

# **Scalable Distributed Machine Learning for Knowledge Graphs**

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

vorgelegt von  
Carsten Felix Draschner  
aus  
Bergisch Gladbach, Germany

Bonn 2022

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

1. Gutachter: Prof. Dr. Jens Lehmann  
2. Gutachter: Prof. Dr. Stefan Wrobel  
Tag der Promotion: 23.06.2023  
Erscheinungsjahr: 2023

# Abstract

---

Due to the increasing progress of digitization, immense amounts of data are accumulating, which can be summarized under the term Big Data and form an exciting basis for data analyses. Since the data are heterogeneous and come from many different sources, data integration techniques are beneficial to perform analytics. Knowledge Graphs (KG) link the heterogeneous data within a directed multi-graph by unique resource identifiers. These data can be used for data analytics and prediction methods. One subbranch of Artificial Intelligence (AI) is Machine Learning (ML). ML models are developed and trained, which, based on the available training data, should approximate the target data as closely as possible. The samples in the training data are usually represented by features. For most data analytics and ML approaches, these features are fixed-length numeric feature vectors. However, in the context of KGs, there is no native representation within fixed-length numeric feature vectors. Depending on the use case, these problems can also require the concrete use and inclusion of individual actual values from the KG. The sheer size of some large-scale KG data does not fit into the memory of today's computers. One solution is to use cluster computation through distributed execution, which distributes the data and processing tasks across multiple computers. Both the technologies and the algorithms for this distributed computation must be designated. Due to the possible impact of the results from these data analysis pipelines, special technical implementation of accessible, reproducible, reusable, and explainable approaches is beneficial. These ML and AI development meta-dimensions belong to Ethical AI and Sustainable AI concepts. Within this work, we developed novel approaches for ML on KGs while considering ethical and sustainability dimensions. In particular, we developed technologies that create fixed-length numeric feature vectors. These include methods that, like graph kernels, extract features from the graph in the context of the map-reduce operations relevant for distributed computation. The feature extraction also includes the multi-modal data of KG literals. Accordingly, we have developed methods that enable SPARQL-based feature extraction and assist in creating complex feature-extracting queries. Based on these extracted features, we further contributed scalable, distributed, and explainable ML and data analytics methods such as semantic similarity estimation and classification or regression ML pipelines demonstrating noticeable performance. We support the transparency, reusability, and reproducibility of our novel open-source approaches by results and meta-data semantification. This semantification transfers the original graph data with the hyper-parameter setup and explainability information, in addition to the predicted results of the ML pipelines, into a semantic native KG. Due to the technological complexity, we enable the application of our algorithm technologies through complementary work such as the use in coding notebooks and the use in Rest API-based environments. Our work also describes the multidimensional and interwoven optimization dimensions of ethical and sustainable KG-based ML. We extended the existing technology stack SANSa, which is used for distributed processing and native semantic data handling, by several scientific publications and software framework releases to offer these functionalities for distributed ML on KGs.





# Acknowledgements

---

**Funding** During my Ph.D., I was employed as a researcher at the University of Bonn and part of a team supporting the EU Horizon 2020 Project PLATOON<sup>1 2</sup>

**Institutions** I want to take this opportunity to thank several institutions that have made my doctorate possible and have accompanied me on this path. First, I would like to thank the University of Bonn for making my studies and doctorate possible. I would also like to thank the Studierendenwerk, which has provided us students, with a diverse and supportive range of services, from sports to housing and even lunch. In particular, I would also like to thank the Argelander program, which offers diverse and interdisciplinary voluntary courses and workshops through the Doctorate Plus program in the Research, Research Management, and Business and Organizations tracks.

**Supervisors** Tremendous thanks go, especially to my supervisor, Jens, and my mentor Hajira. Thank you very much for offering me a place in the SDA and enabling me to complete my Ph.D. through a mixture of trust, freedom, and support. Through your work, help and provided environment, despite the corona pandemic, the considerable project challenges, and the significant changes within the SDA, I still could comprehensively conduct applied and fundamental research, supervise students, further develop open-source software projects and contribute to the EU Horizon 2020 project PLATOON. I appreciate it very much to have received from you not only always very competent but also consistently friendly and fair support. I also thank my second supervisor Stefan Wrobel, whose initiative helped me considerably during the transition phase in the SDA Research Group 2022 and on the final steps of thesis submission.

**Colleagues** I would also like to thank my colleagues. You have always been helpful, whether it was about brainstorming, technical challenges, administration, or just a pleasant exchange and chat. So I would like to thank you, the whole SDA, and especially Martina and Peter, for your administrative support, you, Fathoni, Tobi, and Firas for the excellent cooperation within our PLATOON project, you, Claus, Lorenz, and Patrick for your help in the development of the SANSA stack and you Farshad for supporting so many joint projects with me. I would also like to thank the bachelor and master students I have worked with and supervised.

**Friends** A huge thanks go to my friends. I want to thank my friends who have always created a great time next to work through the varied sports and recreational events. Besides the pleasant and relaxing free time, I also know I can rely on you, which is an incredible privilege. So thank you: Alex,

---

<sup>1</sup> <https://cordis.europa.eu/project/id/872592>

<sup>2</sup> <https://platoon-project.eu>

Andreas, Basti, Clara, Elena, Ha Ahn, Hanna, Jannik, Jil, Jonas, Jonas, Joseph, Judith, Konsti, Laura, Laura, Leon, Linda, Lucas, Madita, Malte, Moritz, Nils, Suse, Thorben, Tim and Tobi. Franzi, I would like to thank you so much for all you did; you've been the one who accompanied and supported me the most over recent years! Thank you!

**Family** My dear family, I am so grateful that I have you, that you always support me, that you are there for me, and that I can count on you. Primarily I want to thank my dear parents, Sanja and Frank, my siblings, Adrian and Nadja, my grandma Edith and my brother-in-law Julius. And I would like to thank you, Charlotte, you made me an uncle at the beginning of 2020, and since then, you have given me a lot of joy and love.

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Problem Definition and Challenges . . . . .	6
1.3	Research Questions . . . . .	8
1.4	Thesis Overview . . . . .	9
1.4.1	Contributions . . . . .	9
1.4.2	Publications . . . . .	11
1.4.3	Thesis Outline . . . . .	12
<b>2</b>	<b>Preliminaries</b>	<b>13</b>
2.1	Knowledge Graphs, RDF and Semantic Analytics . . . . .	13
2.1.1	Knowledge Graphs (KG) . . . . .	13
2.1.2	Resource Description Framework (RDF) . . . . .	14
2.1.3	Ontology . . . . .	15
2.1.4	SPARQL Query Language . . . . .	16
2.1.5	Apache Jena . . . . .	17
2.2	Scalable and Distributed Data Analytics . . . . .	17
2.2.1	Distributed Computing . . . . .	17
2.2.2	Hadoop Distributed File System (HDFS) . . . . .	18
2.2.3	Apache Spark . . . . .	19
2.2.4	Scala . . . . .	20
2.3	Artificial Intelligence and Machine Learning . . . . .	20
2.3.1	Artificial Intelligence (AI) . . . . .	21
2.3.2	Machine Learning (ML) . . . . .	21
2.4	AI Ethics and Sustainable AI . . . . .	22
2.4.1	AI Ethics . . . . .	22
2.4.2	Sustainable AI . . . . .	22
<b>3</b>	<b>Related Work</b>	<b>23</b>
3.1	Knowledge Graph data in downstream ML Pipelines . . . . .	24
3.1.1	KG Feature Tensors . . . . .	24
3.1.2	Query-based Feature Extraction and Propositionalization . . . . .	24
3.1.3	Graph Kernels . . . . .	25
3.1.4	Knowledge Graph Embedding (KGE) . . . . .	25
3.2	Semantic Similarity Estimation . . . . .	26
3.2.1	Similarity Estimation Families for Knowledge Graphs . . . . .	26

3.2.2	Feature Based Semantic Similarity Estimation . . . . .	27
3.2.3	Probabilistic Similarity Estimation . . . . .	28
3.2.4	Frameworks . . . . .	29
3.3	Scalable Semantic Analytics . . . . .	29
3.3.1	KG Data Analytics Complexity . . . . .	29
3.3.2	Distributed KG Processing Frameworks . . . . .	29
3.4	KG ML Meta-Dimensions . . . . .	31
3.4.1	Reusability and Reproducibility through Metadata Semantification . . . . .	31
3.4.2	Ethical AI . . . . .	31
3.5	Summary . . . . .	32
<b>4</b>	<b>Scalable Similarity Estimation for RDF KGs</b>	<b>33</b>
4.1	Motivation . . . . .	33
4.2	DistSim Approach . . . . .	34
4.2.1	Pipeline Architecture . . . . .	34
4.2.2	DistSim as Resource . . . . .	35
4.2.3	DistSim Feature Extraction . . . . .	36
4.2.4	Semantic Similarity Estimation Models . . . . .	38
4.2.5	(Hyper-)Parameter Setup Effects . . . . .	39
4.2.6	Use Cases . . . . .	39
4.3	Experiment and Evaluation . . . . .	40
4.3.1	Dataset Description . . . . .	41
4.3.2	Scalability over increasing horizontal Cluster Computation . . . . .	41
4.3.3	Scalability over Dataset Size . . . . .	41
4.3.4	Hyper-Parameter Evaluation . . . . .	42
4.4	Summary . . . . .	44
<b>5</b>	<b>An Automatic Scalable RDF Graph Feature Extractor</b>	<b>45</b>
5.1	Motivation . . . . .	45
5.2	Literal2Feature Approach . . . . .	46
5.2.1	Pipeline Architecture . . . . .	47
5.2.2	Seed Generator . . . . .	47
5.2.3	Graph Search . . . . .	47
5.2.4	Path Generator . . . . .	48
5.2.5	SPARQL Generator . . . . .	48
5.2.6	SPARQL Executor . . . . .	48
5.2.7	Literal2Feature as a Resource . . . . .	48
5.2.8	Use Cases . . . . .	49
5.3	Experiment and Evaluation . . . . .	49
5.3.1	Data Description . . . . .	50
5.3.2	Performance Evaluation . . . . .	50
5.3.3	Discussion . . . . .	55
5.4	Summary . . . . .	56

<b>6</b>	<b>Distributed ML Pipelines for RDF KGs</b>	<b>57</b>
6.1	Motivation . . . . .	57
6.2	DistRDF2ML Approach . . . . .	58
6.2.1	Pipeline Architecture . . . . .	59
6.2.2	SparqlFrame . . . . .	60
6.2.3	SmartVectorAssembler . . . . .	62
6.2.4	Machine Learning Models . . . . .	63
6.2.5	Semantification of Results: . . . . .	64
6.2.6	DistRDF2ML as a Resource . . . . .	64
6.3	Experiment and Evaluation . . . . .	66
6.3.1	Data Description . . . . .	66
6.3.2	Description Evaluation Dimensions . . . . .	66
6.3.3	Processing Power vs Processing Time . . . . .	67
6.3.4	SPARQL Complexity vs Processing Time . . . . .	68
6.3.5	Dataset Size vs Processing Time . . . . .	68
6.3.6	Spark Setup vs Processing Time . . . . .	70
6.3.7	Sample DistRDF2ML Pipelines . . . . .	70
6.3.8	Discussion . . . . .	70
6.4	Summary . . . . .	71
<b>7</b>	<b>Distributed Explainable Multi-Modal Similarity Estimation</b>	<b>73</b>
7.1	Motivation . . . . .	73
7.2	SimE4KG Approach . . . . .	74
7.2.1	Pipeline Architecture . . . . .	74
7.2.2	Probabilistic Collection of Candidate Pairs . . . . .	75
7.2.3	Feature Extraction . . . . .	76
7.2.4	Domain Aware and Multi-Modal Similarity Estimation . . . . .	77
7.2.5	Weighting of Features and overall Similarity Assessment . . . . .	79
7.2.6	Semantification of Results and Meta Data . . . . .	79
7.2.7	Impact & Use Cases . . . . .	80
7.2.8	Quality, Reusability, and Availability . . . . .	81
7.3	Experiment and Evaluation . . . . .	82
7.3.1	Dataset Description . . . . .	82
7.3.2	Dataset Size vs. Processing Time . . . . .	82
7.3.3	Processing Power vs. Processing Time . . . . .	84
7.3.4	Comparison of SimE4KG to Distsim and DistRDF2ML . . . . .	85
7.3.5	Discussion . . . . .	86
7.4	Summary . . . . .	86
<b>8</b>	<b>Implementation and Technical Setup</b>	<b>87</b>
8.1	Resources . . . . .	88
8.1.1	GitHub Repository . . . . .	88
8.1.2	Documentation . . . . .	89
8.1.3	Releases . . . . .	90

8.2	Scalable Semantic Analytics within PaaS . . . . .	90
8.2.1	Motivation . . . . .	91
8.2.2	Technical Setup . . . . .	91
8.2.3	Summary . . . . .	92
8.3	Micro-Service based Semantic Analytics integration . . . . .	92
8.3.1	Motivation . . . . .	93
8.3.2	Technical Setup . . . . .	94
8.3.3	Summary . . . . .	96
<b>9</b>	<b>Ethics and Sustainability in KG-based ML</b>	<b>97</b>
9.1	Motivation . . . . .	97
9.2	KG-based ML Pipeline Schema . . . . .	98
9.3	Responsible AI Approach . . . . .	99
9.3.1	Ethical KG-based ML Approach . . . . .	99
9.3.2	Responsible Investment of Resources . . . . .	99
9.4	System Setup . . . . .	100
9.4.1	Efficient Hardware Usage . . . . .	100
9.4.2	Optimizing Carbon Footprint . . . . .	100
9.5	Data Insights . . . . .	101
9.5.1	Knowledge Graph Data Reliability . . . . .	101
9.5.2	Protecting Privacy vs. Enrichment of Context . . . . .	102
9.6	Knowledge Graph-based Machine Learning . . . . .	103
9.6.1	Ethical & Explainable ML for KGs . . . . .	103
9.6.2	Critical and Sensitive Features . . . . .	104
9.7	Training and Evaluation . . . . .	104
9.7.1	Bias in Training and Evaluation . . . . .	104
9.7.2	Sustainable Training . . . . .	105
9.8	Accessible Deployment . . . . .	106
9.8.1	Prediction and Meta Data Semantification . . . . .	107
9.8.2	Crowd Sourced AI Labeling and Intervention . . . . .	107
9.9	Summary . . . . .	107
<b>10</b>	<b>Conclusion</b>	<b>109</b>
10.1	Review of Contributions and Conclusions . . . . .	109
10.2	Future Work . . . . .	111
10.3	Closing Remarks . . . . .	114
	<b>Bibliography</b>	<b>115</b>
	<b>List of Figures</b>	<b>131</b>
	<b>List of Tables</b>	<b>133</b>

## Introduction

---

Within this chapter, we motivate the topic of this thesis in Section 1.1. We primarily present how knowledge graph (KG) data and scalable machine learning (ML) are interconnected and what are the exciting meta-dimensions. Section 1.2 illustrates the specific problems and challenges of developing scalable distributed ML for KGs. Based on the motivations and challenges we formulate in Section 1.3, the research questions guided the efforts presented in this thesis. Finally, in Section 1.4, we provide an overview of our core contributions (see Sect. 1.4.1). In Section 1.4.2, we present our underlying scientific publications that we have developed as part of this thesis and which build the basis of several core chapters. In this section, we also describe the contributions of each author. Section 1.4.3 provides a short overview of chapters' content and how these match with scientific publications that have been developed (see Fig. 1.3).

### 1.1 Motivation

Within this section, we motivate our research efforts in *scalable distributed ML for KGs* (See Fig. 1.1). For this purpose, we introduce KG as an exciting but challenging data source for data analytics and ML. Furthermore, we refer to the possible data analytics and ML approaches. On the one hand, we present classical downstream ML pipelines on KG data and, on the other hand, semantic similarity estimations approaches. In particular, we explain how the complexity of KGs requires scalable approaches like distributed ML and data analytics. Finally, we explain which further meta-dimensions can be considered in KG-based ML. These include the subdomains AI Ethics and Sustainable AI. We introduce reusability, reproducibility, and explainability as properties of ethical and sustainable algorithms and libraries.

**Knowledge Graphs** In recent decades, our world has changed tremendously due to extensive digitization. Many added values and analyses can be created using data. The data sources and formats are diverse. Data formats include text, numbers, date-time, images, audio, video, or graph structures. Data integration of source information is necessary for the data-driven solutions of various use cases. The information and correlations depicted in the data can be represented in knowledge graphs (KG). In KGs, information is mapped as Linked Data. Directed labeled edges connect information and entities in a multi-graph. Tim Berners Lee introduced the Semantic Web [1], describing the vision to interlink facts, knowledge, and data sources from the internet. Entities in the KG are uniquely identified by

Unique Resource Identifiers (URI). If more than one URI represents the same entity, these URIs can be linked by *sameAs* relations. Relations play an outstanding role when representing information within a KG. The relations allow complex interlinked data structures. The W3C has standardized Linked Data within KGs through the Resource Description Framework (RDF) [2]. Data is represented here in triples. Ontologies were introduced to support the standardization and harmonization of the used URIs. Ontologies can define more complex vocabularies than taxonomies. Taxonomies and ontologies can be found for several data sub-domains. In these technological developments, both open available KGs and enterprise KGs emerged. These KGs avail different approaches to cover data domains with varying complexity and expressivity. Several openly available KGs merge various information sources into holistic or domain-specific KGs. DBpedia, YAGO, FreeBase, LMDB, and Wordnet [3–7] are well-known KG datasets. Also, several companies use KG as the core of their technologies. These include Alphabet/Google for access to information or Meta/Facebook to represent social media graphs [8, 9]. Based on the Semantic Web vision and the KG technologies, further domains transform and integrate various data sources as semantically linked data resulting in domain-specific KGs. Data integration within KGs through labeled directed relations allow understanding of the information in the context of the overall data. Information integrated into KG contains dense, interlinked native graph structures and allows storing and representing multi-modal source data within associated literals like numeric values, date-time, or texts. These strengths of data integration through KGs and the possible representations of complex information contexts form an outstandingly exciting data source for building ML, and data analytics approaches.

**Machine Learning on Knowledge Graphs** KGs build an exciting source for data analytics. Data analytics, data science, ML, and AI help solve various use cases. Use cases can cover economic or socially beneficial projects. Examples of socially beneficial KG-based projects are search engines improving access to knowledge or scientific publication. Examples of KG-based ML with economic impact are on-demand streaming services or e-commerce shops providing user-specific recommendations. Classical ML approaches based on KG data can improve the KG itself or create standalone results. Improving the KG data can include enrichment, completion, or improving quality. Examples of approaches for KG-based ML improving the source KG are entity matching or entity resolution. These approaches try to merge nodes if they represent the same entity or resolve wrongly merged nodes. KG-based ML can also enhance the source KG through link prediction. Link prediction inserts new triples, which likely should be valid and have not been present in the actual KG. KG-based recommendation systems could exploit these link predictions. Standard ML pipelines can be used to perform classification and regression. These classification or regression results can be used for KG data type annotations and literal value predictions. Same time these predicted values can be the bases for standalone data analytics approaches. Additionally, ML pipelines on KG data can be used to cluster KG entities. Clustering, anomaly detection, and recommendations can rely on similarity estimations of entity node pairs. Also, semantic similarity estimation can be used for entity matching and resolution. ML operates on feature representations of samples. In the case of KG data, these features represent KG substructures or single KG entity nodes. For several use cases, KG-based ML must be capable of operating on complex and multi-modal linked data structures. Also, further challenges need to be investigated when using KG data as source data for KG-based ML. Example data in KG could include temporary valid information or entity disambiguation. The following data could be present in a KG: *Joe Biden is currently the US president in 2022, but he will hold this role only temporarily.* Other



information sources might use the terms *Joe Biden* and the *US president* synonymously. Same time entities like the city of *New York* might be referred to as *the Big Apple*. Another challenge can be that *Apple*, *apple*, and *the Big Apple* should have different entity nodes as they sometimes might refer to the US city, sometimes to the fruit, and sometimes to the tech company. Covering this data and domain knowledge can be challenging. KG data is challenging in ML processes due to its structural complexity. Most of the ML approaches require fixed-length numeric tensor representations. As a KG could be used for representing, e.g., social media data, it is likely that certain person instances have broadly distributed numbers of *friendOf* or *likes* relations to other entities. KG entity samples must represent these arbitrary number of associated features in aligned fixed-length numeric feature vectors. First, algorithms, libraries, and frameworks try to provide needed feature extraction and representation learning approaches. Some use Knowledge Graph Embeddings (KGE). KGEs start for all elements of the KG with random feature vectors, so-called latent embeddings. These KGEs are optimized by a procedure assuring that they meet specific mathematical criteria. An example is TransE [10], which tries to optimize the latent embeddings to meet the mathematical loss across all data triples. Alternatively, graph kernels collect surrounding data of each entity in the graph by, e.g., random walks. For some of the possible data analyses, however, the KGEs may not fit because, in other cases, concrete KG literal values are crucial, such as the exact *number of inhabitants of a region*. In our work, we repeatedly came across use cases that use KG as an information base in connection with the unique data integration possibilities. In this thesis, we target the development of classical data analytics and ML approaches for this multi-modal KG data. We explicitly investigate how KG-based ML can be executed in distributed environments, running via scalable algorithms and considering the meta-dimension of ethical AI.

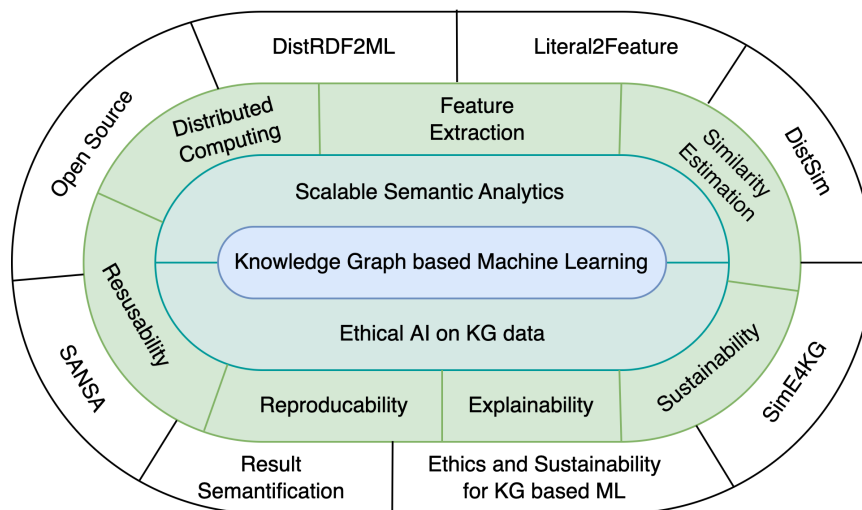


Figure 1.1: Research dimensions, considerations and outcomes

**Scalable Knowledge Graph Analytics** When KGs are the core of data analysis projects, it must be considered what immense sizes they can have [11, 12]. This data scale leads to the fact that KG no longer fits into the main memory (RAM) of classic computers when it comes to processing and further use. The main memory in computers is limited by the modules produced and available on the market

for a machine. Likewise, upgrading becomes disproportionately expensive from the point where the hardware specifications leave the space of classic consumer hardware. The solution is to scale horizontally as hardware requirements increase. This horizontal scaling means that several computers in a cluster provide the necessary resources. Software frameworks are used that enable the distributed management and processing of data. The Hadoop Distributed File System (HDFS) is used for the distributed management of data. Apache Spark [13] is widely used for distributed data processing, particularly in data analysis pipelines. The existing Apache Spark data analysis pipelines require data in tabular form. The Scalable Semantic Analytics Stack SANSA [14] was developed in this context. It provides methods to operate natively on RDF KGs by using Apache Spark [13] and Apache Jena [15]. Distributed execution of ML pipelines and data analytics on KG data have unique challenges in feature retrieval. The graph traversal, the embeddings, and feature collection across multiple samples distributed over multiple machines are technical challenges. Cluster communication of multiple machines is a bottleneck because of the needed network connection between the machines. Within our work, we investigate the opportunities of scalable distributed semantic similarity estimations and downstream machine learning pipelines on multi-modal KG data.

**Semantic Similarity Estimation** Semantic similarity estimation for KG data is crucial for several possible semantic data analytics and ML scenarios. These analytics require semantic similarity estimation of entities in the KG. All or a subset of entities in the KG are assigned similarity scores that indicate how similar they are in pairs. These semantic similarity scores can be used for several ML predictions. In clustering, elements are assigned to their similarity scores in groups/clusters. In recommendation systems, alternatives remarkably similar to the previously preferred ones can be suggested and would otherwise have been overlooked. Entity Matching can be applied if the similarity score is sufficiently high, inducing a likelihood that the nodes represent the same entity. Also, classification can be implemented by giving entities with missing labels a weighted label of the most similar entities with existing/labeled classes.

