145/2623330.2623732 (cit. on pp. 28, 29).

[73] P. Ristoski and H. Paulheim, "RDF2Vec: RDF Graph Embeddings for Data Mining", *The Semantic Web – ISWC 2016*, ed. by P. Groth et al., Cham: Springer International Publishing, 2016 498, ISBN: 978-3-319-46523-4 (cit. on pp. 28, 29).

[74] B. Yang, W. Yih, X. He, J. Gao and L. Deng, "Embedding Entities and Relations for Learning and Inference in Knowledge Bases", *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, ed. by Y. Bengio and Y. LeCun, 2015, URL: http://arxiv.org/abs/1412.6575 (cit. on p. 28).

[75] J. Ugander, B. Karrer, L. Backstrom and C. Marlow, *The Anatomy of the Facebook Social Graph*, CoRR **abs/1111.4503** (2011), arXiv: 1111.4503, URL: http://arxiv.org/abs/1111.4503 (cit. on p. 28).

[76] A. Bordes, N. Usunier, A. García-Durán, J. Weston and O. Yakhnenko, "Translating Embeddings for Modeling Multi-relational Data", *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, ed. by C. J. C. Burges, L. Bottou, Z. Ghahramani and K. Q. Weinberger, 2013 2787, URL: https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html (cit. on p. 28).

[77] J. Mynarz and V. Svátek, "Towards a Benchmark for LOD-Enhanced Knowledge Discovery from Structured Data", *Proceedings of the Second International Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data, Montpellier, France, May 26, 2013*, ed. by J. Völker, H. Paulheim, J. Lehmann, M. Niepert and H. Sack, CEUR Workshop Proceedings, CEUR-WS.org, 2013 41 (cit. on p. 28).

[78] H. Paulheim and J. Fürnkranz, "Unsupervised generation of data mining features from linked open data", *2nd International Conference on Web Intelligence, Mining and Semantics, WIMS '12, Craiova, Romania, June 6-8, 2012*, ACM, 2012 31:1 (cit. on pp. 28, 32, 48, 50).

[79] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn and K. M. Borgwardt, *Weisfeiler-Lehman Graph Kernels*, J. Mach. Learn. Res. **12** (2011) 2539, URL: https://dl.acm.org/doi/10.5555/1953048.2078187 (cit. on p. 29).

[80] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, ed. by Y. Bengio and Y. LeCun, 2013, URL: http://arxiv.org/abs/1301.3781 (cit. on p. 29).

[81] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston and O. Yakhnenko, "Translating Embeddings for Modeling Multi-Relational Data", *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, Lake Tahoe, Nevada: Curran Associates Inc., 2013 2787 (cit. on pp. 29, 48, 50).

[82] S. M. Kazemi and D. Poole, "SimplE Embedding for Link Prediction in Knowledge Graphs", *Advances in Neural Information Processing Systems*, 2018 (cit. on pp. 29, 48, 50).

[83] B. Yang, W. Yih, X. He, J. Gao and L. Deng, "Embedding Entities and Relations for Learning and Inference in Knowledge Bases", *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, ed. by Y. Bengio and Y. LeCun, 2015 (cit. on pp. 29, 48, 50).

[84] Z. Wang, J. Zhang, J. Feng and Z. Chen, "Knowledge Graph Embedding by Translating on Hyperplanes.", *AAAI*, ed. by C. E. Brodley and P. Stone, AAAI Press, 2014 1112, ISBN: 978-1-57735-661-5, URL: http://dblp.uni-trier.de/db/conf/aaai/aaai2014.html#WangZFC14 (cit. on p. 29).

[85] Z. Sun, Z.-H. Deng, J.-Y. Nie and J. Tang, "RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space", *International Conference on Learning Representations*, 2019, URL: https://openreview.net/forum?id=HkgEQnRqYQ (cit. on p. 29).

[86] W. W. Cohen, *TensorLog: A Differentiable Deductive Database*, CoRR (2016), arXiv: 1605.06523 (cit. on p. 30).

[87] W. Y. Wang, K. Mazaitis and W. W. Cohen, "Structure Learning via Parameter Learning.", *CIKM*, ACM, 2014 1199, ISBN: 978-1-4503-2598-1 (cit. on pp. 30, 40).