**Downstream Machine Learning pipelines** Our efforts also target scalable distributed downstream ML pipelines on the multi-modal KG data. There are relevant models for regression or classification already implemented in distributed frameworks like Apache Spark [16], for example, Random Forest or Multi-Layer Perceptron (MLP). These frameworks do not offer native handling of RDF KG triple data. In principle, downstream data science pipelines use multiple software modules to be concatenated in pipeline architectures. ML training and prediction operate on fixed-length numeric feature vectors. For samples or entities in KGs, the features consist of both associated entities and the values in literals. This information can be extracted and retrieved from a KG using SPARQL queries. Through technologies such as Sparqlify [17] or Ontop [18], SPARQL to SQL rewriting is used and can be used on existing Apache Spark HDFS systems for distributed extraction of features to entities. Since KG entities can have any number of associated features, sampling, compression, or accumulation must be used to provide the required fixed-length numeric feature vectors. These assembled vectors build the basis for training and evaluation datasets. ML models use these data representations to train and also predict. Further examples of ML pipelines can be found in Section 2.2.3.

**Research, Development, and Deployment considerations** Our scalable distributed ML developments for KG involve technical and algorithmic results but additionally investigate meta-dimensions.

An increasingly relevant topic in Artificial Intelligence (AI), Data Science, and ML is optimizing these developments' transparency and ethical aspects. Several options can be taken to offer holistically better solutions in the technologies' research, development, and deployment. Proper implementations enable explainable, accessible, reproducible, and reusable ML components. *Explainable AI* means that the prediction of an AI can be justified or explained. This explainability in KG-based ML means that the feature vectors and their entries can be assigned meaning and effect on the result. The results can also be processed and semantified so that they can be better understood by humans interacting with the AI. Explainability can also mean that the result provides insight into the decisive features based on which a decision was made. In addition to excellent and comprehensible access to the results and predictions, good *accessibility* to the technology is also essential. This accessibility allows the technology to be used and verified by as many interested stakeholders as possible. The integration as a modular library or framework deployed on platforms like GitHub and Maven enables this. Likewise, open-source GitHub repositories can be used to inspect details, find bugs, and contribute to the further development of the technology. Good documentation is also necessary for the interaction. This documentation includes a basic explanation of the technology, code docs, code samples, and sample pipelines that can be easily adapted for specific use cases. In addition, scientific publications can strengthen and validate the methodological part through the review process with the scientific community. In addition to stakeholders working directly on the technology stack, others want to try out the technology or integrate it into existing architectures. Trying it out is possible in web-hosted notebooks like Google Colab or Databricks [19] using only a browser and processing power provided for free without installation. Technologies such as Docker and Rest make it possible to run AI as a service and use it standardized through an abstract interface. Due to the impact and the interest in the further development of ML-supported technologies, it is relevant to be able to reproduce the results. This *reproducibility* can be enabled by a dense linking of the ML pipeline (hyper-) parameter setup with the created results. By linking the results and predictions with the original KG, it is possible to operate on all data and metadata in a semantic native way. The already introduced open-source principles, documentation, samples, fixed version releases, and sample data make it easier to reproduce existing technology results. Due to the effort and the resources used in ML prediction, it would be advantageous if the calculated results could be reused. *Reusability* is supported by generic and modular technologies and easily accessible results and predictions. The library designs popularized by scikit-learn [20] in the form of transformers allow the arbitrary concatenation of exciting models that enable a final use-case-specific ML pipeline. The results should be well and densely linked in a semantified version to the original data to derive further conclusions for subsequent analyses.

**Ethical and Sustainable dimensions of KG-based ML** Apart from the developments covering reusability, reproducibility, accessibility, and explainability, a holistic investigation of further KG-based ML usage and impact is needed. The complexity of KG and the application of ML approaches to it has significant ethical and sustainability implications. The sustainability implications of KG-based ML arise primarily from the sheer complexity of the data. On the one hand, these are very large and, on the other hand, can require considerable hardware resources for training, e.g., the optimization and processing of KGE. KGs also change regularly and can thus require regular updates in gathered feature representations. A significant part of the CO<sub>2</sub> footprint is the production and provision of hardware and electrical energy [21]. Efficient use of hardware and electricity is possible through various optimizations. These include selecting the company that should provide the electricity through

renewable energies. The ethical implications of KG-based ML are central to the KG data used as training and evaluation data and to the ML methods and resulting predictions based on that data. The data originate from various sources, each having a different certainty of correctness and completeness. The available entities of the training and evaluation data samples can also contain biases in critical features. The enormous data integration possibilities enable combining data from the most diverse resources. An enriched KG resource enables use cases beyond the initially intended approaches. It is essential to consider which ethical scenarios are enabled by ML-enhanced KG. Furthermore, many of today's ML models are perceived as black boxes. Since latent embeddings like KGEs or graph neural networks are also used in the context of KG-based ML, the feature vectors and the results are increasingly difficult to explain. These explainable feature vectors are necessary to build trust through transparency in use cases with societal impact. Through the possibilities of semantic data and conversational AI, interactive ML models have become increasingly feasible.

## 1.2 Problem Definition and Challenges

The semantic linked data resources in the KGs provide an exciting starting point for data analytics. However, data analytics are challenging, especially regarding arbitrarily complex graph structures and multi-modal features. A significant part of the challenge is the sheer scale of data. As a result, the existing data volumes no longer fit into the main memory of individual computers. There is a need for easily accessible libraries and frameworks to support the construction of data analytics pipelines. These must be distributed, scalable, easy to use, and native to RDF KGs.

**Explainable Scalable Distributed in-memory Semantic Similarity Estimations for multi-modal RDF Knowledge Graphs** For a variety of data analytics, semantic similarity estimations are essential. These include Clustering, Recommendation Systems, Entity Matching, and many others. Since the similarity estimation of all entity pairs has quadratic complexity, the approach's scalability must be considered. Furthermore, the complexity of the similarity estimation of exactly two entities is relevant. Due to the structure of the data and the complexity of the multi-modal features, the applied similarity estimation approaches need to be capable of handling these multi-modal features. Special treatment of RDF KGs is needed because relevant features lie within the literal of the KG for some use cases. These are multi-modal and also not equally represented across all entities. Due to the complexity of the all-pair estimation, the results must be reusable. Reproducibility and explainability are also relevant to enable trustworthy analytics based on these ML pipeline results. We target these challenges and problems with our Research Questions RQ1 and RQ4.

**Explainable Scalable Distributed in-memory Downstream ML Pipelines for multi-modal RDF Knowledge Graphs** Similarly to similarity estimations, downstream ML pipelines operating in RDF KGs' literals are needed for some use cases. However, the graph's structure may be so complex that the feature-extracting SPARQL queries cannot be quickly produced. If there is no ontology for the KG or no good knowledge about SPARQL, the creation of feature-extracting SPARQL queries becomes a challenge. Alternative approaches like Knowledge Graph Embeddings (KGE) are trained based on random initial vectors. So fixed length numeric feature vectors for KG entities are not explainable when generated by, e.g., KGE approaches. Explainable feature vectors are beneficial if targeting follow-up explainable ML pipelines. Existing data analytic pipelines for scalable processing

of ML pipelines operate on tabular data. The missing support in distributed ML libraries of RDF triple data means that they cannot be applied straightforwardly to the linked data KGs. Also, simple stackable downstream ML pipelines through transformer-based modules are considered standard, such that the required pipelines can be constructed quickly through generic modules. In addition, several meta-levels are part of good downstream ML pipelines for KG. These are accessibility, reusability, reproducibility, and explainability. These challenges and problems are reflected in Research Questions RQ2 and RQ3.

**Ethical and Sustainability Challenges in KG-based ML Pipelines** Due to the considerable impact of AI and analytics on KG data, further dimensions that can be summarized under the term ethical AI should be considered. For developed tools, it must be ensured that the entire community and stakeholders have good access to these technologies. For this purpose, both code, libraries, and frameworks must be available as open-source code. Accessibility facilitates the reusability and verifiability of the methods. Improved access also allows more people to participate in further development. Both explainable ML methods and reproducible pipelines can establish confidence in ML predictions. Specific software and ML pipeline approaches must be used to make this possible. The widely established KGE as the basis for fixed numeric feature vectors has problems in explainability and scalability due to missing out on sample and inductive methods. Besides the pipeline approaches, care must also be taken to handle training datasets responsibly. Dataset insight analytics includes an awareness of the problems related to data validity and the possible bias of the data. The OWA may limit data validity. However, automatic knowledge retrieval can also contain faulty sources or algorithms. As well as by deliberate misinformation and sarcasm or humor, wrong information can be mapped. Bias also occurs at the feature level, which includes non-representative distributions. Bias in data can include features associated with discrimination, such as age, origin, gender, sexual orientation, skin color, or political orientation. Also, sustainability is a significant factor in such complex processes as big scalable data KG-based ML pipelines. These use powerful hardware and power resources. These resources must be both minimized and justified. Sustainable AI can be supported by transparent and high awareness of the carbon resource footprint. These ethical and sustainability considerations for KG-based ML are reflected in RQ5.

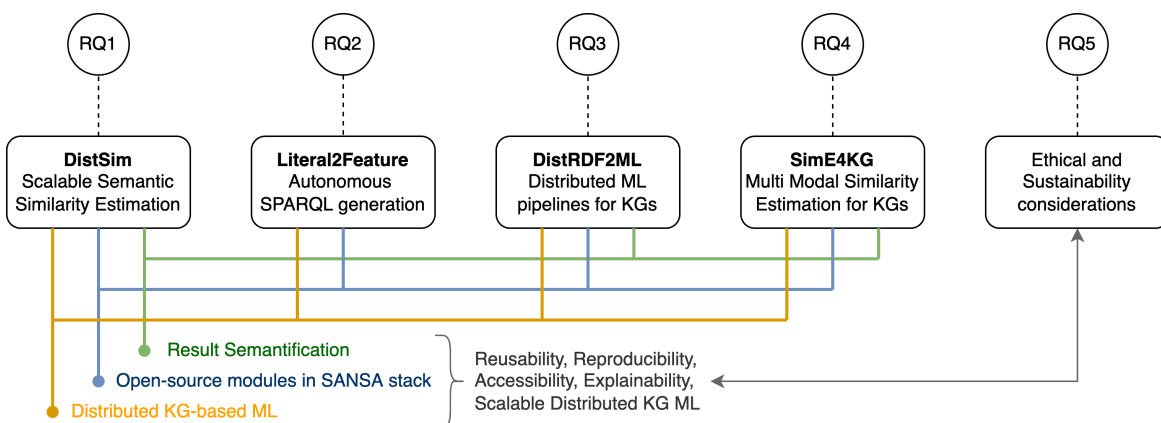


Figure 1.2: Research question and contributions

### 1.3 Research Questions

Based on the motivation and dimensions outlined in Problem Definition and Challenges (see Sect. 1.2), we define the core research question in this thesis as follows:

**Overall RQ: How can we perform Scalable Distributed Machine Learning on RDF Knowledge Graphs?** This overall research question breaks down into multiple specific research questions. The challenges are mapped to specific research questions and all together contribute to the overall research problem definition tackled by this thesis.

**RQ1: Can we perform horizontal scalable Semantic Similarity Estimation on RDF KGs?** In order to answer this question, we have adapted existing feature-based Semantic Similarity Estimations and corresponding probabilistic approaches for the application on RDF KGs. For this purpose, the Scalable Semantic Analytics Stack SANSA, which uses Apache Jena and Apache Spark for processing large-scale RDF KGs, was extended with corresponding functions. Through modular feature extraction and the use of MinHash Locality Sensitivity Hashing, a horizontally scaling implementation of feature-based similarity estimation was possible (see Chapter 4).

**RQ2: How can we assist SPARQL query-based feature extraction for RDF KGs?** To solve this problem, we developed a new approach, which assists the generation of feature-extracting SPARQL queries without extensive knowledge of the KG. The technology stack SANSA was extended by the Literal2Feature module, which searches for interesting graph patterns to extract relevant literals fed into ML pipelines. The SPARQL queries are generated so that they can create explainable feature vectors by using relation's URIs within the query (projection-) variables (see Chapter 5).

**RQ3: Can we perform scalable KG literal value-based ML pipelines?** We can solve this research question by introducing the DistRDF2ML approach. The existing transformer-based pipelines of Apache Spark MLlib operate on tabular DataFrames. Several new modules have been integrated into the SANSA stack for end-to-end native RDF KG downstream ML pipelines, which operate specifically on multi-modal literal values. Based on the feature extracting SPARQL queries automatically created by Literal2Feature, the new SparqlFrame Transformer can extract these features. A native Apache Spark DataFrame is created by using SPARQLIFY. For downstream ML pipelines, fixed-length numeric feature vectors are relevant. These can be generated based on the output of the SparqlFrame Transformer using the SmartVectorAssembler. The results of Apache Spark MLlib-based ML pipelines can be transformed into native RDF KG results by new modules, which allow direct links to the original data as well as a mapping to the (hyper-) parameter setup of the pipeline (see Chapter 6).

**RQ4: Can we perform Explainable Semantic Similarity Estimation on large-scale multi-modal RDF KGs?** As an answer to this question, several already developed approaches are combined. The SANSA stack driven by Apache Spark and Apache Jena was extended based on the ideas of DistSim to handle the multi-modal features of RDF KGs. For this purpose, the feature extraction approaches from Literal2Feature and DistRDF2ML are reused. The SPARQL-based feature extraction is replaced by a performance-optimized novel feature extraction Transformer called SmartFeatureExtractor. The multi-modal similarity estimations are computed and aggregated by the SimE4KG framework. The

dedicated computation of individual feature-based similarity scores enables an explainable presentation of the estimation in the semantified version of the results (see Chapter 7).

**RQ5: Which Ethical and sustainable dimensions can we consider in KG-based ML?** We answer this research question by presenting a number ethical and sustainability dimensions. These are presented across the entire research and development pipeline in KG-based ML. For each step, several scenarios are presented. These consist of the definition, the declaration of the particular challenges, a presentation of exemplary examples, the formulation of dedicated research questions, and several recommendations for optimization (see Chapter 9).

## 1.4 Thesis Overview

This section introduces our major contributions, references to developed scientific publications, and a thesis outline.







Chapter 4	Chapter 5	Chapter 6	Chapter 7	Chapter 8	Chapter 9
RQ1	RQ2	RQ3	RQ4	Tech/Implementation	RQ5
					
Draschner et. al. at IEEE ICSC 2021	Moghaddam, Draschner et. al. at Semantics 2021	Draschner et. al. at ACM CIKM 2021	Draschner et. al. at IEEE AIKE 2022	Draschner, Moghaddam et. al. at LAMBDA DW 2021	Draschner et. al. at IEEE AIKE 2022

Figure 1.3: Chapter, Research Questions and Publication Overview

### 1.4.1 Contributions

Our contributions cover several areas of Scalable Distributed Machine Learning on KGs. These areas include the automatic development of feature-extracting SPARQL queries, distributed Semantic Similarity Estimation, and the construction of scalable downstream ML pipelines for RDF KGs. Additionally, we consider technical implementation optimizations for Ethical and Sustainability considerations of KG-based ML (see Fig 1.2).

1. *DistSim - Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs*: DistSim includes several feature-based graph entity similarity estimators. The Similarity Estimation Pipeline was created by merging generic software modules. DistSim introduces MinHash with locality sensitivity hashing for large-scale RDF data to improve scalability over all-pair similarity estimations. DistSim’s modules can be configured using a variety of (hyper-) parameters to manage the trade-off between information considered and processing time. Furthermore, the similarity estimation pipeline’s output is native RDF. DistSim is part of the SANSA stack, with documentation in Scala-docs and unit tests [22].

2. *Literal2Feature - An Automatic Scalable RDF Graph Feature Extractor*: Literal2Feature is a generic, distributed, and scalable software framework for translating massive RDF data into an explainable feature matrix. Many standard ML algorithms can make use of this matrix. Our method uses Semantic Web and Big Data technologies to extract a variety of features from a big RDF graph by deep traversing it. The proposed framework is open-source, technically documented, and wholly integrated into the Semantic Analytics Stack community project [23].
3. *DistRDF2ML - Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs* The generic, scalable, and distributed framework DistRDF2ML is presented for building in-memory data preprocessing pipelines for Spark-based ML on RDF KGs. This framework includes software modules that convert large-scale RDF data into fixed-length numeric feature vectors suitable for ML. The generated modules are tailored to KGs' multi-modal nature. The pipeline's modules, hyper-parameters, and outputs are all exported in a semantic structure that can be utilized to improve the original KG. The semantic representation of metadata and ML outcomes has the advantage of enhancing the reusability, explainability, and reproducibility of ML processes [24].
4. *Semantic Analytics in the palm of your browser*: Setting up a distributed in-memory cluster computation system with all its dependencies and environments can be challenging, and it may also necessitate significant processing power. We lower the entry barriers for analyzing and testing the SANSA framework's capabilities and potentially bringing this technology to production using only the browser. We show how to run the SANSA stack within Databricks without requiring extra Apache Spark knowledge or installs [25].
5. *Semantic Web Analysis with Flavor of Micro-Services*: Developing distributed KG-based ML pipelines demands in-depth knowledge and experience in computer systems, networking, distributed computing, and other related fields. Furthermore, even if developers have the technical skills, creating such a cluster takes time and effort. To address the challenges mentioned above, we present a micro-service Rest-API-based architecture that allows end-users to readily access the capabilities of SANSA and distributed semantic data analysis without requiring technical expertise in these domains [26].
6. *SimE4KG - Distributed Explainable multi-modal Semantic Similarity Estimation for Knowledge Graphs*: We present the SimE4KG framework, which provides generic open-source modules for computing semantic similarity estimation in multi-modal KGs. SimE4KG delivers reproducible, reusable, and explainable results to justify the computing costs of similarity estimation. The pipeline outcomes are a native semantic RDF-KG, including the experiment results, hyper-parameter setup, and result explanation, as the most influential features. We utilized Apache Spark to develop a distributed strategy for efficient and scalable execution in memory. This framework's overall development is integrated into the SANSA distributed semantic analytics stack [27].
7. *Ethical and Sustainability considerations for Knowledge Graphs based Machine Learning*: This paper presents ethical and sustainability considerations in KG-based ML. These are presented along the classical R&D life-cycle, including responsible AI check, technical system setup, data insights, machine learning, training & evaluation, and finally, deployment. Each of these steps is introduced by multiple sub-dimension. These are structured by definition, explaining



challenges, providing examples, stating research questions, and offering recommendations. We also investigate data validity, bias, and privacy-related concerns of KG data in ML pipelines. The ethical and explainability dimension, as well as the sustainability aspect of processing power-intensive KG-based ML, are presented [28].

### 1.4.2 Publications

The thesis presented here also includes scientific publications developed by Carsten Felix Draschner (see Fig. 1.3). The Ph.D. mentor, Dr. Hajira Jabeen, and the Ph.D. supervisor Prof. Dr. Jens Lehmann, were always available in terms of content and methodology and helped with the further development of the ideas.

- *Conference Papers (Peer-Reviewed):*

1. **Carsten Felix Draschner**, Jens Lehmann and Hajira Jabeen, *DistSim - Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs*, 2021 IEEE 15th International Conference on Semantic Computing (ICSC), 2021, pp. 333-336, DOI: 10.1109/ICSC50631.2021.00062. <https://ieeexplore.ieee.org/document/9364473>
2. Farshad B. Moghaddam, **Carsten Felix Draschner**, Jens Lehmann and Hajira Jabeen, *Literal2Feature: An automatic scalable RDF graph feature extractor*, in: Further with Knowledge Graphs, International Conference on Semantic Systems, (SEMANTICS), IOS Press, 2021, pp. 74–88. DOI:10.3233/SSW210036, <https://ebooks.iospress.nl/volumearticle/57407>, The publication Literal2Feature is the joint effort of Farshad Bakhshandegan Moghaddam (FBM, another Ph.D. Student at the University of Bonn and member of SDA) and Carsten F. Draschner (CFD) to develop a scalable graph feature extractor. The idea and the methodology were developed in a joint effort. CFD did the code prototyping in python, and the same for Java was done by FBM. The final implementation and embedding into the SANSA stack using Scala and Apache Spark was done by CFD, while FBM drove the evaluation of the server-side experiments.
3. **Carsten Felix Draschner**, Claus Stadler, Farshad B. Moghaddam, Jens Lehmann, and Hajira Jabeen. 2021. *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. DOI:10.1145/3459637.3481999, <https://dl.acm.org/doi/10.1145/3459637.3481999>, The DistRDF2ML paper has been developed mainly by Carsten Felix Draschner (CFD) with the following exceptions. The second author Claus Stadler (CS, former Ph.D. student at the University of Leipzig, and part of the SDA), also supported adjustments and fixes of bugs in the SPARQLIFY [17] framework so that it could be used for the execution of SPARQL queries. Farshad Bakhshandegan Moghaddam (FBM) assisted in the initial execution of the first experiments based on the developments of the scientific publications Literal2Feature [23]. All other parts were developed and written by CFD.
4. **Carsten Felix Draschner**, Hajira Jabeen, Jens Lehmann, "*SimE4KG: Distributed Explainable multi-modal Semantic Similarity Estimation for Knowledge Graphs*", 2022 IEEE Fifth

International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pp. 1-8, DOI: 10.1109/AIKE55402.2022.00007, <https://ieeexplore.ieee.org/document/9939143>, This published paper got additionally invited by the IJSC Journal and is submitted and under review at the point of thesis submission.

5. **Carsten Felix Draschner**, Hajira Jabeen, Jens Lehmann, "*Ethical and Sustainability considerations for Knowledge Graphs based Machine Learning*", 2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pp. 53-60, DOI: 10.1109/AIKE55402.2022.00015, <https://ieeexplore.ieee.org/document/9939282>

- *Workshops, Demos, and Doctoral Consortium (Peer-Reviewed):*

6. **Carsten Felix Draschner**, Farshad B. 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>, This publication was co-authored with Farshad B. Moghaddam, with large parts developed by the main author Carsten Felix Draschner. At the same time, Farshad B. Moghaddam assisted with troubleshooting the Databricks platform and deploying the SANSA jars.
7. Farshad B. 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>, In this paper, an idea for REST interaction with SANSA was jointly developed, with significant parts of the implementation done by Farshad B. Moghaddam. Carsten Felix Draschner developed the sample ML pipelines.

### 1.4.3 Thesis Outline

This thesis consists of ten chapters. Chapter 1 provides a basic overview of the scientific fields, problems, and challenges. It also presents the research questions and the corresponding contributions and associated papers. We introduce reoccurring terms in Chapter 2. These include Semantic Web technologies, ML concepts, and Ethical AI dimensions. Chapter 3 introduces the related literature. These mainly include scalable semantic data analytics like semantic similarity estimation, KG-based downstream ML Pipelines, and Ethical AI approaches. Chapter 4 introduces the DistSim approach, a distributed in-memory semantic similarity estimation approach for native RDF KGs. Chapter 5 then introduces Literal2Feature which is an automated feature extracting SPARQL query generation approach. Chapter 6 shows how explainable downstream ML pipelines can be created using the newly available DistRDF2ML resource. The new software modules SparqlFrame, SmartVectorAssembler, the SANSA stack, and Apache Spark MLlib allow the development of end-to-end ML approaches. Chapter 7 presents Sime4KG, an Explainable Distributed multi-modal Semantic Similarity Estimation for KGs. In particular, the multi-modal feature types in RDF KG literals are discussed, and the results are made explainable and reproducible by semantification. In Chapter 8, we present technical details and approaches for accessible notebooks, and REST interfaces. In Chapter 9, we present ethical and sustainability considerations when developing KG-based ML approaches. Finally, in Chapter 10, we present a summary of this thesis on Scalable Distributed Machine Learning for Knowledge Graphs. There we also provide an outlook on future work.

# Preliminaries

---

In this chapter, we describe the meaning of reoccurring terms. We start with Knowledge Graph and Semantic Web terms. In the second section, we introduce terms and technologies according to technical details in distributed processing. Third, we explain the meaning of Artificial Intelligence (AI) and Machine Learning (ML). Finally, we introduce the AI meta-level dimensions of AI Ethics and Sustainable AI.

## 2.1 Knowledge Graphs, RDF and Semantic Analytics

In this section, we introduce the terms associated with Knowledge Graphs (KG) and Semantic Processing, like KGs, the Resource Description Framework (RDF) standardizing linked data expressions, SPARQL as RDF data corresponding query language, and Apache Jena as Java framework for semantic data processing.

### 2.1.1 Knowledge Graphs (KG)

KGs are a way to represent information about the state of knowledge and facts. A central focus in Linked Data [29] or Semantic Data is on the relationships of information to each other. However, there is not a unique central definition of KGs, but the work of Paulheim [30] includes a definition that also fits the KG term used across this whole thesis:

- 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.

Two types of KGs have become particularly popular. One is the property KG, and the other is the RDF KG. In contrast to Property KGs, RDF KGs always contain standardized information triples. In contrast, additional information can be assigned to relationships and objects in the property KG. Through the development of RDF\* [31], however, RDF KGs are capable of comparable data representation. The basic idea is that two entities, like a person and their employer, are related and can

be linked. Here the person is a node, just like the employer. A directed edge connects both, labeled with "employed\_by". Conceivable would also be the inverse edge "has\_employees" (see Fig. 2.1, top right). In Property KGs as well as RDF\* KGs, there is the possibility to add the start- and end-date of the employment to this edge (see Fig. 2.1, right).