[88] J. Cohen, *A View of the Origins and Development of Prolog*, Commun. ACM **31** (1988) 26, ISSN: 0001-0782, URL: https://doi.org/10.1145/35043.35045 (cit. on p. 30).

[89] Martín Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015, URL: https://www.tensorflow.org/ (cit. on p. 30).

[90] L. Bühmann, J. Lehmann and P. Westphal, *DL-Learner - A framework for inductive learning on the Semantic Web.*, J. Web Semant. (2016) 15 (cit. on pp. 30, 40).

[91] L. Galárraga, C. Teflioudi, K. Hose and F. M. Suchanek, *Fast rule mining in ontological knowledge bases with AMIE+.*, VLDB J. (2015) 707 (cit. on pp. 30, 40).

[92] L. Galárraga, C. Teflioudi, K. Hose and F. M. Suchanek, *Fast rule mining in ontological knowledge bases with AMIE+*, VLDB J. (2015) 707 (cit. on p. 30).

[93] M. Ali et al., *PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings*, J. Mach. Learn. Res. **22** (2021) 82:1, URL: http://jmlr.org/papers/v22/20-825.html (cit. on p. 31).

[94] X. Han et al., "OpenKE: An Open Toolkit for Knowledge Embedding", *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, ed. by E. Blanco and W. Lu, Association for Computational Linguistics, 2018 139, URL: https://doi.org/10.18653/v1/d18-2024 (cit. on p. 31).

[95] J. J. Dai et al., *BigDL: A Distributed Deep Learning Framework for Big Data*, CoRR **abs/1804.05839** (2018), arXiv: 1804.05839, URL: http://arxiv.org/abs/1804.05839 (cit. on p. 31).

[96] D. Zheng et al., "DGL-KE: Training Knowledge Graph Embeddings at Scale", *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, ed. by J. X. Huang et al., ACM, 2020 739, URL: https://doi.org/10.1145/3397271.3401172 (cit. on p. 31).

[97] Z. Zhu, S. Xu, J. Tang and M. Qu,
"GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding",
The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019,
ed. by L. Liu et al., ACM, 2019 2494,
URL: https://doi.org/10.1145/3308558.3313508 (cit. on p. 31).

[98] G. Malewicz et al., "Pregel: a system for large-scale graph processing",
Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010,
ed. by A. K. Elmagarmid and D. Agrawal, ACM, 2010 135,
URL: https://doi.org/10.1145/1807167.1807184 (cit. on p. 31).

[99] C. Stadler, G. Sejdiu, D. Graux and J. Lehmann, "Sparklify: A Scalable Software Component for Efficient Evaluation of SPARQL Queries over Distributed RDF Datasets",
The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II, Lecture Notes in Computer Science, Springer, 2019 293 (cit. on p. 31).

[100] G. Sejdiu, *Efficient Distributed In-Memory Processing of RDF Datasets*,
PhD thesis: University of Bonn, Germany, 2020,
URL: https://hdl.handle.net/20.500.11811/8735 (cit. on p. 31).

[101] G. Sejdiu, I. Ermilov, J. Lehmann and M. N. Mami,
"DistLODStats: Distributed Computation of RDF Dataset Statistics",
The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II, ed. by D. Vrandecic et al., vol. 11137,
Lecture Notes in Computer Science, Springer, 2018 206,
URL: https://doi.org/10.1007/978-3-030-00668-6%5C_13 (cit. on p. 31).

[102] R. Dadwal, D. Graux, G. Sejdiu, H. Jabeen and J. Lehmann,
"Clustering Pipelines of Large RDF POI Data",
The Semantic Web: ESWC 2019 Satellite Events - ESWC 2019 Satellite Events, Portorož, Slovenia, June 2-6, 2019, Revised Selected Papers, ed. by P. Hitzler et al., vol. 11762,
Lecture Notes in Computer Science, Springer, 2019 24,
URL: https://doi.org/10.1007/978-3-030-32327-1%5C_5 (cit. on p. 31).

[103] D. Wienand and H. Paulheim, "Detecting Incorrect Numerical Data in DBpedia",
The Semantic Web: Trends and Challenges, ed. by V. Presutti et al.,
Springer International Publishing, 2014 504, ISBN: 978-3-319-07443-6 (cit. on pp. 32–34).