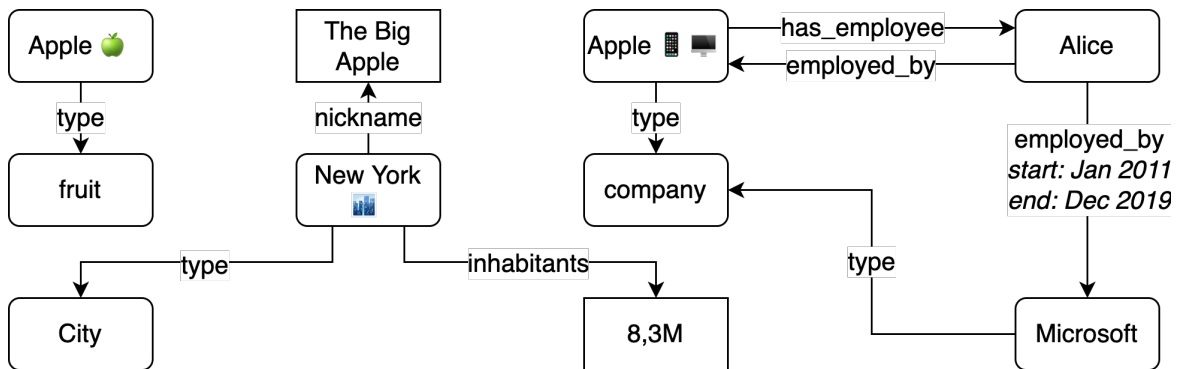


Figure 2.1: KG example with similar naming, inverse relations, and relation with time property

### 2.1.2 Resource Description Framework (RDF)

RDF is the W3C standard for representing semantic data and KGs<sup>1</sup>. Tim Berners Lee developed the semantic web vision [1], which describes the idea of connecting any information on the internet and makes it interpretable as a KG. Unique links act as unified resource identifiers (URIs). Internationalized Resource Identifiers (IRIs) extend the URIs, which are limited to ASCII characters, to the entire character space of the Unicode Character Set. A unique entity should be assigned precisely one URI, and nicknames or alternative names are noted, but the information is always assigned to this entity node [2]. One mathematical definition of an RDF Graph is based on Perez [32]:

- Let  $N, B, L$  be disjoint infinite sets of URIs, blank nodes, and literals. An RDF KG  $G$  is a directed labeled multigraph  $G = (N, E, \Sigma, L)$
- $N \subset (U, B, L)$  a finite set of RDF terms that correspond to nodes
- $E \subseteq N \times N$  a finite set of edges that connect RDF terms
- $\Sigma \subset U$  a set of labels uniquely identified with URIs
- $L : E \mapsto 2^\Sigma$  that maps edges to sets of labels

For example, *Apple*, *Apple*, and *The Big Apple* should be mapped by three different URIs because these can map to the fruit, the company, and the US city (see Fig. 2.1, top left). *The Big Apple* is only the alternative nickname for New York City. All of this RDF KG data is encoded in RDF triples. These consist of a *Subject*, a *Predicate*, and an *Object*. *Subject* and *Predicate* are unique URI or *Blank Nodes*, and the *Object* can be either a URI, *Blank Node*, or a *literal*. *Literals* are not entities but simple values. These values can be multi-modal data like texts, numbers, or images. Also, the *Subject*,

<sup>1</sup> <https://www.w3.org/RDF/>

*Predicate*, and *Object* can be occupied by blank nodes. The *Blank Node* acts like an unspecified URI. A short example RDF file is presented in Listing 2.1, and The graphical representation of the RDF file is presented in Figure 2.2.

```

1 <http://one.example/subject1> <http://one.example/predicate1> <
  http://one.example/object1> .
2 _:subject1 <http://an.example/predicate1> "object1" .
3 _:subject2 <http://an.example/predicate2> "object2" .

```

Listing 2.1: RDF file example based on triple patterns based on W3C example [33]

Which types of entities can be related to each other and by which standardization the triples should be constructed is determined by OWL ontologies<sup>2</sup>. OWL ontologies are defined explicitly for various data and information domains. Using ontologies, the created RDF KG is standardized and can be merged and integrated better with another KG. Ontologies can also describe additional information, such as entity classification hierarchies [34].

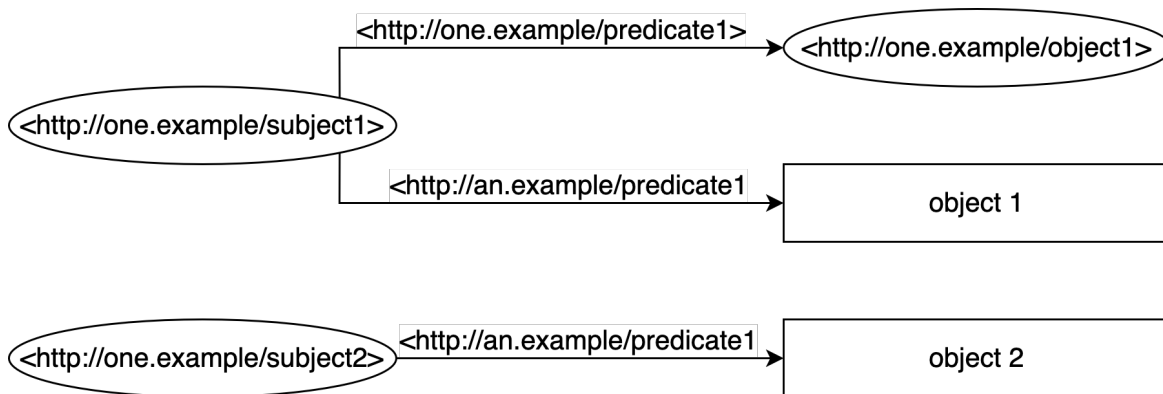


Figure 2.2: Example RDF Graph correspond to file presented in Listing 2.2

### 2.1.3 Ontology

An ontology enables people and applications to map domain information by a unified specification [35, 36]. Information with unified data structures can better interact with each other, especially in technical applications. Ontologies are also created for specific information domains [37]. In the context of structured data, ontologies play the role for RDF data that schemas do for relational data. Another parallel to RDF data is that ontologies map their information into triple data. These definitions include the possible property relations of entities and also class hierarchies. In addition, axioms define the structurally possible relation characteristics of properties such as equivalence, symmetry, inversion, and transitive relations [38, 39]. Unified ontologies create the basis for cooperative linked data sharing within the community that drives the exchange of RDF data. This standardization of linked data and data sharing reduces data silos. This crowd-distributed approach of unified exchange through linked data structures is consistent with the characteristics of the Semantic Web and Linked Open Data [40] (LOD). The definition of ontology based on Guarino et al. [35] is:

<sup>2</sup> <https://www.w3.org/OWL/>

- Let  $C$  be a conceptualization, and  $L$  a logical language with vocabulary  $V$  and ontological commitment  $K$ .
- L an ontology  $O_K$  for  $C$  with vocabulary  $V$  and ontological commitment  $K$  is a logical theory consisting of a set of formulas of  $L$ , designed so that the set of its models approximates as well as possible the set of intended models of  $L$  according to  $K$ .

#### 2.1.4 SPARQL Query Language

SPARQL is the acronym for SPARQL Protocol and RDF Query Language. SPARQL is the W3C recommended standard for RDF KG Queries<sup>3</sup>, corresponding to SQL as the query language for tabular data and relational databases. The queries describe graph patterns. The queries are majorly built by triple patterns. These SPARQL queries set fixed sub-graph structures and allow the flexibility to fit the data via variables. Corresponding to the RDF triple, a triple pattern in a SPARQL query consists of placeholders, i.e., variables, at positions of Subject, Predicate, and Object (or multiple of the same time, even all). Multiple of these triple patterns can be combined for expressive SPARQL queries. This combination of triple patterns is called Basic Graph Pattern (BGP). The definition of a BGP is [34]:

- Let  $U, B, L$  be a disjoint set of URIs, blank nodes, and literals.
- A set  $V$  be variables disjoint from RDF Graph nodes:  $V \cap (U \cup B \cup L) = \emptyset$
- Then a triple pattern  $t$  is a member of the set  $(U \cup V) \times (U \cup V) \times (U \cup L \cup V)$
- These triple patterns can be combined into a BGP with logic operations like:  $t_1$  AND  $t_2$  AND  $t_3$

In particular, projection variables are then specified for explicit returned data. SPARQL also supports operators that are similar to the SQL syntax. These include *SELECT*, *WHERE* for selection and projection, *GROUP BY* for grouping, *OPTIONAL* for allowing optional information, *AVG*, *SUM*, and *MEAN* as mathematical operators, and *isURI* and *isLiteral* for the dedicated RDF data specification retrieval and filtering. This list is not complete but should show the options of SPARQL. These operations form the operators on the BGPs, forming the dedicated SPARQL Queries. In Listing 2.2, we show a sample SPARQL Query for the RDF KG presented in Figure 2.2. This SPARQL Query should result in respective subjects with the associated object literals.

---

```
1 SELECT ?s ?o
2 WHERE {
3   ?s <http://an.example/predicate1> ?o
4 }
```

---

Listing 2.2: Sample SPARQL query corresponding to RDF graph in Fig. 2.2

---

<sup>3</sup> <https://www.w3.org/TR/rdf-sparql-query/>

### 2.1.5 Apache Jena

Apache Jena is an open Apache framework developed in Java<sup>4</sup>. It supports the development and use of Linked Data, RDF data, and semantic web programs. It is possible to operate natively on KG data through many functions. This includes the creation of entities, literals, and triple data [15]. Also, the filter functions known from SPARQL, like *isURI* and *isLiteral*, are supported. So it is possible to work with KG data in classical Java programs. Listing 2.3 presents the usage of Apache Jena code through Scala.

## 2.2 Scalable and Distributed Data Analytics

KG dataset sizes can reach extensive volumes. A variety of graph data analytics can also rely on algorithms that exhibit non-linear computational complexities. NP-hard or even polynomial algorithms can hardly be executed on the data within acceptable processing runtimes, even with the high-performant resources. However, considerable processing power is already required for the simple representation of the data, which adds to the processing power of algorithms with high computational complexity. In the context of the work presented here, we, therefore, focus on both scalable data analytics and distributed data analytics. Scalable data analytics involves developing and using algorithms that exhibit a lower computational complexity and are thus more likely to be feasible within the required processing runtimes despite large amounts of data. To this end, existing non-linear algorithms are replaced by approximating, probabilistic, or optimized variants that deliver precise results within an expected accuracy deviation in a much shorter time. In addition, we make sure that the scalable algorithms can be parallelized in the best case. Thus, several threads of a computer or even several computers can solve the task simultaneously in a distributed manner. For example, we use a MinHash-based Locality Sensitivity Similarity Estimation approach for the at least quadratic scaling *all pair similarity* (APS). On the other hand, we use distributed computing to provide a sufficient amount of memory and have more processing power available through accumulated cores.

### 2.2.1 Distributed Computing

The data and algorithms in ML and data analytics can take up considerable resources. These are the hard-drive resources for managing the data volumes and the RAM resources since, for ML, much of the extensive data must be accessible for processing. Also, the task can be very complex, so significant processing resources are required. For a speedup or the possibility of providing these resources, it may be necessary to use several computers in a cluster. Single computers cannot be increased arbitrarily in performance because, at a certain point, the hardware becomes over-proportionally expensive or is not available in sufficient size. A connection of several computers as nodes in a cluster can accumulate hardware performance. Developed frameworks and software support managing distributed data and computation [14].

---

<sup>4</sup> <https://jena.apache.org>

---

```

1 # create some literal and operate on it
2 val l: Literal = model.createLiteral(25)
3 val age: int = l.getInt()
4 val isLit: bool = l.isLiteral()
5 # create a blank node
6 val bn: Node = NodeFactory.createBlankNode("someHash")
7 # create URI
8 val someUri: Node = NodeFactory.createURI("someUri")
9 # create triple
10 val someTriple: Triple = Triple.create(bn, someUri, l)
11 # get information out of triple
12 val subject: Node = someTriple.getSubject()

```

---

Listing 2.3: Sample Apache Jena Code written in Scala programming language

## 2.2.2 Hadoop Distributed File System (HDFS)

With Apache Hadoop, a framework for distributed processing and storage is optimized for large-scale data on cluster computation. Part of the functionality is fault tolerance and high availability, even on usual consumer hardware. Several other tools have been developed based on this. The Hadoop File System (HDFS) is one of the core components of distributed handling of large-scale datasets. HDFS includes data partitioning and fault tolerance techniques and minimizes data movement and duplication. Fault tolerance is enabled by splitting data into blocks and distributing replicated blocks across the system's nodes. The HDFS architecture follows a driver/worker model. The namenode is the driver managing all operational steps like the namespace, replicates, and metadata and keeps track of the processed operations. For the namenode, a fallback passive namenode is initiated, which can overtake drivers' work in case of failure for faster recovery. The workers are datanodes allowing access and storage of data [41].

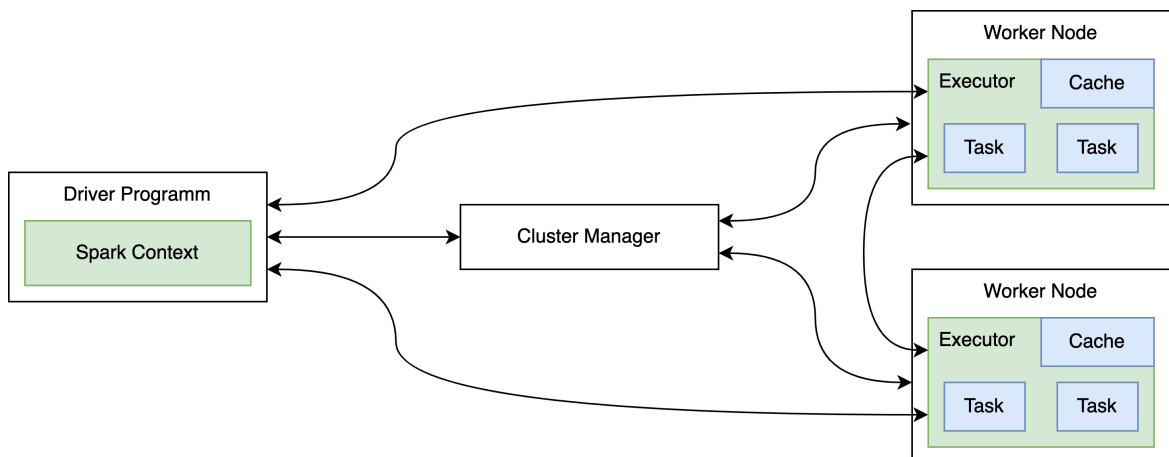


Figure 2.3: Apache Spark Components



### 2.2.3 Apache Spark

Apache Spark (Spark) is a framework for cluster computing [42]. Apache Spark extends the concepts of MapReduce [43] of scalable distributed and fault-tolerant data processing. It is available under the Apache Open Source License<sup>5</sup>. Apache Spark encapsulates software modules that optimize the execution of big data analytics pipelines. These pipelines can be executed in distributed and in-memory on Spark clusters. Apache Spark follows a driver/worker architecture [13]. A central Spark Session hosted on the driver node coordinates tasks distributed across worker nodes (see Fig. 2.3). The workers retrieve needed data from HDFS systems. They operate in locally cached in-memory partitions, which improve performance in iterative processes over the same data before storing it back to HDFS drives. Apache Spark operates on RDD structures which are fault-tolerant immutable collections of records. In addition, Apache Spark was extended to also work natively on DataFrames aligned to the success of python data analytics pipelines with scikit-learn, pandas, and more. Apache Spark handles the splitting and partitioning of Datasets and RDDs. Apache Spark provides several libraries and APIs for data processing. Structured querying is supported by *SparkSQL*, streaming of data is supported by *Spark Streaming*, and Graph Parallel Processing can be computed by *GraphX*. Distributed ML is usable through *MLlib* [42].

**Apache Spark MLlib** Apache Spark MLlib [44] adapts the ideas of the well-known python library scikit-learn<sup>6</sup> to provide standard interfaced transformers that allow high modular and generic ML pipeline construction [16]. They operate on DataFrames, which are tabular-typed representations of data. The advantage of Apache Spark MLlib compared to libraries like scikit-learn is that it can be utilized in distributed processing environments. Apache Spark is natively implemented in Scala and also supports Python and Java.

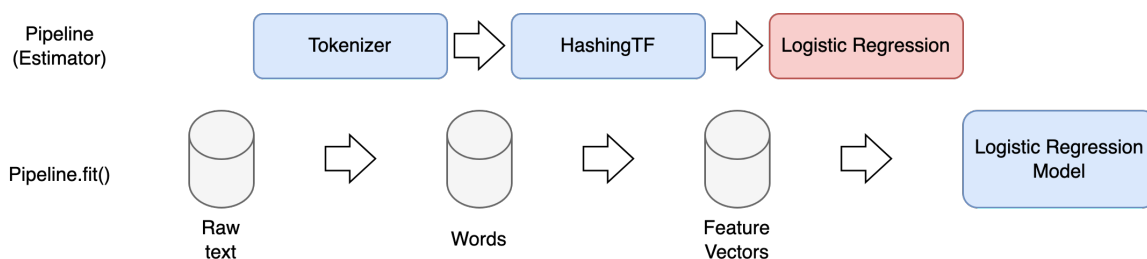
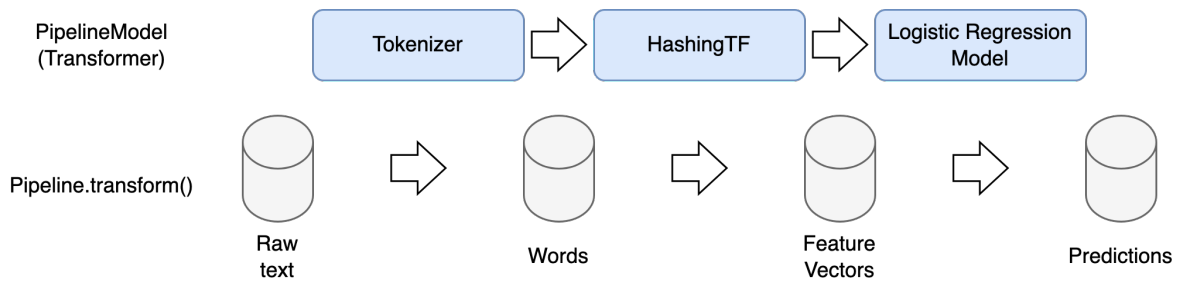


Figure 2.4: Apache Spark MLlib pipeline *fit*

Apache Spark MLlib approaches are created within *pipelines*. These ML pipelines consist of multiple steps, each being either an Estimator or a Transformer. Transformers can be executed directly on DataFrames, transforming the data into the desired novel Dataframe. Estimators need to be trained or adjusted with the fit function to become a Transformer. In Figure 2.4, the blue stages are already Transformers. At the same time, the ML model Logistic Regression (red) needs to be fitted to become a Logistic Regression models, which is then a Transformer itself [45]. Transformers and estimators fitted into a Transformer model can be straightforwardly executed on data. Corresponding example code can be found in Listing 2.4, which is based on Apache Spark MLlib tutorials [44].

<sup>5</sup> <https://www.apache.org/licenses/LICENSE-2.0>

<sup>6</sup> <https://scikit-learn.org>

Figure 2.5: Apache Spark MLlib pipeline *transform*

## 2.2.4 Scala

Scala is a functional, object-oriented, and imperative programming language. It was created by Martin Odersky and was released in 2004 [46]. Scala uses static types to reduce bugs in complex applications<sup>7</sup>. It is also possible to use Java libraries as Scala is being executed within the Java Virtual Machine (JVM) [46]. Common IDEs for Scala are IntelliJ, NetBeans, and Eclipse.

---

```

1 // DFs have columns "id", "text", "label"
2 val train: DataFrame = [...]
3 val test: DataFrame = [...]
4
5 val tokenizer = new Tokenizer()
6   .setInputCol("text")
7   .setOutputCol("words")
8 val hashingTF = new HashingTF()
9   .setNumFeatures(1000)
10  .setInputCol(tokenizer.getOutputCol)
11  .setOutputCol("features")
12 val lr = new LogisticRegression()
13  .setMaxIter(10)
14  .setRegParam(0.001)
15 val pipeline = new Pipeline()
16  .setStages(Array(tokenizer, hashingTF, lr))
17 val resultDF: DataFrame = model.transform(test)

```

---

Listing 2.4: Scala example code for Apache Spark MLlib pipeline

## 2.3 Artificial Intelligence and Machine Learning

This section introduces the terms Artificial Intelligence (AI) and Machine Learning (ML). First, we introduce Artificial Intelligence which is intelligence powered by machines as an alternative to animal/human/natural intelligence. Based on this definition, we will introduce ML, a technique to reach AI through a robust mathematical-statistical approach currently dominating this field.

---

<sup>7</sup> <https://www.scala-lang.org>

### 2.3.1 Artificial Intelligence (AI)

Artificial Intelligence (AI) is the theory and development of computer systems capable of performing tasks that typically require human intelligence. These include visual perception, speech recognition, decision-making, and translation between languages. Various mathematical, algorithmic, and technical approaches are used to implement AI, including statistics, evolutionary algorithms, and ML [47]. Artificial Intelligence already has a daily presence in our lives through Conversational AI/understanding of human speech (Alexa, Cortana, Siri, Google), self-driving cars (Tesla), Recommendation Systems (Youtube, Amazon, Netflix, Twitter, Facebook), web search engines (Google, Microsoft), and natural language translation (DeepL, Google).

### 2.3.2 Machine Learning (ML)

Machine Learning (ML) describes the ability of computers to learn and adapt without following explicit instructions. This is achieved by statistical models that analyze patterns in data and infer them [48]. Within ML, a distinction is made between supervised and unsupervised ML. The difference is how training data with accurate labels are available and used. True labels are training data values that the ML approach should predict desirably. This label can be, for example, a continuous numerical value in regression problems or a discrete value that specifies a sample classification. In the training of supervised ML models, a subset is selected from the available labeled data. The ML model is optimized under a mathematical condition that minimizes the difference between the predicted and actual labels. The remaining labeled part of the data is used for testing and validating the model. This ensures that the measured performance is based on something other than already known and, thus, memorized samples. ML is applied to arbitrary data sources like images and video for object recognition, human written texts for Natural Language Processing and Understanding, linked social network data structures for a recommendation, or classical tabular data. In most cases, ML needs numeric tensor representation for training feature sets and labeling. As not all of these data sources have a dedicated fixed length tensor representation, preprocessing steps for the overall ML pipeline are needed to transform initial data into a fitting representation of the respective ML model (see Figs. 2.4, 2.5 and List. 2.4). Often used ML algorithms are Decision Trees and Forests, Support Vector Machines, XG-Boost, Multi-Layer-Perceptrons, or even the more Complex Artificial Neural Networks with more hidden layers called Deep Neural Networks. A broad range of libraries have been developed and are openly accessible. For common scientific python projects, scikit-learn<sup>8</sup> provides already a set of classical ML approaches, while tensorflow<sup>9</sup> through keras<sup>10</sup> and pytorch<sup>11</sup> are libraries for deep neural network creation and processing. The library landscape is vastly changing, and novel tools, libraries, and frameworks are established, simplifying ML creation or improving performance and accuracy.

---

<sup>8</sup> <https://scikit-learn.org/stable/>

<sup>9</sup> <https://www.tensorflow.org>

<sup>10</sup> <https://keras.io>

<sup>11</sup> <https://pytorch.org>

## 2.4 AI Ethics and Sustainable AI

In this section, we introduce the terms AI Ethics and Sustainable AI. AI Ethics describes the efforts to reflect, measure, and optimize AI's impact on social structures, discrimination, trust, and ethical considerations. Sustainable AI describes the efforts to reduce ML research and development lifecycle's negative environmental footprint, primarily measured in CO<sub>2</sub> emissions based on electric energy consumption.

### 2.4.1 AI Ethics

AI Ethics (or Ethical AI) describes the efforts to investigate ethical and moral aspects of AI-driven processes [49, 50]. As AI-driven processes can impact our daily lives and might perform autonomous decisions based on pre-trained models, those can lead to unethical, discriminating situations [50]. Bad AI behavior can be due to various reasons, like data bias or model failure. Especially complex neural networks are perceived as black-box systems. Through the effort of Explainable AI, the models are leveraged to create reasonable and explainable results[50, 51]. A typical example of Ethical AI comes from the already-known trolley dilemma. In this dilemma, a trolley or a train can no longer be stopped, and a decision is made about which group of people the train will hit and damage. The scenarios describe one vs. three people or one child vs. two seniors. These extreme situations should establish moral awareness, which might also be made [49], e.g., in self-driving cars in a brake failure accident situation. This car will be driven by an AI and thus need guidance for those edge case decision-making processes. Besides the self-driving car scenario, many existing and foreseeable situations might involve AI in ethics-related processing [52, 53] also based on KG data.

### 2.4.2 Sustainable AI

Sustainable AI includes optimizations of AI/ML methods that minimize the resource consumption of ML algorithms and, if necessary, also justify it concerning the purpose [51, 54, 55]. In contrast to Sustainable AI, AI for Sustainability are approaches that use AI to drive processes that lead to a more sustainable world [54]. Primary importance in the context of Sustainable AI is the usage of resources. ML approaches and their training can have vast needs of processing power correlating with the demand for electric energy. Electric energy production can have various sources with different CO<sub>2</sub> footprints. Same time, the hardware processing time can be scheduled over different day periods reflecting their renewable energy production footprint. Purchasing new hardware can be motivated if novel systems efficiency motivates the replacement. Efficiency is often measured by processing power (Floating point operations FLOPS) compared to needed electric energy (Watt). As Artificial Intelligence is highly dependent on parameters and hyper-parameters, the exploration and optimization of models can also significantly impact the ML model footprint.

---

## Related Work

---

This chapter presents the versatile related work that formed the foundation for this thesis. Section 3.1 shows relevant techniques for machine learning (ML) pipelines on knowledge graph (KG) data. In particular, we focus on the possibility of extracting fixed-length numeric feature vectors from KG data. Section 3.2 summarizes related work of semantic similarity estimation on KG data. In semantic similarity estimation, pairs of KG entities in the graph are assigned a continuous value representing the similarity. Different approaches using the data and structure enable the computation, which can also be found in optimized probabilistic approaches and implemented libraries and frameworks. In the following Section 3.3, we show which approaches have been created to keep up with the increasingly complex datasets of vast KG and to be still able to execute ML with the datasets through scalable semantic analytics frameworks. Finally, in Section 3.4, we address the KG ML meta dimensions, such as reproducibility being strengthened by semantification. We also discuss aspects such as explainability, AI ethics, and sustainable AI in the context of ML on KGs. Section 3.5 provides the summary of related work.

This chapter is based on the related work sections of the following publications [22–24, 27, 28]. Further details can be found in Section 1.4, especially in Section 1.4.2:

- **Carsten Felix Draschner**, Jens Lehmann and Hajira Jabeen, *DistSim - Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs*, 2021 IEEE 15th International Conference on Semantic Computing (ICSC), 2021
- Farshad Bakhshandegan Moghaddam, **Carsten Felix Draschner**, Jens Lehmann and Hajira Jabeen, *Literal2Feature - An automatic scalable RDF graph feature extractor*, Further with Knowledge Graphs (SEMANTICS), 2021
- **Carsten Felix Draschner**, Claus Stadler, Farshad Bakhshandegan Moghaddam, Jens Lehmann, and Hajira Jabeen. 2021. *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, 2021
- **Carsten Felix Draschner**, Hajira Jabeen, Jens Lehmann, "*SimE4KG - Distributed Explainable multi-modal Semantic Similarity Estimation for Knowledge Graphs*", IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2022

- **Carsten Felix Draschner**, Hajira Jabeen, Jens Lehmann, "*Ethical and Sustainability considerations for Knowledge Graphs based Machine Learning*", IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2022

## 3.1 Knowledge Graph data in downstream ML Pipelines

This section describes existing scientific work that uses ML and data analytics on KG data. We introduce several opportunities to extract fixed-length numeric feature tensors from the multi-modal KG data.

### 3.1.1 KG Feature Tensors

Most current ML methods expect fixed-length numeric feature tensors or vectors representing the samples' information [16, 56]. Since a sample node in a KG can be linked to any number of other nodes, and these nodes can also have any additional information attached, no method extracts a fixed set of information that fully represents the values and the structure of the sample [11, 57]. Approaches have been developed that approximate the representation of sample information. We present several approaches that use graph queries [17, 58, 59], propositionalization [60], graph kernels [61–63], and KG embeddings [10, 64, 65] to create these feature vectors (see Sect. 3.1.2 - 3.1.4).

*Example:* KGs can contain much data for one sample node. A classic use case of KG is social media [66]. A sample node representing a *person* can have structural information and multi-modal values. For a URI representing a *person*, there are then arbitrarily complex *friend-of* relationships to other URIs in the graph. Likewise, there can be relationships between multi-media content and, for example, *genres* [6]. Depending on the social media platform, incoming and outgoing *followers* are also tracked. All this information is structural. It can reflect influence, interests, trends, and much more. Same time, data such as *date of birth*, *geolocation*, *time* of content creation, a series of images or videos, and texts in any language are also available. Enriching images and posts with tags and metadata increases abstraction levels of data structure [67]. All this information results in comprehensive and complex data structures that are difficult to map in a fixed number of purely numerical feature vectors.

### 3.1.2 Query-based Feature Extraction and Propositionalization

Several Data Science and ML approaches start on tabular data encoded in DataFrames [16, 20]. For the ML approaches on tabular data, comparable data can be obtained by feature extraction on KGs. Query-based feature extraction for the edges, nodes, and literals in RDF KGs and its transformation into binary and numerical features is called propositionalization [68]. This feature extraction into tabular data can be created by specifying SPARQL queries (see Chapter 2 Section 2.1.4). One of these approaches is LiDDM [58], which provides a user interface (UI) assisted opportunity to extract features suitable for ML algorithms by specifying a SPARQL query. A more automated approach for the feature extraction based on constructed SPARQL queries was presented by Cheng et al. [59]. Several tools and plugins have been developed for data mining and ML platforms like Rapid Miner. One example is the *semweb-rapidminer* plugin [60] supporting the native use of RDF KGs in ML pipelines. This approach also uses classical SPARQL-based feature extraction but uses it directly within an ML environment. As an alternative to this, FeGeLOD [69] and its successor *Rapidminer*

*Linked Open Data Extension* were developed [68] which explore the feature space of KGs and assist in the unsupervised creation of vectors for downstream ML. Several alternative opportunities for feature vector creation are provided and can be selected by the user.

In the context of this thesis, we develop in Chapters 4-7 several different novel approaches and open-source software modules that transform KG data into fixed-length numeric feature vectors [22–24, 27]. Different alternative user-configurable options are offered to adapt both the complexity of the features and the processing time. The modules are developed to be integrated into the widely used Apache Spark MLlib pipelines [16]. In Chapter 8, we propose further interaction opportunities with these software modules through Platform as a Service (PaaS) utilizing notebooks and within a containerized environment that offers the functionalities through REST APIs. Finally, in Chapter 9, we show the effect of propositionalized features from KGs in ML pipelines from an AI Ethics perspective.

### 3.1.3 Graph Kernels

The paper "Weisfeiler-Lehmann graph kernels" [64] presents how graph kernels can be used to extract feature vectors for classical data analytics, similarity estimation, or ML approaches. In the complexity analysis, however, it is presented that these methods have difficulties operating on the far more complex directed labeled multi-graph KGs with millions of nodes and billions of edges. However, subsequent work such as Deepwalk [62] introduces simplified approximative approaches that make the extraction of subgraphs possible even in the context of KGs. The methods of Word2Vec [70] are further used in Node2Vec [61], which interprets the subgraphs into an entity like a document of sentences. Deepwalk and Node2Vec are tailored for graphs with one type of edges. The subgraphs correspond to sentences and the collection of random walks for one entity in the document. This problem transformation allows the creation of fixed-length numeric feature vectors like in NLP. A dedicated implementation for RDF KGs is presented by the RDF2VEC [63] approach, which can operate natively on Semantic Web KGs. It also can handle the directed labeled edges of common KGs.

We use the concept of graph kernels in Chapter 4 as a high-performance method to extract features. The depth of the kernels was lowered to be efficiently implemented by Apache Spark's map-reduce [22]. In Chapter 5, 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]. In Chapter 7, we extend the concept from DistSim [22] to address the multi-modal nature of KG data better, creating a tradeoff between the highly efficient method from DistSim and the precise but processing-intensive feature extraction from Chapter 6 DistRDF2ML's feature extraction [24, 27].

### 3.1.4 Knowledge Graph Embedding (KGE)

Another option to produce fixed-length numeric feature vectors is Knowledge Graph Embeddings (KGE). Mathematical methods like Tensor Factorization or Stochastic Gradient Descent (SGD) optimize latent embeddings until they meet the optimization criteria [10, 65]. On the one hand, there are tensor factorization options. Here the adjacency matrix of the KG entities is decomposed so that the tensor factorizations are suitable as feature vectors. Examples of this are DistMult [65] and Simple [65]. On the other hand, there are also KGE that represent all entities and relations in random fixed-length vectors. These vectors are latent embeddings adjusted until the existing triple structure is approximated as near as possible in the vector space. The initial approach TransE [10] optimizes

the vectors until  $h + r = t$ . Here  $h$  stands for the *heading* subject of the triple,  $r$  for the *relation* latent embedding representation, and  $t$  for the *tailing* object of the triple. In order to avoid that the optimal solution is the zero vector for all relations and entities, techniques such as negative sampling are used [10, 71]. Further latent embedding procedures, which optimize by SGD mathematical optimization for various criteria, have been developed. These alternative embedding approaches deal better with inverse or cyclic relations. These methods make it impossible to say why a particular value is part of the feature vector. A different random KGE initialization would have resulted in entirely different feature vectors. Also, it is difficult to define new embeddings for not yet seen entities or relations.

We benchmarked against latent embeddings in Chapter 5 as part of the evaluation of our novel approach. Furthermore, we have applied Word2Vec [70] latent embeddings for natural language strings as a propositionalization strategy in the Chapter 6 [24].

### 3.2 Semantic Similarity Estimation

Semantic similarity estimation is a subset of semantic analytics and learning on KGs. Here, subgraphs or nodes representing entities are assigned values in pairs that indicate their similarity. Similarity and relatedness should be distinguished from each other. Similarity describes the concept that one entity can be conceptually replaced by another entity [72]. Relatedness describes the relationship that things have with each other. For example, a bicycle and a car are both vehicles and somehow similar, but gasoline is related to cars. Semantic similarity is present in language analysis and the definition of synonyms [73], but it is also used on structured data such as KGs [74]. Various approaches and technologies have been developed to compute semantic similarity [75], which are also provided as usable tools. These semantic similarity estimations can then be applied in the most diverse KG data domains, such as in the field of music recommendation systems or the bio-medical domain. In the example of music-based recommendation systems, similarity estimation can suggest suitable alternatives to songs that have already been preferred. In our Chapters 4 and 7, we introduce Semantic Similarity approaches for RDF KGs.

#### 3.2.1 Similarity Estimation Families for Knowledge Graphs

Different approaches for the computation of semantic similarity estimations have been developed for structured linked data like Semantic Web [1] data or KGs. These follow different approaches assigning similarity values on a graph data basis [76, 77]. The most common approaches can be divided into the following four categories:

- Structure and Path
- Information Content
- Feature-Based
- Vectors and Latent Embeddings

There are also more advanced approaches that combine several of these families into so-called hybrid models [78].



**Structure and Path-based Semantic Similarity Estimation** Structure and path-based estimators calculate semantic similarity by determining the distance between two entities in the graph [79] and relating this to the overall distribution of distances in the graph. This group of similarity estimations includes approaches such as multiple shortest paths or particularly dense links [80, 81]. As processing shortest path search for all entity pairs in large-scale KGs is not scalable, we decided not to use this group of similarity estimators.

*Example:* In a KG that maps the articles of scientific publications, the content and thematic similarity can be measured by the distance of references and hyperlinks from one article to another. More topic-similar articles should have a higher likelihood to be more densely linked and have a smaller graph distance to each other, especially when considering multiple shortest paths.

**Information Content-based Semantic Similarity Estimation** Information content-based semantic similarity determines the similarity of two entities in the graph based on their common properties [82, 83]. The information content of the common properties indicates how individual this property is. The calculation is related to the TF-IDF calculation in NLP data analytics. Increasing weight on similar features that are more expressive can support the accuracy of assigned similarity values. In Chapter 7, we introduce multi-dimensional weighting techniques, which also fetch the idea of feature expressivity.

*Example:* In the example of a movie database, the information content of the description that two entities are of the type movie would not be significant since this property is present across all samples [84]. In contrast, the genre labeling of a sports drama is exceptional in that it is so rare in relative terms that it is a meaningful indicator of high similarity.

**Feature-based Semantic Similarity Estimation** A third family of similarity estimation approaches is the set of feature-based similarity functions. These compare the extent to which two entities have common features and calculate these in relation to the total set of features present. This usually results in an intersection over union (IOU) of the feature sets [76, 85–90]. The application of feature-based semantic similarity estimation plays a fundamental role in our DistSim developments in Chapter 4 and offers one option in the SimE4KG approach presented in Chapter 7.

*Example:* In the context of a music library, playlists are similar if they contain a large number of joint songs divided by the total number of songs they contain.

**Vector and Latent Embedding-based Semantic Similarity Estimation** The fourth group of semantic similarity estimations is based on numeric feature vectors, which can be obtained by query extraction, propositionalization, graph kernels, or KGEs (see Sect. 3.1). The similarity of the two entities is measured by the euclidean distance of the vector representations. Embedding-based similarity estimation is important when structural features should be measured or natural language strings (see Chapter 7) are needed. However, embedding-based similarity estimation in a high-dimensional embedding space might lack explainability.

### 3.2.2 Feature Based Semantic Similarity Estimation

In Section 3.2.1, we presented a feature-based semantic similarity estimation procedure for graphs. These aggregate the available properties of the corresponding entity samples and build feature sets. Over the last decades, several different score functions have been developed. Assume that entity  $A$  has

associated features  $X$  and entity  $B$  has associated features  $Y$ . The resulting scores of these functions can be used as a similarity value. Known feature-based semantic similarity scores applied to graphs for feature sets are shown below:

$$score_{JaccardSimilarity}(A, B) = \frac{|X \cap Y|}{|X \cup Y|} \quad [87]$$

$$score_{SimpsonSimilarity}(A, B) = \frac{|X \cap Y|}{\min\{|X|, |Y|\}} \quad [89]$$

$$score_{DiceSimilarity}(A, B) = \frac{2|X \cap Y|}{|X| + |Y|} \quad [86]$$

$$score_{OchiaiSimilarity}(A, B) = \frac{|X \cap Y|}{\sqrt{|X| * |Y|}} \quad [88]$$

$$score_{TverskySimilarity}(A, B) = \frac{|X \cap Y|}{|X \cap Y| + \alpha |X - Y| + \beta |Y - X|} \quad [90]$$

$$score_{BatetDistance}(A, B) = \log_2 \left( 1 + \frac{|X - Y| + |Y - X|}{|X \cap Y| + |X - Y| + |Y - X|} \right) \quad [76]$$

$$score_{Braun-BlanquetSimilarity}(A, B) = \frac{|X \cap Y|}{\max\{|X|, |Y|\}} \quad [85]$$

The primary structure of the formula is similar across these examples. They have the calculation of intersection over the union (IoU) in common. The various methods differ in weighting possibilities and how the score finally turns out. Especially for KGs, IOU can compute the feature set similarity of associated URIs. If arbitrary literals are included in the feature set, they have to be taken into account sensitively (see Chapter 7).

### 3.2.3 Probabilistic Similarity Estimation

Since all-pair semantic similarity (APS) has at least quadratic complexity just because every pair-wise score must be computed, it is challenging to scale for large graphs. In addition, the computational complexity of one semantic similarity estimation for one pair of entities in a large KG can barely be scalable. With methods such as structure-based similarity estimation like multiple shortest path search or graph kernels, as well as propositionalization, a significant computational complexity arises. Scalable systems distribute data and processing across multiple nodes in the cluster. This data distribution requires a partitioning of the graph. Traversing a distributed cluster in shortest-path search or aggregation of features can require data exchange of multiple nodes. This data exchange is one of the significant bottlenecks in distributed processing. Through probabilistic algorithms, more scalable semantic similarity estimations can be implemented. One probabilistic alternative to the Jaccard Similarity Estimation uses MinHash Locality Sensitivity Hashing [91] to group similar feature sets [92] based on their features-hashes and compute the similarity score across the samples in one bucket. So the overall needed similarity computation is not the full quadratic space but only quadratic for each bucket.

### 3.2.4 Frameworks

One implementation for a scalable semantic similarity estimation using MinHash Locality Sensitivity Hashing (LSH) [91, 92] has been developed as an extension [93] to the distributed data processing stack Apache Spark [13, 94, 95]. The developed API is not designed to handle graph or RDF KG triple data. Other similarity estimation libraries and frameworks specifically for graphs and RDF KGs were developed but were not optimized for scalable computation on large-scale KGs [75].

## 3.3 Scalable Semantic Analytics

Data analytics and ML on large KGs may require distributed computation architectures. These distributed computation systems divide both the data and the processing steps among multiple nodes of a cluster. This section describes the complexity of known KGs and ways to perform data analytics and ML on semantic linked data natively.

### 3.3.1 KG Data Analytics Complexity

KG-based ML and KG-based data analytics have been used for various use cases over the past years. The approaches, algorithms, libraries, and frameworks had to cope with the increasingly large KG datasets [29]. Also, the widely used open available knowledge graphs were often the source for data analytics, the basis for ML approaches, or acted as benchmarking [12, 71, 96]. These datasets contain billions of triples in GB-orders of magnitude. Furthermore, domain-specific datasets Wordnet or LMDB [6, 7] were used for ML approaches. In the area of, e.g., KGE, the reduced KGs datasets FB15k and WN18 [71] are used for benchmarking and evaluation. In particular, scalable techniques like DGL-KE and Pytorch-BigGraph can handle not only the reduced but also the total KG datasets like Freebase [5, 12, 96]. The libraries and frameworks are implemented to function as efficiently as possible but can also process the data analytics and ML on distributed systems. This efficiency and distributed computation implementation require different adaptations depending on the KG-based ML approach [12, 16, 96, 97].

Inspired by the approaches for scalable semantic data analytics frameworks and libraries and the distributed downstream ML frameworks, we identified several research gaps. In particular, we developed frameworks within the Chapters 4 - 7 that can perform ML approaches natively on RDF KG data through scalable distributed approaches. For this purpose, we leverage the strengths of the widely used distributed data processing Apache Spark stack. We use large-scale KGs like DBpedia or LMDB [6, 98] to evaluate our contributions.

### 3.3.2 Distributed KG Processing Frameworks

In data analytics, especially scalable data analytics and ML, a wide variety of data are processed. This includes data in the form of tabulars, images, texts, audio, videos and graph. Tabular data is clearly ordered. Texts have an order of words and a structure in sentences. Images are limited by their resolution and color depth. Audio and video add a time dimension to these structures. Data as graph has the challenge of not having a classic internal order when compared to other data forms. A KG can be arbitrarily large and have arbitrary links and relations for each graph entity. This complex graph structure leads to more difficult processing in distributed environments if a graph is distributed across

multiple systems for processing. A distributed graph across systems reduces the efficiency of graph traversal when an arbitrary node of the graph is requested. Several graph processing frameworks for the relevant core data analytics and processing pipelines have been developed in recent years [99]. Some frameworks enable node-centric graph processing through message passing [100–103]. Other frameworks focus on the distributed creation and optimization of KG embeddings. Finally, some developments extend known distributed data analytics pipelines to operate on graph data.

**Graph Processing** Several software projects adapt the node-centric processing introduced by Pregel [100]. With Pregel, each node in the graph is understood as an acting instance, information flows, and data exchange occurs along the graph edges. In this way, graph analytic problems like PageRank [104] can be determined in a highly efficient manner. Open source distributed software frameworks such as Apache Spark GraphX [102], GraphLab [103], and Apache Giraph [101] take up this processing. Some of these technologies also scale to enterprise KG sizes and associated infrastructure in specific tasks [99, 105]. As we also need literal values in our feature extraction, SPARQL-based and downstream pipelines through scalable frameworks like Apache Spark MLlib are more appropriate for our solutions.

**Knowledge Graph Embedding** Another particularly computationally complex approach is the computation of KG embedding for large-scale KGs. Several libraries can be integrated into classical ML pipelines for KGs and are based on python ML, including PyKeen [71] and Open-KE [106]. However, these do not focus on distributed processing to deal with large KG data. For large data, the frameworks Graphvite [107], Pytorch BigGraph [96] and DGL-KE [12] have evolved. These frameworks use distributed computing to handle the memory load of KGs and the corresponding embeddings. A particular challenge lies in the distributed optimization of KGE. Since KGE tensors are distributed across multiple machines, vectors must be shared and synchronized across optimization cycles. Since KGE lacks explainability, we consider using hybrid approaches as part of our future work (see Sect. 10.2).

**Scalable Semantic Analytics Stack - SANSa** Many distributed data processing approaches rely on Apache Spark, as this framework allows the creation of structured downstream pipelines. However, Apache Spark does not have native support for KGs. With GraphX, there are rudimentary tools for Pregel-like graph processing. However, these do not operate on the direct labeled RDF KGs. The development of the SANSa stack brings together the distributed computation strengths of Apache Spark [13] with Apache Jena's RDF tools [15] into a Scalable Semantic Analytics stack. Over the last few years, several research projects have been developed within the SANSa stack that forms distributed semantic data analytics for individual use cases. Sparqlify [17] is a SPARQL to SQL rewriting framework usable in SANSa that allows distributed query execution on RDF KGs. Further statistical surveys of RDF KG properties were developed in "Efficient distributed in-memory processing of RDF datasets" [108, 109]. Experimental proof of concept projects for numeric outlier detection [110] and clustering on linked geodata [111] have been drafted with SANSa.

We have integrated our novel approaches presented in Chapters 4 - 8 in the SANSa stack. This stack offers the opportunity through Apache Jena and Apache Spark native distributed processing on semantic linked data. Through SANSa, we can reuse modules for KG data read-in and write-out.

Additionally, the first approaches for distributed SPARQL-to-SQL rewriting build a baseline for some feature extraction transformers.

### 3.4 KG ML Meta-Dimensions

The development of KG-based ML requires investigation in meta dimensions. These meta-dimensions include optimizations of reproducibility, reusability, and explainability of ML in the context of KG data. Research efforts increased in the domain of Ethical AI and ML, as well as discussions on optimizing the sustainability aspects of ML.

#### 3.4.1 Reusability and Reproducibility through Metadata Semantification

KGs in ML pipelines are majorly used as the source of training data. Likewise, ML or Inference may create new KG data. Link prediction, inference, or tabular data to RDF triple mapping can create native KG predictions. Another opportunity for novel KG data created in the context of ML processing is the storage of the pipeline's metadata. These contain the respective configuration and the hyperparameter setup. This metadata annotation supports the subsequent reproducibility of the ML pipelines. The native integration of the ML results into the source KG allows easy reusability of predictions together with the source data. Ontologies like MEX [112] or ML-Schema [113] have been developed to align the ML pipeline metadata annotation creation. Notebooks and Platform as a Service (PaaS) providers are a second technical improvement for ML pipeline reusability and reproducibility [114]. Notebooks are a way to present executable code complemented by documentation, intermediate computation results, evaluation, or plots [115]. Several PaaS providers [19] facilitate the hosting and processing of ML pipelines so that an initial setup effort is reduced and the necessary computational power can be scaled as required.

In Chapters 4, 6, and 7, in which we produce data analytics results, we annotate the data analytics results and the corresponding pipeline metadata in addition to the source data. Chapter 8 also presents how all newly developed algorithms and frameworks can be integrated into PaaS Systems and containerized environments, which makes the technology more accessible and reusable for stakeholders.

#### 3.4.2 Ethical AI

In recent years, many ethical dimensions and implications of AI have been investigated and discussed. In various use cases, critical AI and ML-driven results have been reported [50, 51, 114]. Based on these examples and the theoretical implementation of AI, e.g., through ML, guidelines have been developed to support ethical AI research, development, and production. Core topics are discrimination by data bias [51, 116], interaction with humans [117, 118] and intervention possibilities with AI [51], sustainability dimensions of processing-intensive AI [11, 21, 54], explainability of AI predictions [51, 117, 119–121], and personal data protection [51, 122].

In Chapters 5, 6, and 7, we develop data analytics approaches and ML pipelines with transparent and explainable result creation. We also consider good reusability and reproducibility through data Semantification, PaaS, and easy-to-use ML pipeline transformer modules. Especially in Chapter 9, we focus on the ethical AI and sustainable AI dimensions in the dedicated environment of KG-based ML.

Thereby, challenges and solutions for optimization are outlined based on the R&D life-cycle through versatile examples.

### 3.5 Summary

Our literature review and project involvements identified the research gaps that initiated our research questions in Section 1.3. In general, we came across many approaches that supported further development in the KG-based ML research field. Since KGs data offer a valuable source for data analytics and ML through data integration, approaches were developed to operate on these KG data and perform ML pipeline steps [12, 14, 71, 96]. The approaches addressed, on the one hand, the challenge to produce the fixed-length numeric feature vectors [10, 65] and, on the other hand, to implement the necessary scalable approaches due to the amount of data in KGs [12, 14, 96]. Several feature tensor creation algorithms [10, 63, 65, 68] and libraries have been developed [12, 71, 96] that exploit the linked data structure. However, since many use cases utilize KG data for data integration, the relevant information is also contained in the KG literals. Existing literal-based feature extraction methods rely on SPARQL-based feature extraction [17, 58, 59], which requires deep KG-data, KG-ontology, and SPARQL knowledge. We see a research gap in autonomous SPARQL-based feature extraction for multi-modal RDF KGs, as there is no way to use assisted propositionalization strategies natively on graphs in the context of the widely used distributed computation tools such as Apache Spark [13, 16]. Our work on integrating literal-based KG data into ML pipelines is presented in Chapter 5 and Chapter 6. Furthermore, data analysis tasks such as all-pair semantic similarity estimation exhibit at least quadratic complexity. Existing probabilistic algorithms and distributed processing implementations exist for tabular data [91, 92, 95], but solutions are needed to enable end-to-end semantic similarity estimation for KGs. This research gap is tackled in Chapter 4 and Chapter 7. In the context of ML and AI research, Ethical AI plays an increasingly important role as the omnipresent use of these technologies significantly impacts our daily lives [50, 51, 114]. One facet of Ethical AI investigates ML methods' reliability to reduce MLs' perception of being black boxes [117, 119–121]. This increase of trust in ML is achieved by empowering reproducibility through semantic representations of ML architecture and hyper-parameters [112, 113], open source access to all technology components [16, 114] as well as ML approaches that operate on explainable feature vectors and ML models [58, 59, 68, 120]. However, in the KG-based ML environment, these approaches still need to be brought together, which we do in Chapters 4 - 7. In addition, the subdomain of sustainable AI raises questions about energy consumption [54]. Since significant resources are expended in processing large KGs [11, 21, 54], it is therefore essential that the results can be reused. Through the developments in Chapters 4, 6, and 7, we present how the fusion of source KG, meta-data from the ML experiment and prediction semantification extend the initial KG and create accessible and reusable results. In addition, there are further dimensions for a holistic KG-based ML environment that considers both ethical and sustainability aspects [21, 50, 51, 54, 114, 116]. For individual ML sub-disciplines, these have already been explained [116, 117, 121]. However, for the research and development life-cycle of KG-based ML, we miss an overview that provides a good starting point to approach these multifaceted topics. Therefore, we have compiled a dense version of these ethical and sustainability considerations for KG-based ML in Chapter 9.