[104] D. Fleischhacker, H. Paulheim, V. Bryl, J. Völker and C. Bizer,
"Detecting Errors in Numerical Linked Data Using Cross-Checked Outlier Detection",
The Semantic Web – ISWC 2014, ed. by P. Mika et al., Springer International Publishing, 2014 357, ISBN: 978-3-319-11964-9 (cit. on pp. 32–34).

[105] H. Jabeen, R. Dadwal, G. Sejdiu and J. Lehmann, "Divided We Stand Out! Forging Cohorts fOr Numeric Outlier Detection in Large Scale Knowledge Graphs (CONOD)",
Knowledge Engineering and Knowledge Management - 21st International Conference, EKAW 2018, Nancy, France, November 12-16, 2018, Proceedings,
Lecture Notes in Computer Science, Springer, 2018 534 (cit. on pp. 32–35).

[106] H. Paulheim,
"Identifying Wrong Links between Datasets by Multi-dimensional Outlier Detection.",
WoDOOM, ed. by P. Lambrix, G. Qi, M. Horridge and B. Parsia, vol. 1162,
CEUR Workshop Proceedings, CEUR-WS.org, 2014 27 (cit. on pp. 32, 34).

[107] H. Paulheim,
"Identifying Wrong Links between Datasets by Multi-dimensional Outlier Detection.",
WoDOOM, CEUR Workshop Proceedings, CEUR-WS.org, 2014 27 (cit. on p. 33).

[108] A. P. Dempster, N. M. Laird and D. B. Rubin,
Maximum likelihood from incomplete data via the EM algorithm,
Journal of the Royal Statistical Society, Series B (1977) 1 (cit. on p. 34).

[109] M. Hall et al., *The WEKA data mining software: an update*,
SIGKDD Explor. Newsl. (2009) 10 (cit. on p. 34).

[110] T. Kliegr, *Linked hypernyms: Enriching DBpedia with Targeted Hypernym Discovery*,
Journal of Web Semantics (2015) 59, ISSN: 1570-8268 (cit. on p. 34).

[111] M. Datar, N. Immorlica, P. Indyk and V. S. Mirrokni,
"Locality-Sensitive Hashing Scheme Based on p-Stable Distributions",
Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG '04,
Brooklyn, New York, USA: Association for Computing Machinery, 2004 253,
ISBN: 1581138857 (cit. on pp. 34, 59).

[112] B. Micenková, R. T. Ng, X. Dang and I. Assent,
"Explaining Outliers by Subspace Separability", *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*,
ed. by H. Xiong, G. Karypis, B. Thuraisingham, D. J. Cook and X. Wu,
IEEE Computer Society, 2013 518,
URL: https://doi.org/10.1109/ICDM.2013.132 (cit. on p. 35).

[113] X. Dang, B. Micenková, I. Assent and R. T. Ng, "Local Outlier Detection with Interpretation",
Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III,
ed. by H. Blockeel, K. Kersting, S. Nijssen and F. Zelezný, vol. 8190,
Lecture Notes in Computer Science, Springer, 2013 304,
URL: https://doi.org/10.1007/978-3-642-40994-3%5C_20 (cit. on p. 35).

[114] C. C. Aggarwal, *Outlier Analysis*, 2nd, Springer Publishing Company, Incorporated, 2016,
ISBN: 3319475770 (cit. on p. 35).

[115] X. V. Nguyen et al., *Discovering outlying aspects in large datasets*,
Data Min. Knowl. Discov. **30** (2016) 1520,
URL: https://doi.org/10.1007/s10618-016-0453-2 (cit. on p. 35).

[116] F. Doshi-Velez and B. Kim, *Towards A Rigorous Science of Interpretable Machine Learning*,
2017, arXiv: 1702.08608 [stat.ML] (cit. on p. 35).

[117] A. B. Arrieta et al., *Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI*, Inf. Fusion **58** (2020) 82,
URL: https://doi.org/10.1016/j.inffus.2019.12.012 (cit. on p. 35).

[118] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl and B. Yu, *Definitions, methods, and applications in interpretable machine learning*, Proceedings of the National Academy of Sciences **116** (2019) 22071, eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1900654116, URL: https://www.pnas.org/doi/abs/10.1073/pnas.1900654116 (cit. on p. 35).