---

## Scalable Similarity Estimation for RDF KGs

---

**Acknowledgement** This chapter is based on our scientific publication: *DistSim - Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs*, Carsten Felix Draschner, Jens Lehmann and Hajira Jabeen, 2021 IEEE 15th International Conference on Semantic Computing (ICSC), 2021, pp. 333-336, DOI: 10.1109/ICSC50631.2021.00062 [22]. Further details can be found in Section 1.4, especially in Section 1.4.2.

### 4.1 Motivation

Knowledge Graphs (KG) have been used for modeling various data domains in the last years [29]. The data integration possibilities offer high potential to use these KG for data analytics, data science, and machine learning (ML) [14]. Various data analytics and ML approaches require some form of semantic similarity estimation. Semantic similarity estimation in the context of KG data describes the similarity of two entities. These similarity scores can be used to implement data analytics and ML approaches such as clustering, classification, recommendation systems, or entity matching. In clustering, entities are grouped based on their semantic similarity to each other, so entities in a cluster are more similar than entities from different clusters. For recommendation systems, entities are suggested that are similar to previously preferred entities. In the case of entity matching, entities with extremely high similarity are merged. *All-pair-semantic-similarity*, i.e., the computation of all similarity scores for all entities in the graph already has  $O(N^2)$  complexity independent of the computation of the similarity score. In addition, the KG data can take considerable data dimensions<sup>1</sup>. Single entities can use various associated features, which implies that the process requires immense processing power and available main memory. Since individual computers can no longer provide these resources, it is necessary to scale horizontally. In this case, several computers are used in a network to share the KG's data and perform the required computations in a distributed manner [14, 123]. Frameworks have been developed for distributed data analytics pipelines. Still, these distributed executable frameworks do not currently support native handling of RDF KG for the computation of semantic similarity scores. In this chapter, we introduce DistSim, the scalable **D**istributed in-memory semantic **S**imilarity estimation framework for RDF KGs. DistSim can compute both conventional semantic similarity scores and probabilistic similarity scores. Probabilistic similarity scores are a more scalable version of classical

---

<sup>1</sup> <https://www.w3.org/wiki/DataSetRDFDumps>

algorithms to optimize computational complexity [92]. In principle, the goal is to find a good trade-off between reduced computational complexity and manageable losses in the quality of the results. Due to the effort of semantic similarity estimations and the possible further use options, the results should be reusable and reproducible. Through the result semantification and by providing the whole algorithms integrated into the scalable semantic analytics stack SANSa, DistSim can be used for various projects in the semantic web community. The major contributions of this work are:

- DistSim approach for scalable semantic similarity estimation on RDF KGs
- Open source DistSim modules utilizing Apache Spark and Apache Jena
- Scalability evaluation across dataset size dimensions and horizontal scaling processing power
- SANSa DistSim release covered by sample code usages, sample files, Scala docs, and unit tests available in a SANSa stack release

This chapter is organized as follows: in Section 4.2, we introduce the DistSim approach. This section includes algorithmic as well as technical details and considerations. In Section 4.3, we evaluate the approach through experiments, especially in the dimensions of scalability over dataset size and horizontally increasing processing power. Finally, we give in Section 4.4 a summary of this chapter.

## 4.2 DistSim Approach

This section presents the algorithmic and technical inside of our DistSim approach and resource. Section 4.2.1 shows an overview of the DistSim pipeline architecture. In Section 4.2.2, we explain the technical implementation and how the resource is made available. Additionally, we present how result and metadata semantification results appear in a native RDF KG. In Section 4.2.3, we describe the feature extraction process used by the semantic similarity estimation approaches presented in Section 4.2.4. The pipeline modules provide settable (hyper-)parameters. Their effect is explained in Section 4.2.5. We show use cases and applications of DistSim in Section 4.2.6.

### 4.2.1 Pipeline Architecture

The scalable distributed in-memory semantic similarity estimation is implemented as a stacked pipeline aligned with the standards of Apache Spark MLlib. Figure 4.1 shows how the approach consists of the seven modules: *ReadIn*, *Query*, *Feature Extraction*, *Vectorization*, *Similarity Estimation*, *Metagraph Creation*, and *Result Storage*.

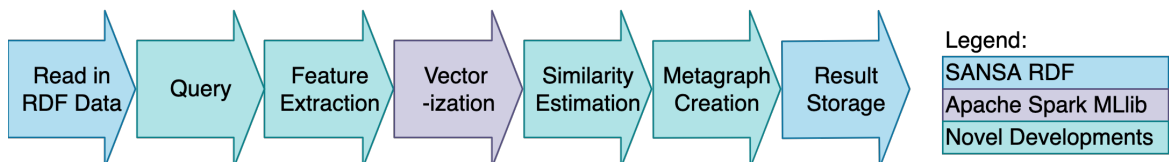


Figure 4.1: DistSim pipeline architecture



DistSim uses ReadIn and WriteOut software modules from the SANSa stack RDF Layer<sup>2</sup> to read and write RDF data. For scalable similarity estimation, it uses CountVectorizer and MinHash with Locality Sensitivity Hashing from Apache Spark MLlib<sup>3</sup>. In addition, DistSim provides feature extraction, a set of similarity estimation models, and meta-graph creation as novel contributions. The pipeline for semantic similarity estimation reads in and writes out RDF data. The modules in between handle the data as Apache Spark DataFrames, which are the standard data representation for ML pipelines.

#### 4.2.2 DistSim as Resource

DistSim is developed as an open-source framework and is fully integrated into the GitHub repository SANSa-ML. The framework is written in Scala and uses Apache Spark to provide a scalable and distributed framework. The software modules and their usage are documented with scala-docs<sup>4,5</sup>. The novel developed software modules are aligned with existing standards in name-space and module usage. The Feature-Extractor is implemented as a Spark Transformer. The implementation of similarity estimation models is aligned with the MinHashLSH model. The estimators provide methods for *nearestNeighbors*, which estimates for one URI the  $k$  most similar elements represented by their URI. Alternatively, *allPairSimilarity* (APS) calculates the similarity of all pairs of URIs from the two DataFrames (of length  $n$  and  $m$ ). DistSim provides the output RDF data enriched with similarity annotations and meta-information about the similarity experiment setup (like the hyperparameter) as output. Figure 4.2 shows, in a small sample, how the internal DataFrame with pairs of URIs and assigned similarity values results in RDF data representation.

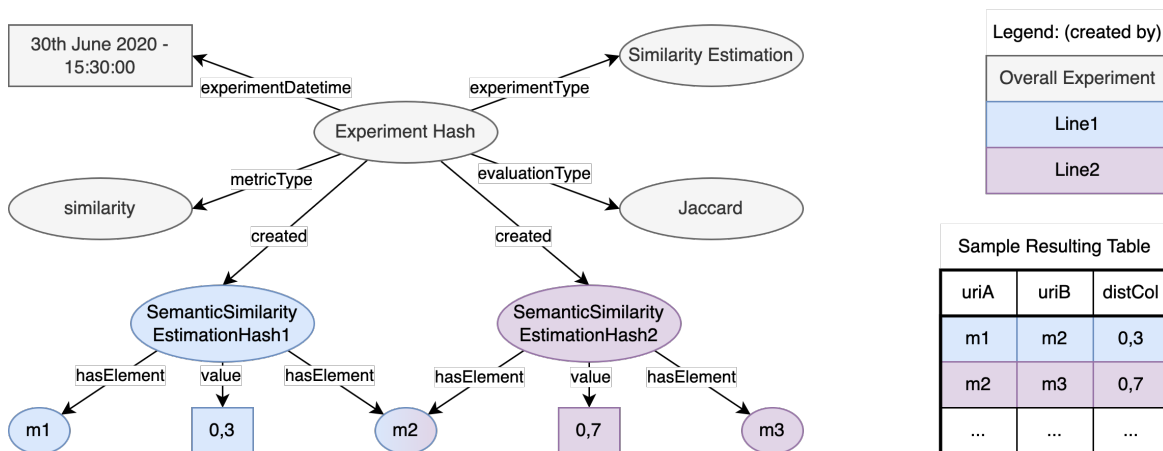


Figure 4.2: Sample resulting meta graph representing semantic similarity estimations

This representation of similarity values allows querying over data to search for similar values instead of recomputing them. Also, other data analytics approaches can use available similarity relations. The annotated meta-information of the similarity estimation not only makes the results reproducible,

<sup>2</sup> <https://github.com/SANSa-Stack/SANSa-Stack>

<sup>3</sup> <https://spark.apache.org/mllib/>

<sup>4</sup> <https://sansa-stack.github.io/SANSa-Stack/>

<sup>5</sup> <https://docs.scala-lang.org/style/scaladoc.html>

but semantification also allows the possibility to comprehend the conditions and parameters used for the estimation. The SANSA stack is maintained by a consortium of research institutes, and further developments are ongoing. For every novel developed module, we provide unit tests.

### 4.2.3 DistSim Feature Extraction

The semantic similarity estimations of DistSim operate based on feature sets. These feature sets are derived from the reading in the RDF dataset by the *Feature-Extractor* module, which is implemented as a Transformer. Developers can set the *Feature-Extractor* methodology over the modes. The mode specifies how the information stored in triples for a specific URI is transformed into the assigned feature set. Table 4.1 presents the 12 current available modes.

Mode Combinations	All	In	Out
Triples	AT (see Figure 4.3)	IT	OT
Neighbors	AN (see Figure 4.4)	IN	ON
Relations	AR (see Figure 4.5)	IR	OR
Stacked	AS (see Figure 4.6)	IS	OS

Table 4.1: Overview of available DistSim feature extraction modes

The first feature extraction mode, *AT*, corresponds to all triples. In this mode, we use all triples which contain the respective URI. The parts of the triples are flattened and used as features. Figure 4.3 shows on the left side the sample data. The node for the respective URI (movie 1) is highlighted in blue. The nodes and relations taken into account for feature extraction are highlighted in green. All information not included in the feature set is annotated in red. On the right-hand side of figure 4.3, the resulting assignment of URI "movie 1" and the resulting features from Feature Extraction in mode AT are presented in tabular (DataFrame) representation.

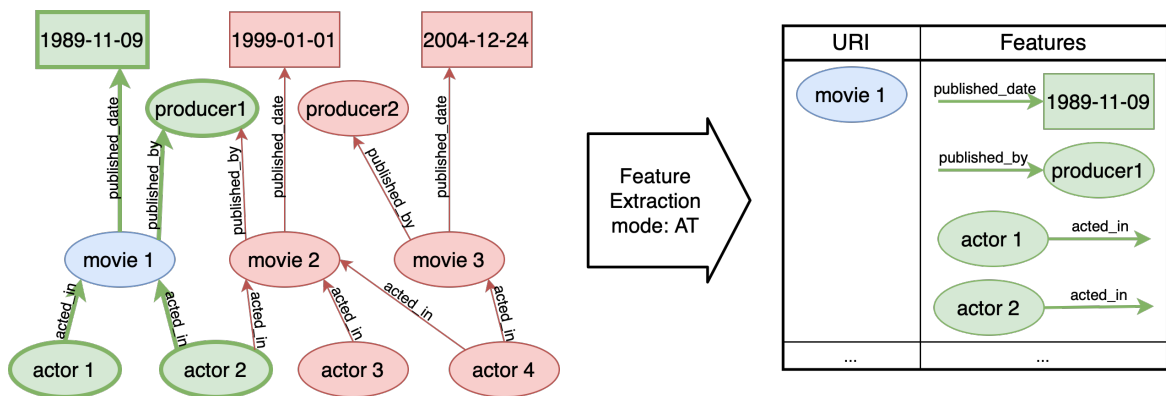


Figure 4.3: Feature extraction in mode AT

Figure 4.4 shows the feature extraction for the mode AN. In this mode, the relations are ignored, and the feature set corresponds to the neighboring nodes. Compared to the modes IN or ON, all neighbors are considered. In the case of IN, only nodes are used as features where the neighboring node is a subject in the triple. For ON, all adjacent nodes where the neighboring node is the object in

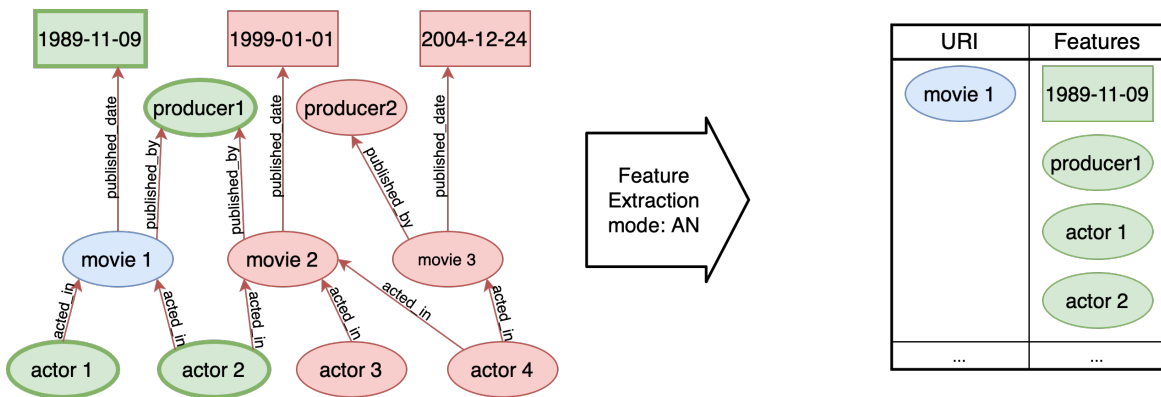


Figure 4.4: Feature extraction in mode AN

its respective triples. This A, I, and O system applies to all twelve modes. Figure 4.5 presents the feature extraction using only the relations. In AR, once again, all relations are used. To modify the used direction options, the modes IR and OR can be set.

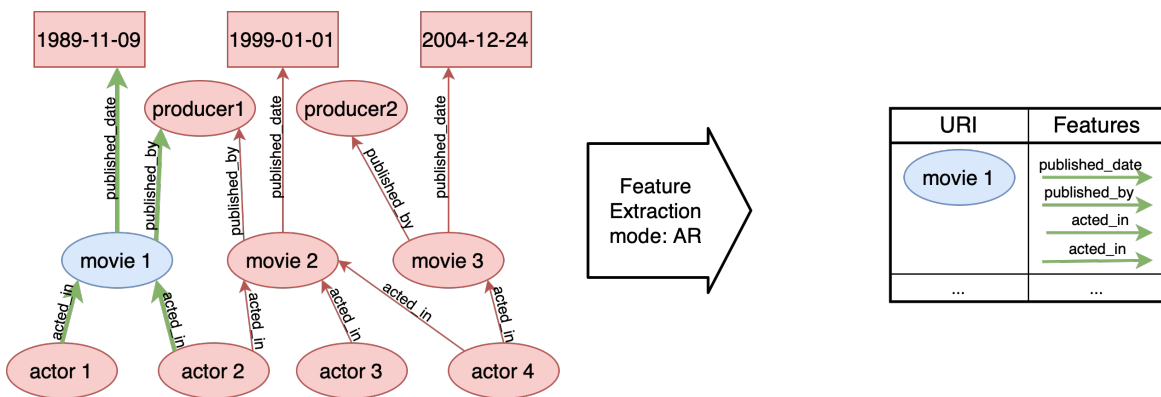


Figure 4.5: Feature extraction in mode AR

Figure 4.6 shows the feature extraction mode AS. In this mode, all neighboring nodes and all relations are extracted for the feature set. Compared to AT, the extracted triple information does not go into the same feature but is used separately or stacked. That is the reason for the S in AS. These different modes allow feature set generation for various use cases and are easily exchangeable by only setting up the model with a specific parameter (see Figure 4.1, Line 4). Figure 4.1 shows how the Feature-Extractor module is called within a pipeline.

```

1 // read in data as DataFrame and feature extraction
2 val triplesDf: DataFrame = [...]
3 val fem = new FeatureExtractorModel()
4   .setMode("an")
5   .setOutputCol("extractedFeatures")
6 val df = fem.transform(triplesDf)

```

Listing 4.1: Code to call feature extraction module

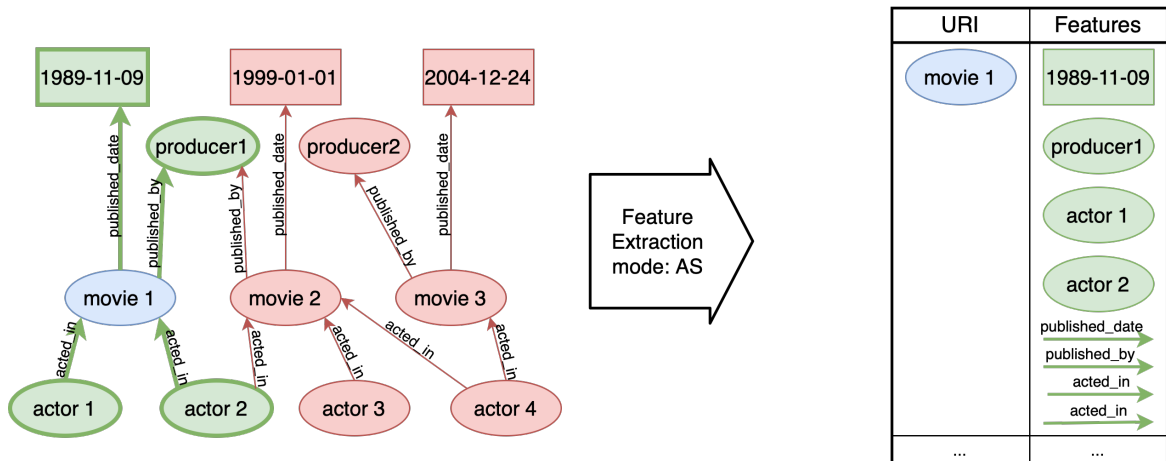


Figure 4.6: Feature extraction in mode AS

#### 4.2.4 Semantic Similarity Estimation Models

DistSim provides feature set based semantic similarity estimations: Batet [76], Braun-Blanquet [85], Dice [86], Jaccard [87], MinHashLSH [95], Ochiai [88], Simpson [89], and Tversky [90] (see table 4.2). The scalable alternative MinHashLSH [91, 95] can be used as a probabilistic approach in calculating Jaccard [87] similarity. This scalable but less precise method is optimal for large-scale data scenarios. In addition, we can stack MinHash with different DistSim models, such that we calculate a set of first estimates in scalable processing time and call in a second step more accurate functions only on promising candidates.

Similarity Coefficient	Formula
Batet Distance	$\log_2 \left( 1 + \frac{ X-Y + Y-X }{ X \cap Y + X-Y + Y-X } \right)$
Braun-Blanquet Similarity	$\frac{ X \cap Y }{\max\{ X ,  Y \}}$
Dice Similarity	$\frac{2 X \cap Y }{ X + Y }$
Jaccard Similarity	$\frac{ X \cap Y }{ X \cup Y }$
Ochiai Similarity	$\frac{ X \cap Y }{\sqrt{ X * Y }}$
Simpson Similarity	$\frac{ X \cap Y }{\min\{ X ,  Y \}}$
Tversky Similarity	$\frac{ X \cap Y }{ X \cap Y +\alpha X-Y +\beta Y-X }$

Table 4.2: Feature set-based semantic similarity estimation formulas

The semantic similarity estimation modules can be called with a few lines of code. They follow the existing Apache Spark MLlib MinHashLSH similarity estimation module. All modules provide the nearest neighbor estimation and all pair similarity. Figure 4.2 shows how Jaccard [87] or Tversky [90] are called.

### 4.2.5 (Hyper-)Parameter Setup Effects

The high modular DistSim pipeline allows many adjustments over (hyper-) parameters. The Scala-docs documentation completely describes the hyperparameters. Over hyperparameters, we can reduce memory usage and processing time. The trade-off comes with a loss of information. The Apache Spark *CountVectorizer* transforms a set of features into a vector with a fixed length. The length results from the number of all features given by all entities. If a feature occurs less often than the set minimal document frequency (*minDf*), it is not part of the vocabulary. The parameter *maxVocabSize* sets the maximal number of features that are the upper bound for the resulting feature-vector-length. With low values set for (*minDf*), the feature vectors become larger and need more memory and processing time. *CountVectorizer maxVocabSize* in a small numeric value yields information loss if the actual number of distinct features is greater than the set number. A smaller value results in a reduction in memory usage and processing time. In semantic similarity estimation over *MinHashLSH*, a higher number of hash tables (*numHashTables*) reduces the false-negative rate in detecting similar elements but increases the processing time and memory usage. The *threshold* for minimal entity similarity, can minimize memory usage and processing time. If this threshold is more strict, fewer pairs of similar values have to be processed over a distributed system in *allPairSimilarity*.

### 4.2.6 Use Cases

Scalable distributed semantic similarity estimations are needed in several use cases. The integration into SANSA allows all users of the framework an easy integration of the novel functionalities. The extended SANSA stack is in use as a generic Big Data analytics toolbox of the Horizon 2020 project

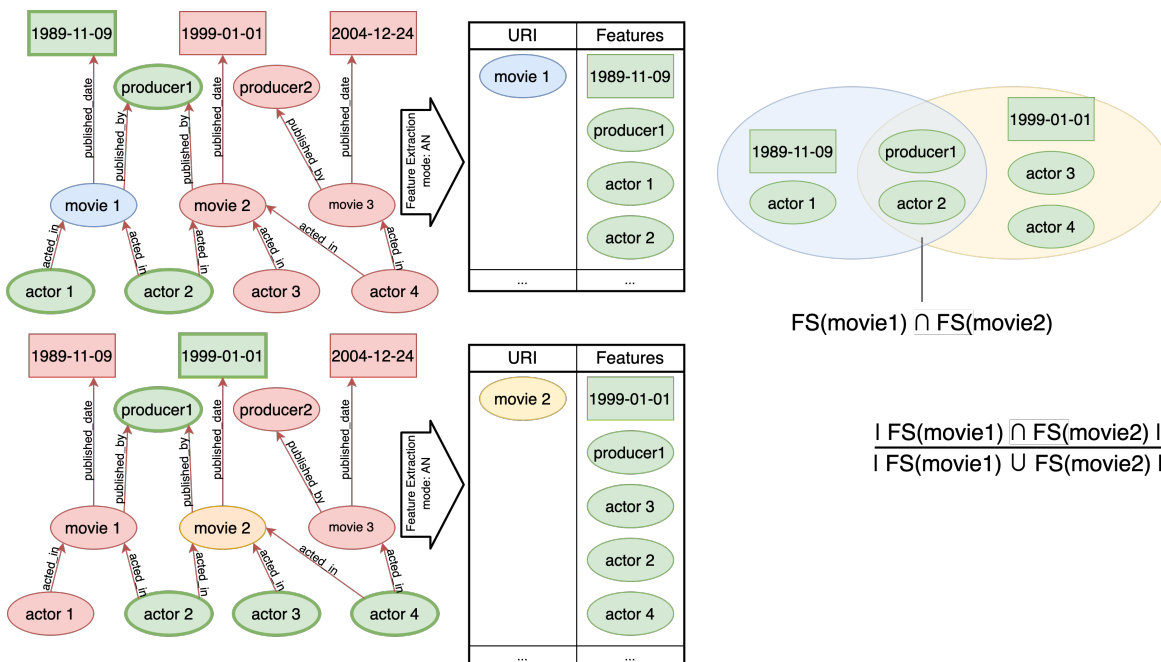


Figure 4.7: Sample similarity estimation with Jaccard

PLATOON<sup>6</sup> (**PLAT**form for **TOO**ls in **EN**ergy). SANSa, as an underlying toolbox for semantic analytics in Opertus Mundi<sup>7</sup>, provides with the novel developed semantic similarity pipeline needed software modules. The project Simple ML<sup>8</sup> provides an easy-to-use generic stackable ML framework that uses SANSa and DistSim as underlying semantic similarity technologies for RDF data. The modular design allows easy integration in arbitrary analytic pipelines to integrate novel functionalities into SANSa. DistSim became a fundamental preprocessing step within the SimE4KG pipeline [27] (see Chapter 7).

---

```
1 // Data from previous steps
2 val sampleKey: Vector = [...]
3 val cvDf: DataFrame = [...]
4 // Jaccard Similarity
5 val model: JaccardModel = new JaccardModel()
6   .setInputCol("vectorizedFeatures")
7 var df: DataFrame = model
8   .nearestNeighbors(cvDf, sample_key, 10)
9 var df: DataFrame = model
10  .similarityJoin(cvDf, cvDf, threshold = 0.5)
11 // Tversky Similarity
12 val model: TverskyModel = new TverskyModel()
13   .setInputCol("vectorizedFeatures")
14   .setAlpha(1.0)
15   .setBeta(1.0)
16 var df: DataFrame = model
17   .nearestNeighbors(cvDf, sample_key, 10)
18 var df: DataFrame = model
19   .similarityJoin(cvDf, cvDf, threshold = 0.5)
```

---

Listing 4.2: Code to call similarity estimation modules

### 4.3 Experiment and Evaluation

DistSim implements well-established similarity estimation functions for RDF data. The evaluation of the performance assessment of DistSim on different data sizes and varying cluster processing setups is needed. Here, the processing time is an indicator of DistSim's distributed processing and scalability. DistSim, as part of the SANSa stack, can be fine-tuned with several (hyper-) parameter settings. These (hyper-) parameters have an impact on the processing time (see Sect. 4.2.5). Our experiments evaluated the effects over a (hyper-) parameter grid. Each experiment has been run at least five times, and the median of experiments is presented as the processing time. The minor variations in processing time are due to background processes. DistSim provides a dedicated implementation to run evaluation experiments over multiple files and a (hyper-) parameter grid. The files and (hyper-) parameter grid can be set up by a *typesafe* configuration file. This configuration file allows easy reproducibility of experiments and easy human-readable and editable evaluation setups. The evaluated experiments result

---

<sup>6</sup> <https://cordis.europa.eu/project/id/872592/de>

<sup>7</sup> <https://www.opertusmundi.eu>

<sup>8</sup> <https://simple-ml.de>

in a DataFrame. Each line in the DataFrame represents one experiment run. Columns correspond to the (hyper-) parameters, data sources, metadata, and measured processing time. The processing time is evaluated in the pipeline modules s.t. we can quickly assess possible bottlenecks. The cluster processing power is adjusted over the spark-submit command, where the number of executor cores can be adjusted.

### 4.3.1 Dataset Description

The evaluation of the scalability of DistSim is performed over multiple datasets with different sizes. The dataset sizes are adjusted by creating synthetic datasets and not by fractions of big datasets. We use synthetic datasets to ensure equally distributed graph density. Cutting off fractions in real-world graphs could lead to an abnormal appearance. The movie domain inspires the synthetic generated datasets. We generate  $n$  movie URIs corresponding to the number of movies. We also generate sets of producer and actor URIs. Each movie gets assigned one producer and a set of actors. Besides, each movie gets assigned a literal representing the publish date. Figure 4.8 shows the principle structure of the generated dataset. The evaluations show that starting from  $10^3$  to  $10^4$  number of movies, the MinHashLSH outperforms Jaccard and others (see Figure 4.10). This dataset size is also used to evaluate the other experiments.

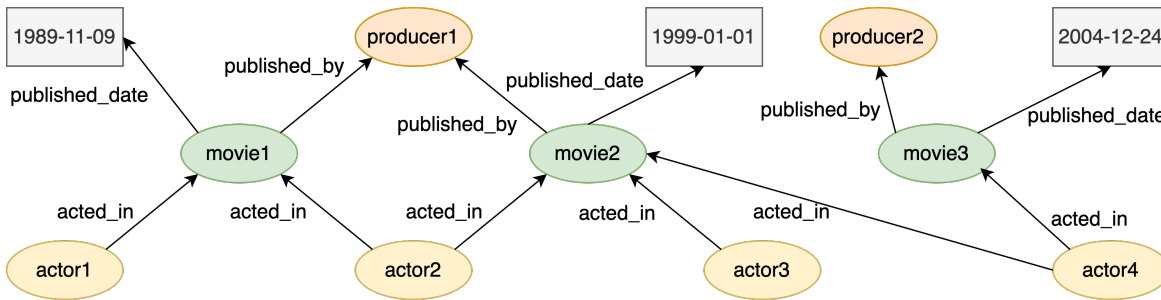


Figure 4.8: Synthetic generated sample data snippets

### 4.3.2 Scalability over increasing horizontal Cluster Computation

The processing power is regulated over the number of available cores. In our experiment, we ran the same set of experiments over a different number of cores. The number of cores starts with  $2^2 = 4$  up to  $2^8 = 256$ . Figure 4.9 shows the scalability over cluster setups. We see an apparent decrease in processing time over increasing computation power.

The reduction of processing time is not lowered by almost 50% when doubling the processing power because managing more cores increases overhead. The bigger the dataset size is, the more evident the reduction of processing times with more processing power.

### 4.3.3 Scalability over Dataset Size

DistSim integrated into the SANSA stack provides semantic similarity pipelines native working on RDF datasets. The use of probabilistic similarity estimator *MinHashLSH* allows scalable processing of large-scale RDF data. Figure 4.10 shows for all pair similarity estimation that, MinHash (Fig. 4.10

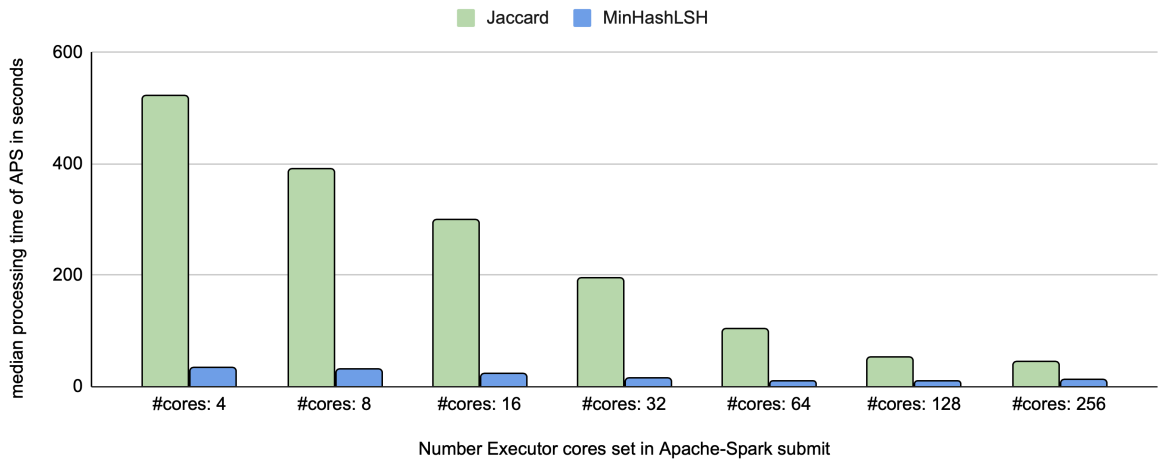


Figure 4.9: DistSim processing power scalability on 10E4 movie data

orange) scales better than the other approaches. The approaches of Batet, Braun Blanquet, Dice, Jaccard, Simpson, and Tversky scale similar. In the Figures 4.9 and 4.11, where only Jaccard and its scalable alternative MinHash, are presented, the other approaches were on the same processing time level as Jaccard. For *nearest neighbor* estimation, all approaches are on a similar scalable level, including *minHash*, because *nearest neighbor* is a linear operation and not quadratic in complexity like all-pair similarity (APS).

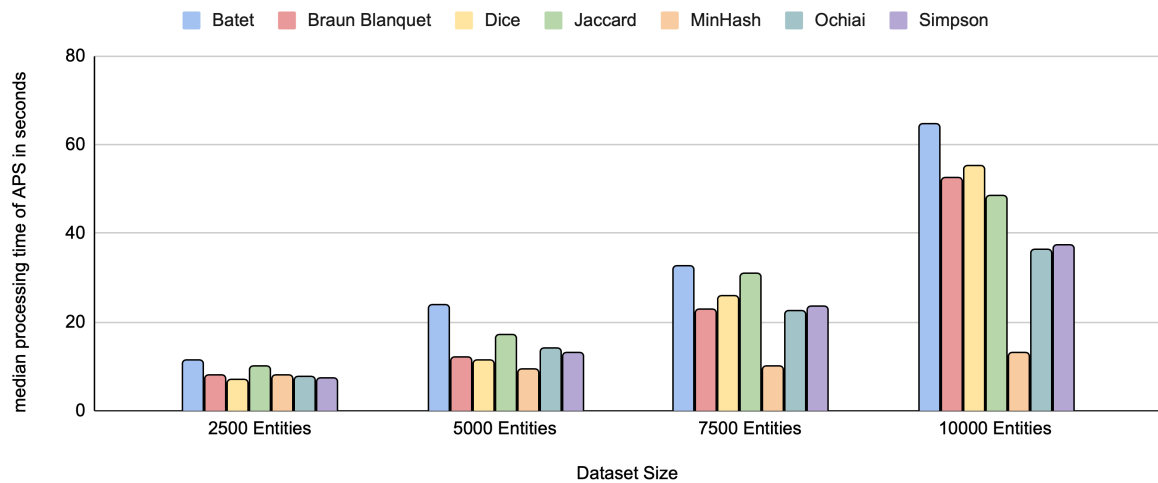


Figure 4.10: Data size scalability of All-Pair-Similarity estimation

### 4.3.4 Hyper-Parameter Evaluation

DistSim provides several (hyper-) parameters explained in section 4.2.5. This section visualizes the effects of various hyper-parameter setups on Jaccard’s models and its scalable alternative MinHash. In the pipeline, all pair similarity is the worst scaling module. The reason is that, in principle, each element needs to be compared to the other, resulting in quadratic scaling behavior. Figures 4.11 - 4.13



present the impact of *CountVectorizer* parameters. Figure 4.11 shows that the bigger the maximum vocabulary size is, the longer Jaccard *AllPairSimilarity* (APS) needs to be performed. This increase in processing time is due to the longer feature vectors. The min-hashing is less influenced because it uses linear transformation before the quadratic scaling of all-pair similarity estimation, reducing the vector length.

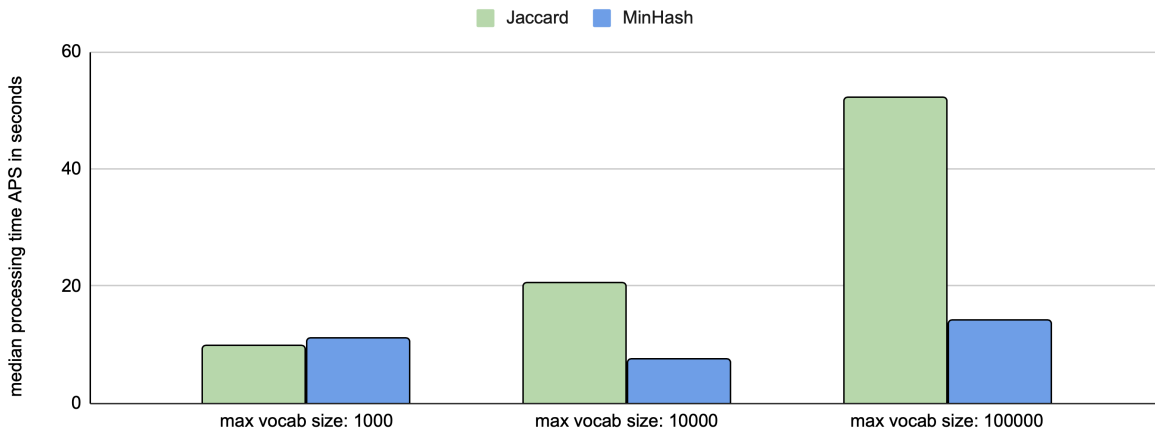


Figure 4.11: Effect of *CountVectorizer* maximal vocabulary size (*maxVocabSize*)

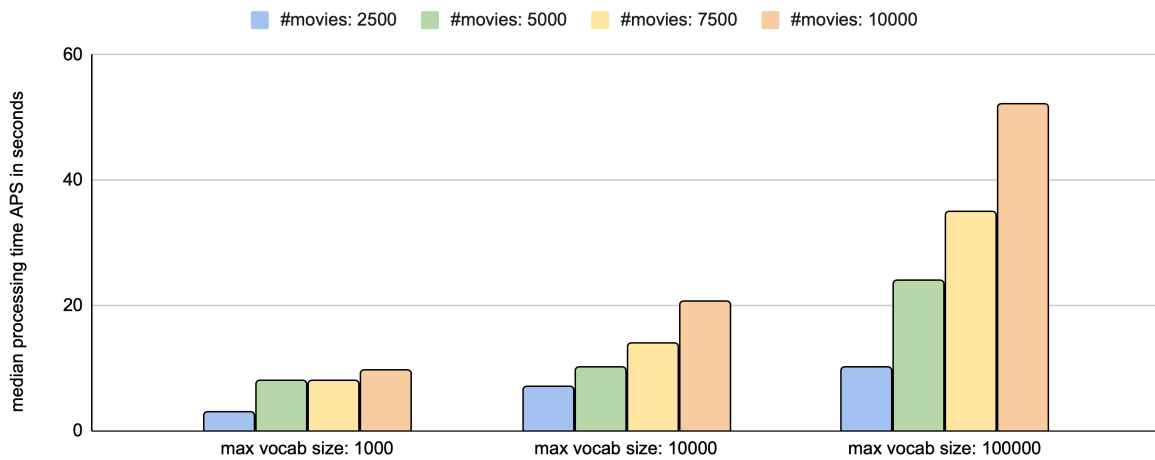


Figure 4.12: Effect of *CountVectorizer* maximal vocabulary size (*maxVocabSize*) for similarity estimation mode Jaccard

Figure 4.13 presents that the greater the *minDf* of *CountVectorizer* is, the faster the estimations are performed. The shorter processing time is due to the reduced features taken into account. Some features occur in fewer documents, so they are not included in the feature vector creation. MinHash with locality sensitivity hashing reduces the feature vector size. The number of hash tables implies the length of the feature vector after min-hashing. This feature vector is also used for bucketing as part of the locality sensitivity hashing. Figure 4.14 shows that an increasing number of hash tables influences the processing time slightly.

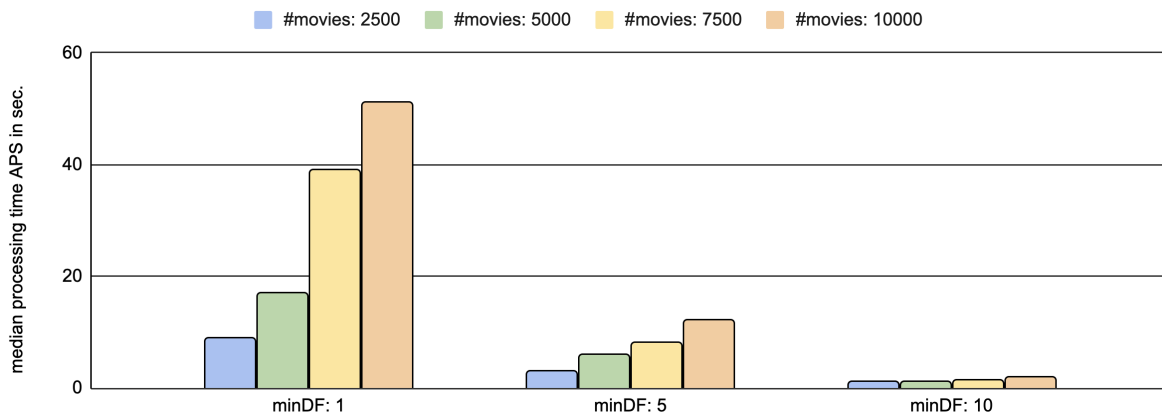


Figure 4.13: Effect of CountVectorizer minimal document frequency (minDf)

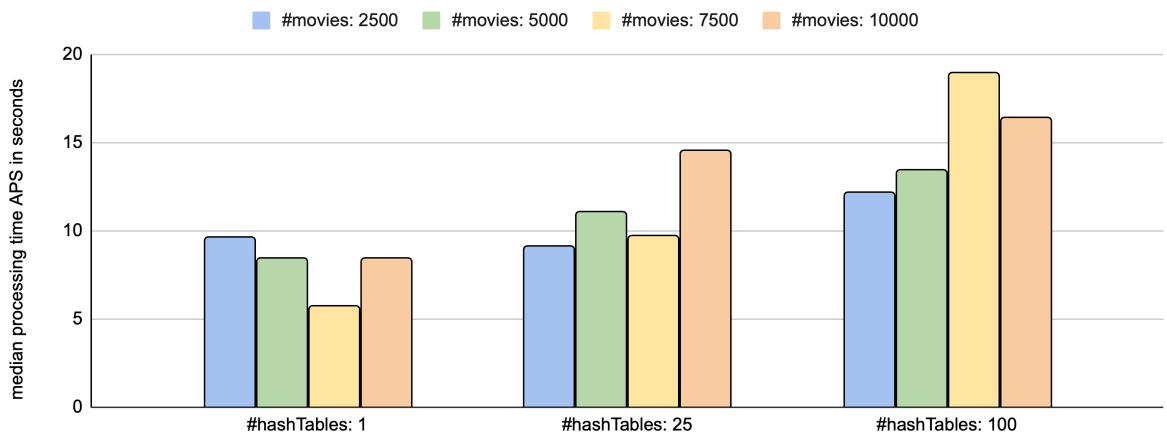


Figure 4.14: Effect of different number of HashTables

## 4.4 Summary

DistSim integrated into the SANSAs stack provides a scalable distributed open-source framework for semantic similarity estimations on RDF KGs. Follow-up projects are already using DistSim modules. The community actively uses the SANSAs stack for scalable distributed semantic analytics on large-scale RDF data. The extension of DistSim now allows more use cases to be performed within this technology stack. In multiple projects, DistSim is in practical use (see section 4.2.6). The availability of an easy-to-use evaluation pipeline offers apparent inferable effects of (hyper-) parameters to the corresponding processing times. The storage in a tabular format and semantic data representation allows high reproducibility and understanding of the needed pipeline setup. The results are human and machine-readable. With DistSim existing analytic pipeline modules for RDF processing, more RDF data analytics can be ported to distributed processing over the extended SANSAs stack. DistSim is documented by state-of-the-art scala-docs and presents quickly usable minimal examples showing how to use the modules with few code lines. The modules are created aligned to the design of Apache Spark MLlib s.t. non-native SANSAs users can easily use these. The evaluation shows good scalability over different dataset sizes and processing power.

---

## An Automatic Scalable RDF Graph Feature Extractor

---

**Acknowledgement** This chapter is based on our scientific publication: *Literal2Feature: An automatic scalable RDF graph feature extractor*, Farshad Bakhshandegan Moghaddam, **Carsten Felix Draschner**, J. Lehmann and H. Jabeen, in: Further with Knowledge Graphs, IOS Press, International Conference on Semantic Systems (SEMANTICS), 2021, pp. 74–88. DOI: 10.3233/SSW210036 [23]. Further details can be found in Section 1.4, especially in Section 1.4.2.

### 5.1 Motivation

In the context of the development of DistSim in Chapter 4 we introduced the relevance of feature extraction in the context of knowledge graph (KG)-based data analytics and machine learning (ML). In particular, many graph features are located in the arbitrary depth of the KG (see Fig. 5.1). For various KG-based data analytics or ML approaches, obtaining fixed-length numeric feature representations is crucial. Some approaches use graph kernels or knowledge graph embeddings (KGE) [124–127], creating latent feature vectors by learning the structural representation of KG samples or collecting over kernels graph data and accumulating a vector representation. For some use cases, this feature representation needs to be both explainable and contain the multi-modal feature data of the KG objects and literals. For the extraction of these multi-modal features, SPARQL query execution may be necessary. The design of feature-extracting SPARQL queries can be challenging, especially considering the possible complex branching of graphs and their taxonomies. Ontologies can support the creation of feature-extracting SPARQL queries. However, if the ontologies are not available or not complete, the production of feature-extracting SPARQL queries becomes infeasible. KG data can result from the fusion of different data sources and KGs. Therefore, no ontology may represent all the existing features. Due to the open-world assumption, it is also possible that despite features being mapped in an ontology, these features are not available, or at least not for every sample. For further information, look into Chapter 2 Preliminaries and Chapter 3 Related Work (SPARQL, Propositionalization, ML, and AI). Since KG data can be complex and large, scalable solutions for such data analytics and feature extraction approaches need to be created. Distributed computing can enable horizontal scaling of KG-based feature extraction. Within this work, we introduce the Literal2Feature approach. Literal2Feature supports the construction of feature-extracting SPARQL queries resulting in

explainable feature matrices. KGs can be large and might exceed the available memory of computers in data analytic tasks and ML pipelines. More processing power can be made available through distributed computation architectures. Literal2Feature has been implemented in Scala using Apache Spark and Apache Jena for distributed semantic native execution. Literal2Feature has been fully integrated into the scalable data analytics stack SANSA [14]. The significant contributions of this chapter are:

- A distributed generic framework generating feature extracting SPARQL queries
- Integration into the scalable semantic analytics stack SANSA<sup>1</sup>
- Coverage with documentation, unit tests, and tutorials [128] available as a SANSA GitHub ML layer extension
- Evaluation of Literal2Feature KG tensor representations for multiple scenarios reaching from clustering to classification
- Scalability evaluation over processing power and dataset size

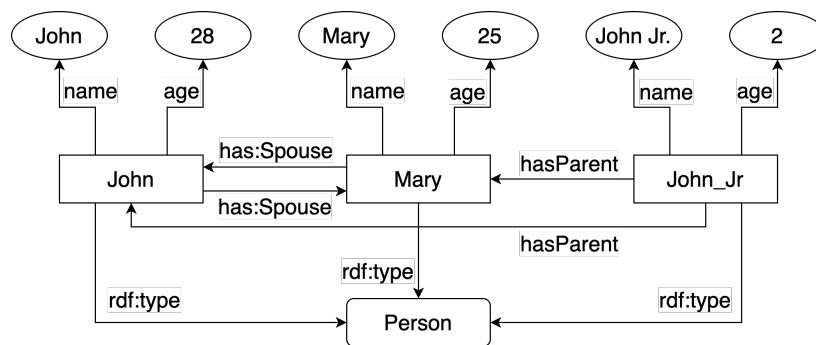


Figure 5.1: A sample RDF graph

This chapter is structured as follows: Initially, we provide insides into the overall Literal2Feature pipeline design. We introduce each pipeline step within Section 5.2.1. Afterward, we provide technical details on how Literal2feature is implemented and available as a resource. Further, we explain in which use cases Literal2feature is already applied. Within Section 5.3, we first provide targeted measures and used datasets. In the next step, we evaluate the performance of created feature matrices and compare those against other state-of-the-art feature creation approaches. The distributed implementation of Literal2Feature is evaluated in multiple experiments comparing the impact of dataset size and processing power. In the discussion, we describe the findings of the experiments. Finally, in Section 5.4, we conclude the results of this chapter.

## 5.2 Literal2Feature Approach

This section presents the Literal2Feature approach. Literal2Feature creates SPARQL queries to retrieve literal-based feature vectors for ML and data analytic pipelines. For several use cases, the

<sup>1</sup> <https://github.com/SANSA-Stack/SANSA-Stack>

KG literals contain valuable information that can be more easily interpreted as features than KGEs. These fetched features are the starting point of downstream ML pipelines. We introduce each step of the Literal2Feature approach. Additionally, we describe Literal2Feature as a resource and provide example use cases where it is already applied.

### 5.2.1 Pipeline Architecture

Literal2Feature itself is implemented as a multi-step pipeline. The major steps are Seed Generation, Graph Search, Path Generator, SPARQL Generator, and SPARQL Execution. Figure 5.2 and Figure 5.3 present the structural pipeline approach.

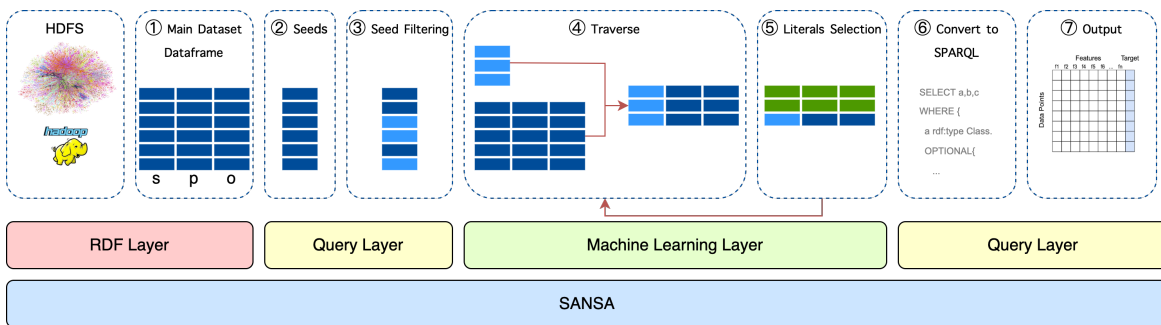


Figure 5.2: Literal2Feature execution pipeline

### 5.2.2 Seed Generator

Literal2Feature starts with a semantic native RDF KG. By the *seedWhereClause* parameter, we define for which entities we want to find features. These seeds can be specified, for example, over the URI type annotation. This filter for selecting seed nodes in the KG will also result in a partial WHERE statement resulting in the created SPARQL query. The fetched seeds are collected, defining a possible starting point for upcoming exploration steps. These seeds are ordered descending by their outgoing links. Seeds with more neighboring nodes are more likely to bring more diverse features in the exploration phase. Alternatively, a random sampling mode for ordering the seeds is also available.