[119] H. Salehinejad et al., *Recent Advances in Recurrent Neural Networks*, CoRR **abs/1801.01078** (2018), arXiv: 1801.01078, URL: http://arxiv.org/abs/1801.01078 (cit. on p. 36).

[120] O. Gorokhov, M. Petrovskiy and I. V. Mashechkin, "Convolutional Neural Networks for Unsupervised Anomaly Detection in Text Data", *Intelligent Data Engineering and Automated Learning - IDEAL 2017 - 18th International Conference, Guilin, China, October 30 - November 1, 2017, Proceedings*, ed. by H. Yin et al., vol. 10585, Lecture Notes in Computer Science, Springer, 2017 500, URL: https://doi.org/10.1007/978-3-319-68935-7%5C_54 (cit. on p. 36).

[121] M. T. Ribeiro, S. Singh and C. Guestrin, ""Why should i trust you?" Explaining the predictions of any classifier", *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016 1135 (cit. on pp. 36, 37).

[122] M. T. Ribeiro, S. Singh and C. Guestri, "Anchors: High-Precision Model-Agnostic Explanations", *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, AAAI Press, 2018 1527 (cit. on pp. 36, 37).

[123] S. M. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions*, Advances in neural information processing systems (2017) (cit. on pp. 36, 38).

[124] S. M. Lundberg and S. Lee, "A Unified Approach to Interpreting Model Predictions", *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, ed. by I. Guyon et al., 2017 4765, URL: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html (cit. on p. 36).

[125] M. A. Siddiqui, A. Fern, T. G. Dietterich and W. Wong, *Sequential Feature Explanations for Anomaly Detection*, ACM Trans. Knowl. Discov. Data **13** (2019) 1:1, URL: https://doi.org/10.1145/3230666 (cit. on p. 36).

[126] G. Pang, L. Cao and L. Chen, "Outlier Detection in Complex Categorical Data by Modelling the Feature Value Couplings", *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, New York, New York, USA: AAAI Press, 2016 1902, ISBN: 9781577357704 (cit. on p. 36).

[127] G. Pang, L. Cao, L. Chen and H. Liu, "Unsupervised Feature Selection for Outlier Detection by Modelling Hierarchical Value-Feature Couplings", *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, ed. by F. Bonchi, J. Domingo-Ferrer, R. Baeza-Yates, Z. Zhou and X. Wu, IEEE Computer Society, 2016 410, URL: `https://doi.org/10.1109/ICDM.2016.0052` (cit. on p. 36).

[128] J. He and J. G. Carbonell, "Co-selection of Features and Instances for Unsupervised Rare Category Analysis", *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, SIAM, 2010 525, URL: `https://doi.org/10.1137/1.9781611972801.46` (cit. on p. 36).

[129] K. Noto, C. E. Brodley and D. K. Slonim, *FRaC: a feature-modeling approach for semi-supervised and unsupervised anomaly detection*, Data Min. Knowl. Discov. **25** (2012) 109, URL: `https://doi.org/10.1007/s10618-011-0234-x` (cit. on p. 36).

[130] H. Paulheim and R. Meusel, *A decomposition of the outlier detection problem into a set of supervised learning problems*, Mach. Learn. **100** (2015) 509, URL: `https://doi.org/10.1007/s10994-015-5507-y` (cit. on p. 36).

[131] Q. Yang, J. Singh and J. Lee, "Isolation-based feature selection for unsupervised outlier detection", *Proc. Annu. Conf. Progn. Health Manag. Soc*, vol. 11, 1, 2019 (cit. on p. 36).

[132] Z. He, X. Xu, J. Z. Huang and S. Deng, *FP-outlier: Frequent pattern based outlier detection*, Comput. Sci. Inf. Syst. **2** (2005) 103, URL: `https://doi.org/10.2298/CSIS0501103H` (cit. on p. 37).

[133] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, ed. by J. B. Bocca, M. Jarke and C. Zaniolo, Morgan Kaufmann, 1994 487, URL: `http://www.vldb.org/conf/1994/P487.PDF` (cit. on p. 37).

[134] Y. Zhu, N. M. Nayak and A. K. Roy-Chowdhury, *Context-Aware Activity Recognition and Anomaly Detection in Video*, IEEE J. Sel. Top. Signal Process. **7** (2013) 91, URL: `https://doi.org/10.1109/JSTSP.2012.2234722` (cit. on p. 37).

[135] K. Vaculík and L. Popelínský, "DGRMiner: Anomaly Detection and Explanation in Dynamic Graphs", *Advances in Intelligent Data Analysis XV - 15th International Symposium, IDA 2016, Stockholm, Sweden, October 13-15, 2016, Proceedings*, ed. by H. Boström, A. J. Knobbe, C. Soares and P. Papapetrou, vol. 9897, Lecture Notes in Computer Science, 2016 308, URL: `https://doi.org/10.1007/978-3-319-46349-0%5C_27` (cit. on p. 37).

[136] I. B. Kraiem, F. Ghozzi, A. Péninou, G. Roman-Jimenez and O. Teste, "Human-Interpretable Rules for Anomaly Detection in Time-Series", *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, ed. by Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis and F. Guerra, OpenProceedings.org, 2021 457, URL: `https://doi.org/10.5441/002/edbt.2021.51` (cit. on p. 37).

[137] D. Cortes, *Explainable outlier detection through decision tree conditioning*, CoRR **abs/2001.00636** (2020), arXiv: `2001.00636`, URL: `http://arxiv.org/abs/2001.00636` (cit. on pp. 37, 68, 77).

[138] D. L. Aguilar, M. A. Medina-Pérez, O. Loyola-González, K. R. Choo and E. Bucheli-Susarrey, *Towards an Interpretable Autoencoder: A Decision-Tree-Based Autoencoder and its Application in Anomaly Detection*, IEEE Trans. Dependable Secur. Comput. **20** (2023) 1048, URL: `https://doi.org/10.1109/TDSC.2022.3148331` (cit. on p. 37).

[139] L. Bertossi, J. Li, M. Schleich, D. Suciu and Z. Vagena, "Causality-based explanation of classification outcomes", *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, 2020 1 (cit. on p. 38).

[140] A. Kristiadi, M. A. Khan, D. Lukovnikov, J. Lehmann and A. Fischer, *Incorporating literals into knowledge graph embeddings*, arXiv preprint arXiv:1802.00934 (2018) (cit. on p. 40).

[141] C. Stadler, G. Sejdiu, D. Graux and J. Lehmann, "Sparklify: A Scalable Software Component for Efficient Evaluation of SPARQL Queries over Distributed RDF Datasets", *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, ed. by C. Ghidini et al., Lecture Notes in Computer Science, Springer, 2019 293 (cit. on pp. 42, 93).

[142] E. Moore, *The Shortest Path Through a Maze*, Bell Telephone System. Technical publications. monograph, Bell Telephone System., 1959 (cit. on p. 42).

[143] F. Xia et al., *Random Walks: A Review of Algorithms and Applications*, IEEE Transactions on Emerging Topics in Computational Intelligence (2020) 95 (cit. on p. 42).

[144] P. Westphal, L. Bühmann, S. Bin, H. Jabeen and J. Lehmann, *SML-Bench - A benchmarking framework for structured machine learning*, Semantic Web (2019) 231 (cit. on p. 48).

[145] P. Ristoski and H. Paulheim, "RDF2Vec: RDF Graph Embeddings for Data Mining", *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, Lecture Notes in Computer Science, 2016 498 (cit. on pp. 48, 50).

[146] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System", *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2016 785 (cit. on p. 48).

[147] M. Steinbach, G. Karypis and V. Kumar, "A comparison of document clustering techniques", *In KDD Workshop on Text Mining*, 2000 (cit. on pp. 58, 59).

[148] C. F. Draschner, J. Lehmann and H. Jabeen, "DistSim-Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs", *International Conference on Semantic Computing (ICSC)*, IEEE, 2021 333 (cit. on pp. 57, 87).

[149] H. Jabeen, R. Dadwal, G. Sejdiu and J. Lehmann, "Divided We Stand Out! Forging Cohorts fOr Numeric Outlier Detection in Large Scale Knowledge Graphs (CONOD)", *Knowledge Engineering and Knowledge Management*, ed. by C. Faron Zucker, C. Ghidini, A. Napoli and Y. Toussaint, Springer International Publishing, 2018 534, ISBN: 978-3-030-03667-6 (cit. on p. 63).

[150] *SANSA Team*, `https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.5_ExPAD`, 2022 (cit. on p. 69).