### 5.2.3 Graph Search

A parameterized fixed number of seeds results in the Graph Search's starting points. The Graph Search offers two options that are settable over another parameter. A complete graph search is implemented by Breath First Search (BFS) [129], and an approximated graph search through random walks (RW) [130]. Further parameters are the maximum path traversal length. All searching path traversals will either end in the graph search by reaching a literal feature or ending because of the set maximum depth. These parameters influence the number of features found. The random walk can result in a smaller number of features. Also, a small number of seeds and smaller search depth can result in a smaller number of reached features (see Figs. 5.5 - 5.8). These parameters influence the number of features found and the runtime of graph execution. The dedicated differences are measured in Section 5.3.

### 5.2.4 Path Generator

Based on the generated walks from Graph Search, we extract several paths. We abstract these paths from the visited entities and keep the triple properties/relations. These cleaned paths are collected and reduced to a distinct set of paths.

### 5.2.5 SPARQL Generator

SPARQL queries are generated based on the seed fetching filter and especially the distinct generated paths. SPARQL variables replace entity RDF subject and object within these paths. Properties stay within the paths to define the traversal structure, later reaching the literal features. Starting nodes result in one projection variable, and the path end nodes result in the feature-specific projection variable. Not every seed reaches the same set of features, and not every entity of interest has the same features available; all paths result in WHERE queries that are wrapped by optional blocks. The naming of projection variables reuses the property identifier string indicating the needed graph traversal from seed to final feature. As long as the property naming in the original KG is reasonable, the created projection variable supports explainability through the projection variable identifier.

---

```
1 SELECT ?person ?hasParent_age WHERE {
2   ?person a Person .
3   OPTIONAL {
4     ?person hasParent ?parent .
5     ?parent age ?hasParent_age .
6   }
7 }
```

---

Listing 5.1: Sample result of the generated SPARQL query

### 5.2.6 SPARQL Executor

The generated SPARQL is, in principle, executable through every RDF KG SPARQL execution framework. The executed SPARQL creates a resulting table with feature extractions. A feature set element is selected randomly if multiple features are collected for one entity in one optional block. This random sample of multiple features for the same feature identifier was the first approach, which got extended with the efforts of DistRDF2ML (see Chapter 6). The extensions for SPARQL execution are presented with the SparqlFrame and SmartVectorAssembler.

### 5.2.7 Literal2Feature as a Resource

Literal2Feature is implemented in Scala utilizing Apache Spark, and Apache Jena integrated into the open-source GitHub repository SANSA stack. It is also distributed through the DistRDF2ML release. ReadMes, sample classes, and tutorials build the documentation. In addition, the modules are covered by unit tests automatically executed by GitHub Actions. Further details can be found in Chapter 8.

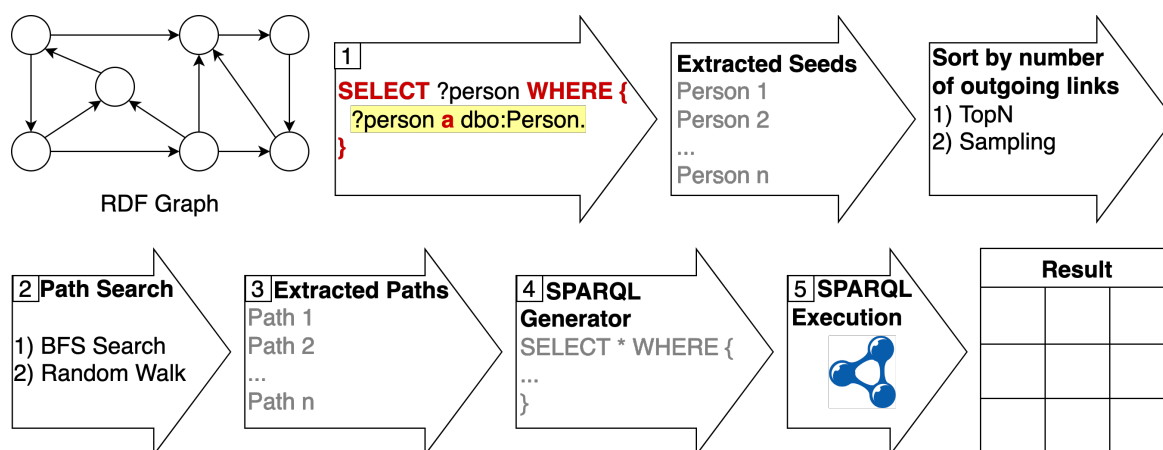


Figure 5.3: Literal2Feature approach processing overview

### 5.2.8 Use Cases

Literal2Feature got applied in multiple use cases. Its explorative behavior, creating explainable literal-based features through a generic tool, is helpful in different scenarios. One project is the PLATOON project, a second project is developed with ENGIE, and other KG-based ML pipelines in the SANSA stack use Literal2Feature.

PLATOON is the digital platform and analytic tools for energy (PLATOON<sup>2</sup>), which is a Horizon 2020 project. PLATOON aims to deploy distributed/edge processing and data analytics technologies for optimized real-time energy system management in a simple way for the energy domain. RDF KG and Ontologies are used to integrate heterogeneous data sources and represent some features in literals. SANSA is an essential part of the Big Data analytic tools in PLATOON that support the exploration and usage of integrated features. PLATOONs heterogeneous stakeholders are supported through Literal2Feature as it allows the creation of feature-extracting SPARQL queries without major knowledge of ontology or SPARQL itself.

ENGIE SA<sup>3</sup> is a French multinational electric utility company that operates in the energy transition, electricity generation and distribution, natural gas, nuclear, renewable energy, and petroleum. Together with ENGIE, we are working on a dataset related to accidents in France. One of the significant challenges is predicting and classifying accidents in an effective and scalable manner. In order to perform this task efficiently and effectively, Literal2Feature integrated into the SANSA stack became an integral module to solve classification.

## 5.3 Experiment and Evaluation

In this Section, we evaluate Literal2Feature from two significant points of view. On the one hand, we evaluate the performance regarding the expressivity of extracted features and how these extracted features perform in different ML and data analytics tasks. On the other hand, we evaluate Literal2Feature’s scalability capabilities across different distributed cluster processing power setups,

<sup>2</sup> <https://cordis.europa.eu/project/id/872592/de>

<sup>3</sup> <https://www.engie.com>

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

Table 5.1: Literal2Feature evaluation dataset statistics (GT = ground truth)

how it behaves with increasing dataset sizes, and how overall (hyper-) parameters influence the result and processing time.

### 5.3.1 Data Description

The evaluation is performed on three different datasets. Two datasets are used for the feature evaluation, and the third is used for the scalability evaluation. The first dataset is the Engie accident dataset<sup>4</sup> which contains vehicle accidents that occurred in France in 2018. The second dataset is the carcinogenesis dataset [131] which contains information about drug molecules. Table 5.1 gives an overview of these two datasets.

For the evaluation across multi-node clusters, we use a synthetic dataset. This dataset is created to have complete control over dataset size and inner structure scaling. The dataset is created in six orders of magnitude, while each node has a branching factor of 50, and features are available at a depth of three. The features lie in a complete tree structure. The German DBpedia has 336M triples resulting in 48GB overall size, which is a similar order of magnitude to our DS6. The DBpedia dataset does not have the same fixed branching factor or feature depth. Table 5.2 shows the details of the synthetic generated datasets DS1 to DS6.

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

Table 5.2: Synthetic dataset description

### 5.3.2 Performance Evaluation

#### Literal2Feature Evaluation in Classification and Clustering Task

The performance of features extracted by Literal2Features is evaluated in classification and clustering tasks. Classification is performed in three scenarios. The first scenario is a classification of how dangerous an accident was. The second scenario is an accident classification on which side of the

<sup>4</sup> Can not be publicly published due to Intellectual Property concerns



Approach		Accident Scenario 1	Accident Scenario 2	Carcinogenesis
FeGeLOD[69]	<i>RF</i>	0.17 ± 0.01	0.05 ± 0.001	0.59 ± 0.07
	<i>LR</i>	0.20 ± 0.02	0.05 ± 0.003	0.61 ± 0.06
	<i>MLP</i>	0.18 ± 0.01	0.05 ± 0.001	0.58 ± 0.07
	<i>XG</i>	0.18 ± 0.02	0.05 ± 0.004	0.58 ± 0.05
RDF2Vec[63]	<i>RF</i>	0.17 ± 0.004	0.05 ± 0.0001	0.41 ± 0.13
	<i>LR</i>	0.25 ± 0.02	0.07 ± 0.006	0.49 ± 0.12
	<i>MLP</i>	0.23 ± 0.02	0.09 ± 0.008	0.51 ± 0.18
	<i>XG</i>	0.23 ± 0.02	0.07 ± 0.006	0.42 ± 0.10
TransE[10]	<i>RF</i>	0.16 ± 0.001	0.05 ± 0.0001	0.52 ± 0.09
	<i>LR</i>	0.17 ± 0.002	0.05 ± 0.001	0.56 ± 0.06
	<i>MLP</i>	0.24 ± 0.006	0.08 ± 0.001	0.56 ± 0.06
	<i>XG</i>	0.24 ± 0.005	0.06 ± 0.001	0.51 ± 0.06
DistMult[65]	<i>RF</i>	0.22 ± 0.01	0.05 ± 0.0001	0.50 ± 0.06
	<i>LR</i>	0.22 ± 0.005	0.05 ± 0.001	0.53 ± 0.06
	<i>MLP</i>	0.26 ± 0.005	0.09 ± 0.004	0.54 ± 0.04
	<i>XG</i>	0.37 ± 0.04	0.09 ± 0.002	0.58 ± 0.05
Simple[132]	<i>RF</i>	0.22 ± 0.01	0.05 ± 0.0001	0.45 ± 0.09
	<i>LR</i>	0.23 ± 0.003	0.05 ± 0.001	0.48 ± 0.04
	<i>MLP</i>	0.28 ± 0.004	0.09 ± 0.004	0.53 ± 0.04
	<i>XG</i>	0.36 ± 0.03	0.08 ± 0.003	0.50 ± 0.07
Literal2Feature	<i>RF</i>	0.37 ± 0.007	0.10 ± 0.01	0.57 ± 0.07
	<i>LR</i>	0.41 ± 0.002	0.12 ± 0.005	<b>0.62 ± 0.08</b>
	<i>MLP</i>	<b>0.46 ± 0.006</b>	<b>0.21 ± 0.005</b>	0.48 ± 0.07
	<i>XG</i>	0.38 ± 0.18	0.19 ± 0.07	0.53 ± 0.04

Table 5.3: Literal2Feature F1-Measure evaluation results

car the accident occurred. The third classification scenario is a carcinogenic drug classification task. As alternative baseline feature creation approaches, we used KGE and propositionalization strategies. These five alternatives are FeGeLod [69], RDF2Vec [63], TransE [10], Simple [132], and DistMult [65]. For FeGeLod, we preserved the initial configuration, and the KGE approaches were created for 200-dimensional vectors. The KGEs have been trained by OpenKE<sup>5</sup> on a Nvidia Geforce GTX 1080 Ti with i5CPU and 8GB of Ram. The learning algorithms selected for classification range from simple linear regression to technical neural networks. More specifically, the models selected are Logistic Regression (LR), Random Forest (RF), XG-Boost [133] (XG), and Multi-Layer Perceptron (MLP). The training is performed through 5-fold cross-validation. As the accident dataset samples are imbalanced in labels for training and evaluation, we present in the following Table the calculated F1 scores. Table 5.3 presents the calculated F1 scores for the scenarios, feature extraction options, and classifiers.

<sup>5</sup> <https://github.com/thunlp/OpenKE>

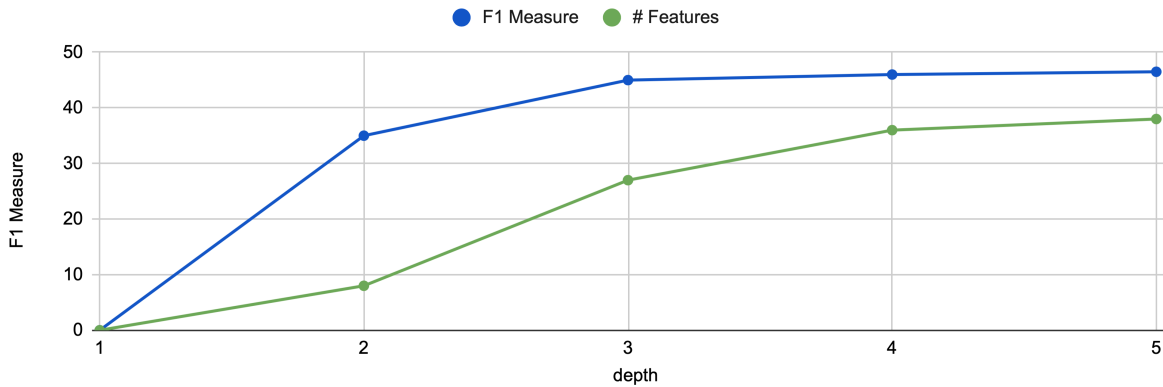


Figure 5.4: Depth impact on the classification result and number of extracted features

Additionally, we evaluated the effect of hyperparameters like the search depth in graph search. The maximum path length impacts the number of features possibly reached. The relevance of the parameter and the effect of predictions are presented in Figure 5.4.

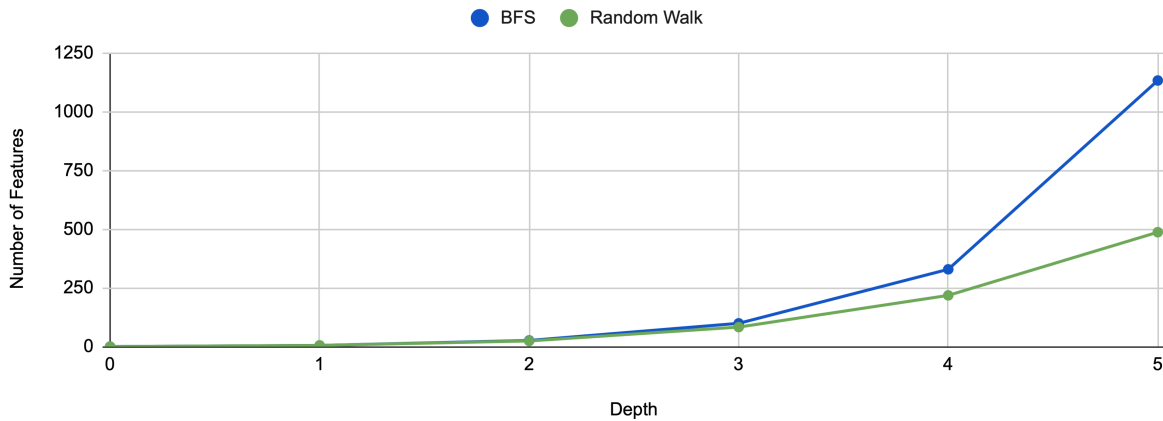


Figure 5.5: Literal2Feature evaluation: depth vs. #features on the ENGIE dataset

Literal2Feature is also evaluated in a clustering scenario. As the data has no ground truth for cluster performance evaluation, we use the silhouette coefficient as the evaluation metric. Two scores are needed to construct the silhouette coefficient. Score  $a$  is the mean distance between one sample and all other points in the same cluster. Score  $b$  is the mean distance of a sample to all other points of the next nearest cluster. The resulting overall silhouette score for one sample is:

$$s = \frac{b - a}{\max(a, b)} \tag{5.1}$$

The Silhouette coefficient is calculated for both datasets, and all five feature extraction respective embedding approaches. The resulting coefficients are presented in Table 5.4.

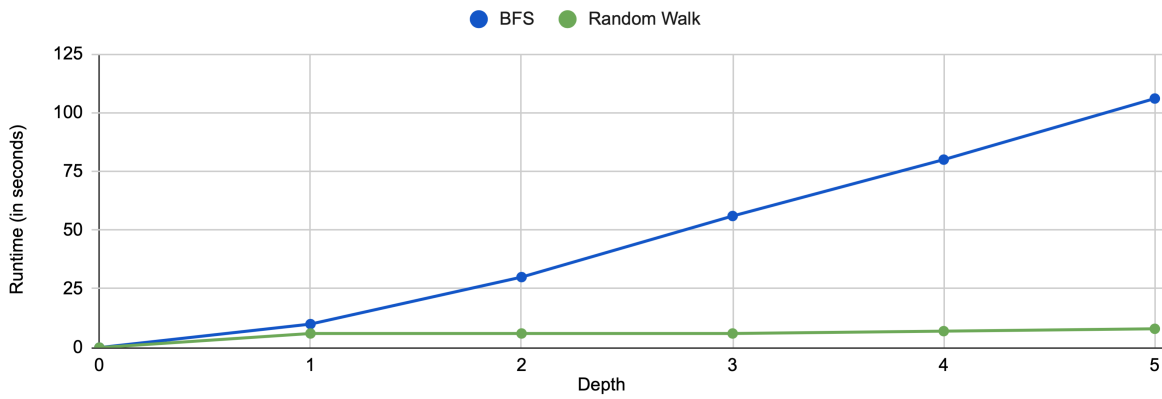


Figure 5.6: Literal2Feature evaluation: depth vs. time on the ENGIE dataset

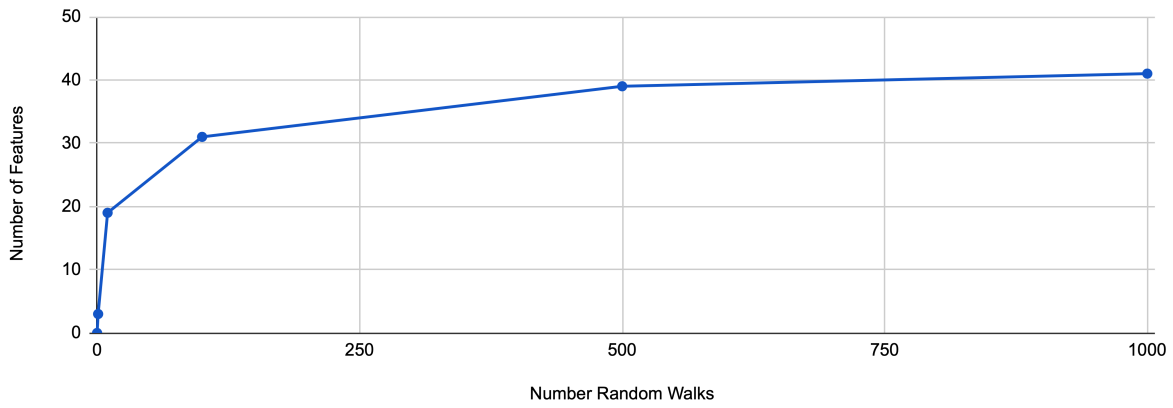


Figure 5.7: Literal2Feature evaluation: #walks vs. #features on the ENGIE dataset

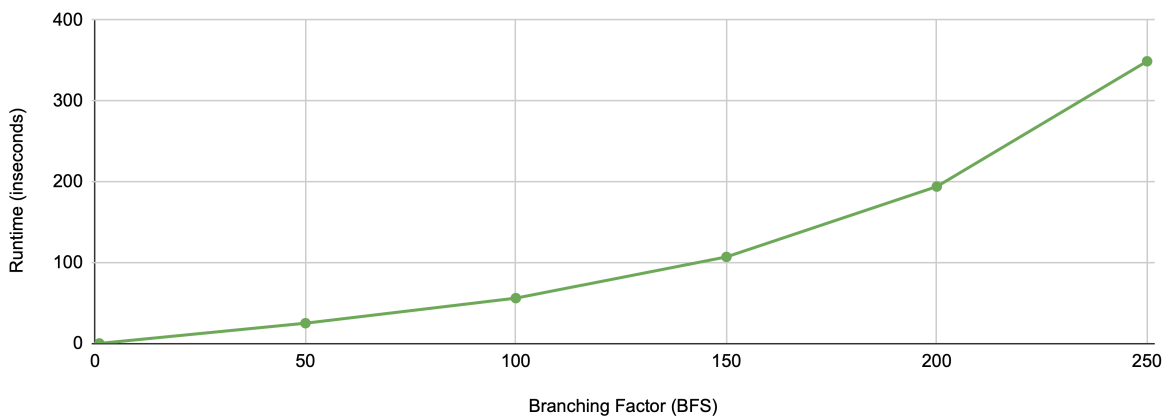


Figure 5.8: Literal2Feature evaluation: branching factor vs. time on synthetic data on a single machine

	Engie	Carcinogenesis
RDF2Vec	0.004	0.263
TransE	0.113	<b>0.669</b>
SimpleE	0.008	0.008
DistMult	0.006	0.009
Literal2Feature	<b>0.133</b>	0.247

Table 5.4: Silhouette coefficient

### Scalability over Dataset and Processing Power

Scalability evaluation over dataset and processing power Literal2Feature is implemented with Scala utilizing Apache Spark and Apache Jena integrated into the SANSA stack. This combination of technologies offers the opportunity to scale horizontally on a distributed cluster computation setup. The cluster setup is built by four computational nodes (one main, three workers), each having AMD Opteron(TM) 6376 with 64 Cores and 256GB of RAM. Gigabit Lan connects the machines. All experiments are executed three times, and the average runtime is reported.

The scalability evaluation over dataset size is performed using the synthetic datasets DS1 to DS6. The processing power is fixed across all datasets to 64 cores. In addition, we compare the effect of full BFS and Random Walk search. The reduced processing time of Random Walk compared to BFS and the effect of dataset size are presented in Figure 5.10.

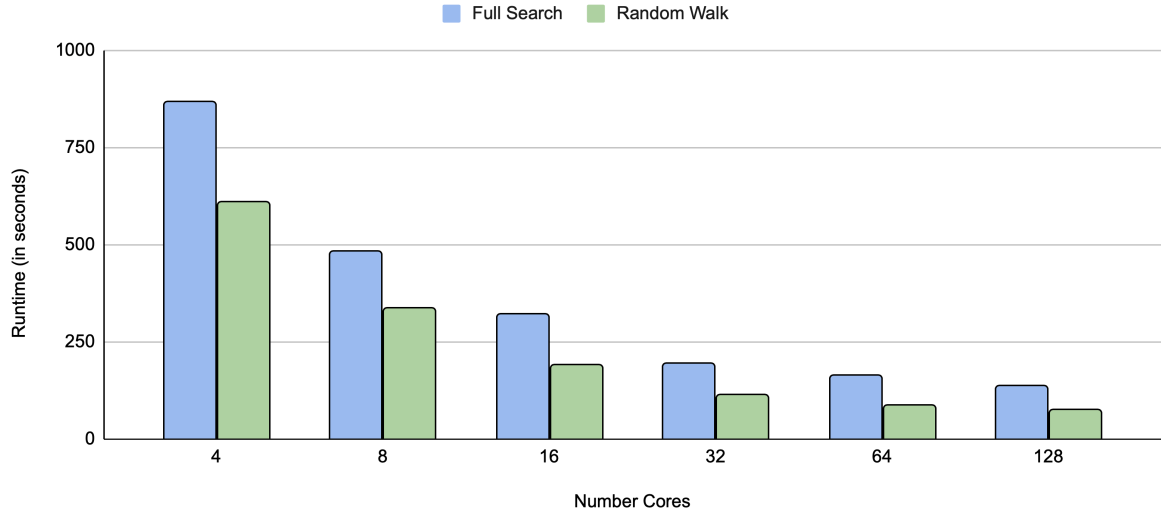


Figure 5.9: Literal2Feature evaluation: processing power scalability on DS 4 dataset

The second central scalability evaluation is performed along with increased processing power. The processing power is adjusted over the Spark Cluster configuration. For the experiments, we perform over fixed dataset multiple executions with a total number of cores starting from  $2^2$  up to  $2^7$  (see Fig. 5.9). The measured runtimes do not include the loading of the dataset nor SPARQL execution, as the query generation builds the central contribution of this work. The execution evaluation of SPARQL

can be found within Chapter 6 (see DistRDF2ML [24] and SPARQLIFY [17]).