[151] D. C. Kozen, "Depth-First and Breadth-First Search", *The Design and Analysis of Algorithms*, Springer New York, 1992 19, ISBN: 978-1-4612-4400-4 (cit. on p. 72).

[152] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana and M. Vidal, *SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs*, CoRR (2020), arXiv: `2008.07176` (cit. on p. 77).

[153] C. F. Draschner, C. Stadler, F. Bakhshandegan Moghaddam, J. Lehmann and H. Jabeen, "DistRDF2ML - Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs", *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021 4465, ISBN: 9781450384469, URL: `https://doi.org/10.1145/3459637.3481999` (cit. on p. 87).

[154] M. P. C. Oktie Hassanzadeh, *Linked movie data base*, LDOW (2009), URL: `http://linkedmdb.org/` (cit. on p. 88).

[155] L. Akoglu, H. Tong and D. Koutra, *Graph based anomaly detection and description: a survey.*, Data Min. Knowl. Discov. (2015) 626 (cit. on p. 98).

List of Figures

1.1 Chapter, Research Questions and Publication Overview 6

2.1 A sample RDF graph 12
2.2 An RDF triple 15
2.3 Apache Spark architecture diagram 18
2.4 Box and Whiskers plot for Interquartile Range 22
2.5 An example of One-Class Support Vector Machine in a 2D space for detecting anomalies 24
2.6 An example of isolating an anomalous point in a 2D space with IF 25

3.1 Standard anomaly detection pipeline over KGs 32

4.1 Literal2Feature: a sample RDF graph 40
4.2 Literal2feature system architecture abstract overview 42
4.3 Literal2Feature evaluation: depth vs. #features on the Engie dataset 43
4.4 Literal2Feature evaluation: depth vs. time on the Engie dataset 43
4.5 Literal2Feature evaluation: #walks vs. #features on the Engie dataset 44
4.6 Literal2Feature evaluation: branching factor vs. time on synthetic data on a single machine 44
4.7 Literal2Feature execution pipeline (best viewed in color) 45
4.8 Literal2Feature evaluation: depth impact on the classification Result and number of extracted features 49
4.9 Literal2Feature evaluation: processing power scalability on DS 4 Dataset 52
4.10 Literal2Feature evaluation: sizeup performance evaluation over 64 Cores 53

5.1 DistAD: system architecture abstract overview 57
5.2 DistAD: accidents which are detected as anomalies 64
5.3 DistAD evaluation: processing power vs processing time 65
5.4 DistAD evaluation: size-up performance evaluation over 64 Cores 66

6.1 ExPAD: trained decision tree with the target variable *age* 70
6.2 ExPAD system architecture high-level overview 71
6.3 ExPAD: trained decision tree with the target variable *age* 77
6.4 ExPAD: detected anomalies on the Accident Dataset 79
6.5 ExPAD evaluation: processing power scalability on *DS3* 80
6.6 ExPAD evaluation: size-up performance evaluation over 32 Cores 81

6.7 ExPAD evaluation: running time of ExPAD over a large number of features 81

7.1 SANSA stack structure 85
7.2 Configuration of cluster 89
7.3 Installation of SANSA library 89
7.4 High-level micro-service architecture overview 91
7.5 REST APIs Swagger UI 92

List of Tables

2.1 The result of the SPARQL query specified in Listing 2.2 16
2.2 Definitions of evaluation dimensions 19
2.3 Knowledge graph quality metric correlation 20

3.1 Existing works of AD on KGs and their characteristics 33

4.1 Literal2Feature: Main parameters of the framework 47
4.2 Literal2Feature: Dataset statistics (GT=Ground Truth) 48
4.3 Literal2Feature: The F1-Measure evaluation results 50
4.4 Literal2Feature: Silhouette Coefficient 51
4.5 Literal2Feature: Synthetic dataset description 51

5.1 DistAD configurable components 58
5.2 DistAD: Dataset statistics 62
5.3 DistAD: Example of real outliers in DBpedia 63
5.4 DistAD: Dataset description 64

6.1 ExPAD configurable components 74
6.2 ExPAD: Predicates used for generating synthetic RDF graph for class *Person* 76
6.3 ExPAD: Sample of detected anomalies and their explanation 78
6.4 ExPAD: Synthetic dataset description 78

7.1 Available endpoints and their functionalities 93