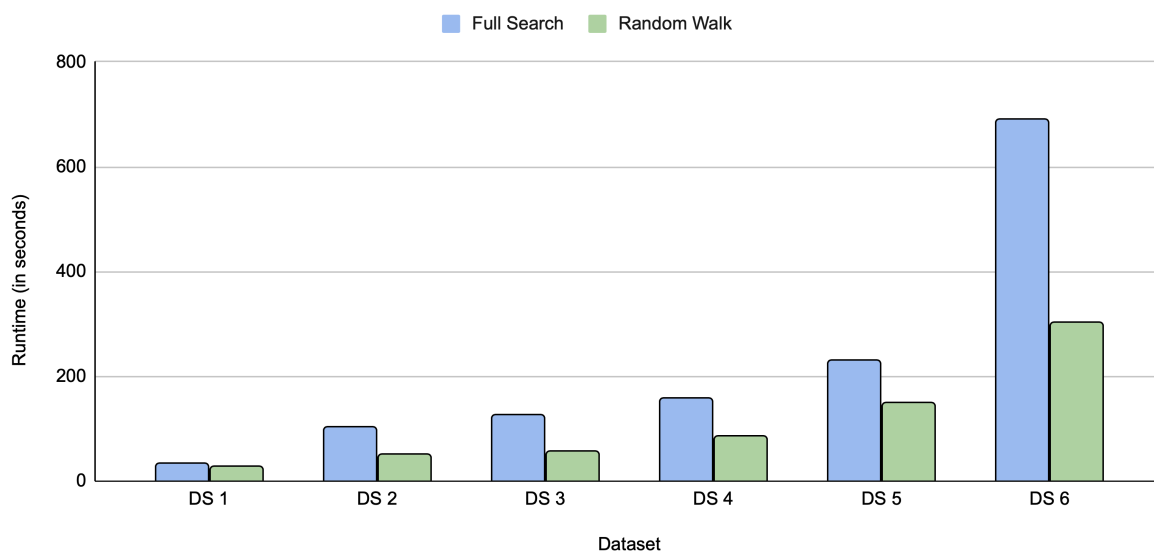


Figure 5.10: Literal2Feature evaluation: dataset size up performance evaluation over 64 Cores

### 5.3.3 Discussion

The classification experiments evaluate the performance of Literal2Feature in all three scenarios. Table 5.3 presents that in several scenarios, Literal2Feature shows good performance measured and reported by the F1 score. In Scenario one, Literal2feature beats Distmult being the second best alternative. In Scenario two, Literal2Feature and the other approaches are all on a lower level, while Literal2Feature was the best performing. In Scenario three, Literal2Feature was best but slightly better than the second-best alternative, FeGeLod. In this scenario, we measured head-to-head results, while it is essential to mention that FeGeLod used 247 features to reach the performance, as Literal2Feature outperformed this metric by using only eight features. In addition, we evaluated how the search depth resulting in a different number of features influenced the prediction. In Figures 5.5 - 5.8, we see a decreasing positive impact of higher search depth. At some point, the increased processing time of traversing deeper stops bringing important further features. In the clustering experiments, Literal2Feature achieved the best performance on the ENGIE dataset, and was close to second best on the carcinogenesis dataset where TransE and RDF2Vec performed better (see Table 5.4). The scalability experiments show that increasing processing power on a multi-node cluster reduces processing time (see Fig. 5.9). The reduction of processing time by scaling processing power decreases because of increasing Spark overhead and data shuffling across more executors. The dataset size scalability has been proven and visualized in Figure 5.10. In this Figure, we also present the effect of Random Walk versus full Breadth First Search, which shows faster execution of Random Walk.

## 5.4 Summary

With Literal2Feature, we introduce a novel approach for feature extraction on KG data. Its usage is explicitly interesting when KG data integration is performed and ML-relevant features are stored within literals. The approach supports the creation of feature-extracting SPARQL queries without deep knowledge of the KG data or the ontology. Literal2feature got applied in several use cases and built reused tools in other SANSAs ML approaches. The evaluation shows predictive performance in several scenarios, while implementation and integration in the Scala, Apache Spark and Apache Jena-driven SANSAs stack make horizontal scaling possible. The Scalability of Literal2feature showed promising results over scaling dataset and scaling processing power. Literal2Feature becomes one central feature-extracting tool for DistRDF2ML in Chapter 6 and builds one feature-extracting opportunity within SimE4KG in Chapter 7.

---

## Distributed ML Pipelines for RDF KGs

---

**Acknowledgement** This chapter is based on our scientific publication: *DistRDF2ML - Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs* **Carsten Felix Draschner**, Claus Stadler, Farshad Bakhshandegan Moghaddam, Jens Lehmann, and Hajira Jabeen. 2021. Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM). Association for Computing Machinery, New York, NY, USA [24], <https://doi.org/10.1145/3459637.3481999>. Further details can be found in Section 1.4, especially in Subsection 1.4.2.

### 6.1 Motivation

A large number of data-driven use cases are based on KG data. These KG data offer excellent opportunities for data integration through taxonomies and ontologies [134, 135]. Many openly available KG such as WikiData [136], YAGO [4], or DBpedia [98] are available in the LOD cloud<sup>1</sup>. These open available structured KG data are a valuable source for machine learning (ML) and data analytics. In addition to the structural information, the multi-modal values in the literals of RDF KGs are also relevant for several ML approaches. Since today's KGs can reach considerable data dimensions, single computers cannot process these Big Data KGs [14]. For other data structures, especially tabular data, frameworks like Apache Spark have been established to enable horizontal scaling through distributed computing. Apache Spark MLlib allows it to create desired data processing and ML approaches through modular pipelines [16, 42]. However, there is no way to run these Spark modules natively on RDF KGs [108]. In particular, many ML approaches require fixed-length numeric feature vectors. Since KGs not only contain multi-modal data but can also contain any number of features for each sample, creating fixed-length numeric feature representations is difficult. Approaches like knowledge graph embeddings (KGE) or graph kernels can create fixed-length numeric feature vectors [10, 63, 65]. Nevertheless, these approaches are not able to represent explainable multi-modal features [23]. Explainability means that the individual indices of a feature vector can be assigned to the original features. Using the feature-extracting SPARQL queries presented in Chapter 5, graph traversal can aggregate KG features. Easily reusable ML modules in Apache Spark MLlib make it straightforward for developers and researchers to create and use these downstream pipelines. Using

---

<sup>1</sup> <https://lod-cloud.net>

the DistRDF2ML approach, we created a way to transform manual-created or Literal2Feature-created SPARQL queries into fixed-length numeric feature vectors. These modules, integrated into the SANSa stack, allow literal feature-based downstream ML pipelines in the distributed execution environment of Apache Spark. The Literal2Feature evaluation outlined in Section 5.3 shows that SPARQL query-based extracted features are suitable for various ML use cases. In DistRDF2ML, we enable ML pipelines on explainable feature vectors using existing ML classifiers like Random Forest or Multi-Layer Perceptrons (MLP). Due to the complexity of the big RDF KG data computations, it is desirable to have the results reproducible and reusable. We improve the reusability and reproducibility by metadata and ML result semantification. This way, predictions can also be directly incorporated into the source KG. The contributions of this work are as follows:

- An open-source framework for distributed ML pipelines on RDF KGs
- Generic modules for the creation of explainable and context-preserving feature vectors
- Various sample ML pipelines documented and testable within Databricks<sup>2</sup> notebooks, executable sample classes, and unit tests
- A software architecture in which its semantic data ML pipelines are aligned with pipelines in established data analytics libraries
- Reproducibility over semantic annotation of pipeline metadata and processed results

The chapter is organized as follows: In Section 6.2, the DistRDF2ML architecture is presented as well as the concept of modules. Section 6.3 shows the evaluation of the framework's performance and scalability among multiple (hyper-) parameters, while Section 6.4 concludes the efforts.

## 6.2 DistRDF2ML Approach

In this section, we introduce the DistRDF2ML approach. Subsection 6.2.1 provides an overview of DistRDF2ML's pipeline architecture. In Subsections 6.2.2 - 6.2.5, we explain the major pipeline steps of SparqlFrame, SmartVectorAssembler, ML modules, and semantification. Subsection 6.2.6 presents the framework as a resource and mainly targets its Novelty, Availability, Utility, and Predicted Impact.

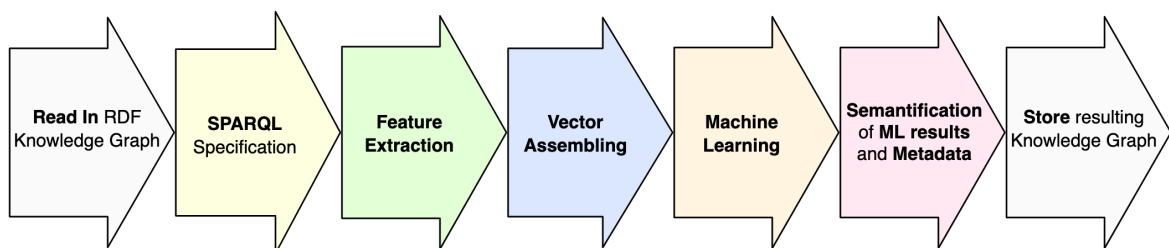


Figure 6.1: DistRDF2ML Pipeline Overview

<sup>2</sup> <https://databricks.com/>



### 6.2.1 Pipeline Architecture

The proposed DistRDF2ML framework offers the construction of end-to-end Apache Spark 3.x pipelines (see Figs 6.1 - 6.3). The principal pipeline contains the following modules:

- Knowledge graph reader
- SPARQL creation
- SparqlFrame feature extractor
- SmartVectorAssembler
- Spark MLlib machine learning
- Semantification of ML results and metadata
- Result exporter

**Read in Knowledge Graph:** Data is read-in over the SANSa framework within the RDF layer<sup>3</sup>. This layer supports multiple RDF formats and can read from Hadoop Distributed File System (HDFS<sup>4</sup>) to incorporate large-scale RDF data. Besides *n-triples*, SANSa also supports concurrent ingestion of the *turtle* and *trig* RDF formats.

---

```

1 SELECT
2 ?movie
3 ?movie__down_genre__down_film_genre_name
4 ?movie__down_title
5 ?movie__down_runtime
6 ?movie__down_actor__down_actor_name
7 WHERE {
8 ?movie <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
   data.linkedmdb.org/movie/film> .
9 OPTIONAL { ?movie <http://purl.org/dc/terms/title> ?
   movie__down_title .}
10 OPTIONAL { ?movie <http://data.linkedmdb.org/movie/runtime> ?
   movie__down_runtime .}
11 OPTIONAL { ?movie <http://data.linkedmdb.org/movie/actor> ?
   movie__down_actor . ?movie__down_actor <http://data.linkedmdb.
   org/movie/actor_name> ?movie__down_actor__down_actor_name .}
12 OPTIONAL { ?movie <http://data.linkedmdb.org/movie/genre> ?
   movie__down_genre . ?movie__down_genre <http://data.linkedmdb.
   org/movie/film_genre_name> ?
   movie__down_genre__down_film_genre_name .}}

```

---

Listing 6.1: Sample SPARQL query from hybrid usage of Literal2Feature and manual edit

<sup>3</sup> <https://github.com/SANSa-Stack/SANSa-Stack/tree/develop/sansa-rdf>

<sup>4</sup> <https://hadoop.apache.org/docs/>

### 6.2.2 SparqlFrame

**SPARQL creation:** In pipelines constructed with DistRDF2ML (see Figure 6.2), we offer three variants to create a SPARQL query that extracts relevant features. The first option is that a KG expert manually crafts the desired SPARQL query. The second option is to let the Literal2Feature [23] (see Chapter 5) module create a SPARQL query automatically. Literal2Feature is 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. The result of Literal2Feature is a SPARQL query that will extract the features. The third option is the hybrid approach, where first Literal2Feature is used to propose a query that is manually post-processed. The third option saves much time designing a syntactically clean SPARQL query and allows restriction to the most relevant features. A possible small sample feature extracting SPARQL query created by the hybrid Literal2Feature model is shown in Listing 6.1. The projection variable names are automatically generated and offer insights about how the respective feature is reached within the RDF KG.

**SPARQL based feature extraction:** SparqlFrame is a high-level transformer that allows for easy and intuitive execution of SPARQL queries on RDF KGs to create the Spark DataFrames that are suitable input to conventional ML pipelines. SparqlFrame bundles SANSa query execution engines (Sparqlify[17] and Ontop<sup>5</sup>) together with a novel schema mapper that was created as part of this work. While the execution of SPARQL queries in SANSa yields a Spark RDD of SPARQL bindings, the schema mapper analyses the set of bindings (especially the used data types) and computes a target schema and a mapping. Upon applying a schema mapping, the target schema becomes the schema of the resulting Spark DataFrame, whereas the mapping is used to convert each binding to a row in the target DataFrame. The schema mapper supports mapping XSD types<sup>6</sup> to corresponding Spark MLib ones and also allows for custom extensions. For example, if a feature query retrieves features about people such as their name, income, and birthday, the columns would be of type<sup>7</sup> StringType, DoubleType, and TimeStampType. If a feature has multiple annotated datatypes, a separate column is created for each data type, which is mapped in the column name. For every unknown RDF literal datatype, a variable is bound to a separate column and will fall back to StringType, receiving the lexical forms of those literals.

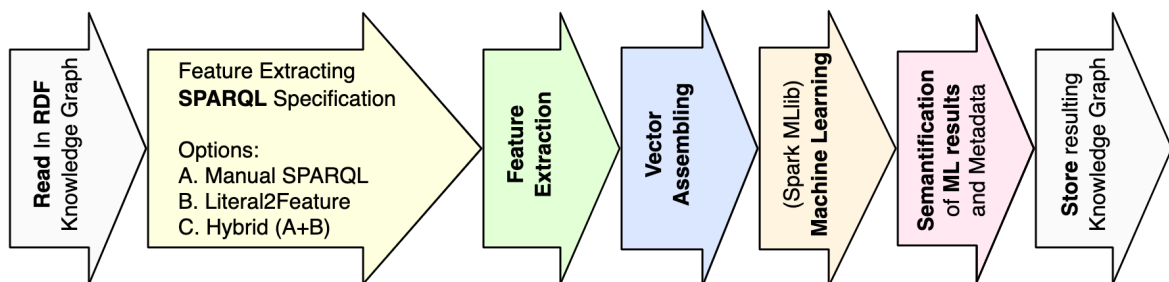


Figure 6.2: DistRDF2ML SPARQL specification options

<sup>5</sup> <https://ontop-vkg.org>

<sup>6</sup> <https://www.w3.org/TR/xmlschema11-1/>

<sup>7</sup> <https://spark.apache.org/docs/latest/sql-ref-datatypes.html>

---

```

1 val dataset: Dataset[Triple] = [...]
2 val sparqlString: String = [...]
3 val sparqlFrame = new SparqlFrame()
4   .setSparqlQuery(sparqlString)
5   .setCollapsByKey(true)
6   .setCollapsColumnName("movie")
7 val extractedFeaturesDf = sparqlFrame
8   .transform(dataset)
9
10 // semantic representation of a transformer
11 sparqlFrame
12   .getSemanticTransformerDescription()

```

---

Listing 6.2: Sample usage of SparqlFrame transformer

**Feature type identification and collapsing:** In the next step, a variety of feature characteristics are collected. These are relevant for the appropriate selection of digitization techniques. The features must be available for common ML approaches as fixed-length numeric feature vectors. The initial features that are not numeric must therefore be transformed into a numeric representation. For Strings, there are various approaches, such as string indexing or Word2Vec [70] transformation. The selection of the appropriate method depends on the feature characteristics. For example, it is relevant for text respective StringType columns, whether they are categorical elements or natural language elements. Due to the transformer, a DataFrame is available in which the relevant feature characteristics are also mapped in the column names, which are relevant for selecting the correct digitization strategy. It is also optional to generate a collapsed DataFrame. A collapsed DataFrame means that there is exactly one row in which all relevant features are contained for each sample. So there can also be columns of type Array[String], e.g., if we query features of a movie via SPARQL and a feature is the list of names of the playing actors. Regarding the feature characteristics, the following information is collected:

- *featureType*: A string representing the major information needed for the respective digitization strategy is also annotated within the column name aggregated from automatically retrieved feature characteristics.
- *nullable*: Whether this feature may be null.
- *datatype*: What is the datatype given by SparqlFrame feature extraction?
- *numberDistinctValues*: How many distinct values are given for this feature? This feature characteristic is important to decide, e.g., if a feature is categorical.
- *isListOfEntries*: This provides the information if a feature could be a list of information for a certain sample entity like the mentioned example of a list of actors.
- *availability*: The ratio of null values is important for some feature weighting techniques.
- *isCategorical*: Whether the feature qualifies as categorical. A heuristic is used to compute this attribute from the value distribution, the ratio of distinct values, and the overall dataset size.

The feature metadata is stored in an object that can be passed to other components of a processing pipeline.

### 6.2.3 SmartVectorAssembler

**Digitization and assembling of features:** This SmartVectorAssembler transformer converts all features corresponding to their feature type into numeric representations. These numerical representations are relevant for the required fixed-length feature vectors as input for standard ML models. For a variety of combinations of feature properties, corresponding transformation pipelines are implemented. The featureType, which decides over the follow-up SVA strategy, is gathered by SparqlFrame and is also denoted in the SparqlFrame output column name, which can also be manually set. For example, non-categorical Strings are transformed into the Word2Vec representation. Alternatively, lists of categorical Strings are transformed into lists of indices. A second example is the treatment of timestamp typed columns. These columns are digitized over representing by multiple digit columns transforming the *datetime* information into: *year, month, dayOfYear, dayOfMonth, dayOfWeek, hour, minute, second*. Especially enriching the feature vector with information like *dayOfWeek* offers further opportunities to predict periodic patterns. The type of transformation is noted in the column name. After this step, all feature columns are available in a numeric representation with column names containing the original feature name and the transformation strategy in brackets.

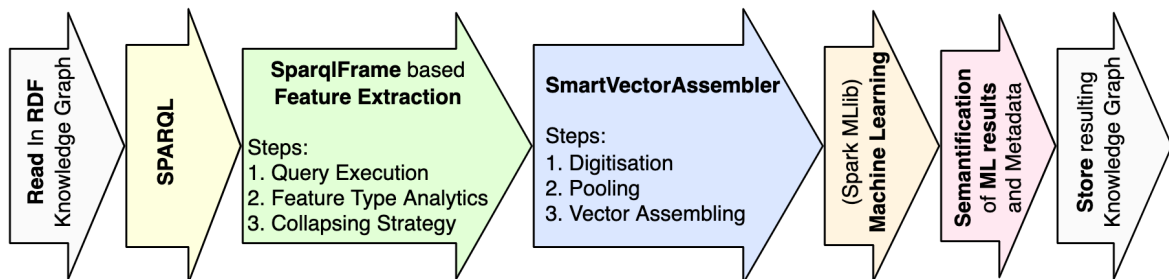


Figure 6.3: DistRDF2ML feature vector creation pipeline through SparqlFrame and SmartVectorAssembler

```

1 val smartva = new SmartVectorAssembler()
2   .setEntityColumn("movie")
3   .setLabelColumn("movie__down_runtime[...]")
4 val assembledDf: DataFrame = smartva
5   .transform(extractedFeaturesDf)
6
7 // explain the feature vector
8 val featureDescriptions = smartva
9   .getFeatureVectorDescription()
10
11 // semantic representation of transformer
12 val svaMetaGraph = smartva
13   .getSemanticTransformerDescription()
  
```

Listing 6.3: Sample usage of SmartVectorAssembler

**Pooling - Aligning number of incorporated features:** Some feature columns are available after the digitization step in variable-length numeric feature representations. The final numeric feature vector

for standard ML models must have a fixed length across all samples. Therefore, the strategy for the variable-length numeric feature is to aggregate them by *min*, *max*, *average*, and *stdev*, which are order-invariant pooling techniques. After this step, the DataFrame contains one row for each sample with a fixed number of numerical features.

**Vector Assembler:** After the Vector Assembler transformer, all numeric feature columns are combined into one explicit column, which is of type Array of Doubles. This assembled feature vector is the required representation for all following standard ML models. However, due to the unique preprocessing pipeline, every single value of the feature vector can be assigned to the original feature (see Listing 6.3 Line 7). This feature vector indices description enables explainability for ML pipelines over KG embedding-based feature vectors. The resulting DataFrame can be used natively in Apache Spark MLlib [16]. A sample resulting DataFrame from SmartVectorAssembler is shown in Listing 6.4.

```

1 +-----+-----+-----+
2 |          entityID | label |          features |
3 +-----+-----+-----+
4 | http://data.linke ... | 101|[ -0.01 , 9 , 3.34 , 8... |
5 | http://data.linke ... | 83|[ -0.06 , 6 , 9.72 , 4... |
6 | http://data.linke ... | 97|[ 0.027 , 5 , 5.16 , 0... |
7 | http://data.linke ... | 92|[ -0.00 , 7 , 7.11 , 4... |
8 | http://data.linke ... | 25|[ 0.021 , 2 , 2.92 , 0... |
9 +-----+-----+-----+

```

Listing 6.4: Sample output of SmartVectorAssembler

### 6.2.4 Machine Learning Models

In the distributed big data processing within Apache Spark, some libraries have been developed, which provide many needed ML models which can be configured over Scala and Python and executed on an Apache Spark cluster [16]. For the most common downstream ML tasks, we can use Apache Spark MLlib<sup>8</sup>. If we need the opportunities to use more complex neural networks than Multi-Layer Perceptrons, the framework BigDL [97] provides further artificial neural network functionalities<sup>9</sup>. Available ML models within the Spark MLlib 3.1 are for example: Logistic Regression, Random Forest, Multilayer Perceptron (MLP), and Linear Support Vector Machine, which can be used for Classification and Regression tasks<sup>10</sup>. The MLlib models can directly operate on the output of the DistRDF2ML pipeline (see examples notebooks [137]). This offers end-to-end ML pipelines running on Apache Spark clusters. Hence, DistRDF2ML is an enabler in processing large-scale RDF data through downstream ML tasks in a distributed fashion.

<sup>8</sup> <https://spark.apache.org/docs/latest/ml-guide.html>

<sup>9</sup> <https://bigdl-project.github.io/master/>

<sup>10</sup> <http://spark.apache.org/docs/latest/ml-classification-regression.html>

### 6.2.5 Semantification of Results:

The ML computations of the DistRDF2ML framework are available in tabular form through the native Spark MLlib algorithms. For prediction tasks, a table is created containing the URI of the sample in the *entityId* column (see Listing 6.4) and the corresponding annotation in the prediction field. In the case of similarity estimations, two columns are used for the pair of URIs and one for the corresponding similarity value. Our ML2Graph module can be used as a Transformer. It creates an RDF KG based on these tabular results using *NodeFactory* from *Apache Jena*. The resulting graph can be exported in a standard RDF data format to complete the initial input KG with the predictions (see Figure 6.4). The Transformers provide semantic representations of their hyper-parameter configuration, enriching the ML pipeline results with metadata that helps to reproduce results and support interpretation. They serve as hints for further ML pipeline optimization.

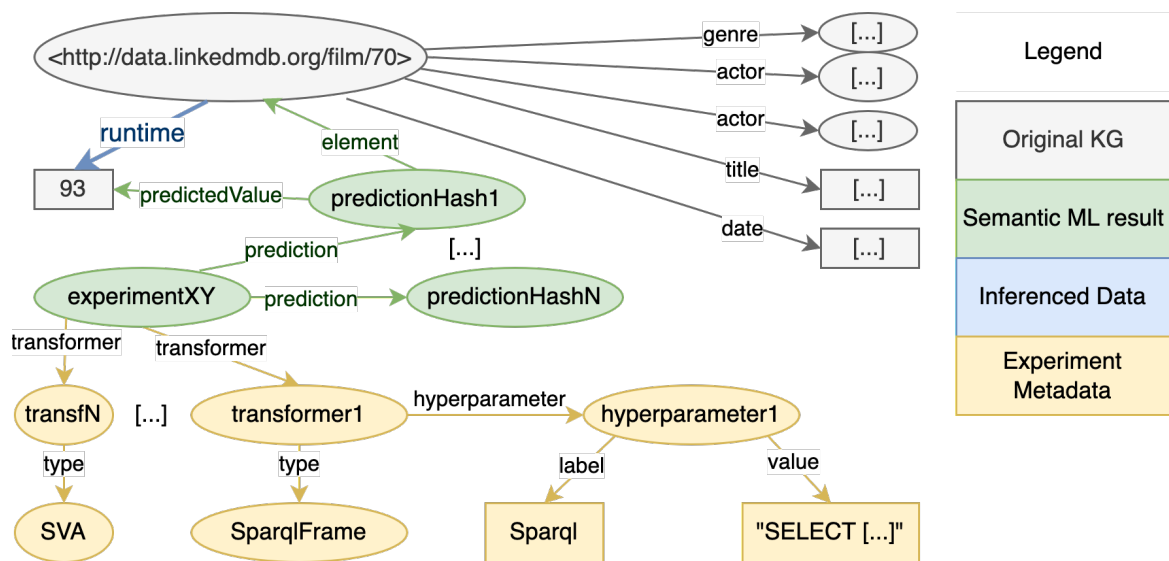


Figure 6.4: Semantically annotated ML result in original KG

### 6.2.6 DistRDF2ML as a Resource

The DistRDF2ML framework proposes modules to preprocess RDF KG data for state-of-the-art artificial intelligence, data mining, and data analytics pipelines. The framework is fully available for the community as an open-source GitHub release [137] and as an extension of the holistic SANSa stack to materialize big RDF data. DistRDF2ML enables various domains to empower their distributed Apache Spark clusters to process ML pipelines on RDF data. The performance and scalability of the framework have been evaluated on multiple (hyper-) parameter setups to present the advantages of the efficient data processing pipelines in data-intensive computing (see Section 6.3). The lack of existing extensions for processing RDF data within Spark MLlib pipelines was a significant motivation to develop this framework. The feature extraction over semi-automated query execution by `Literal2Feature` [23] and `SparqlFrame` modules enables non-native semantic developers to construct explainable feature extraction pipelines. The feature and knowledge extraction based on queries and processed through the `SmartVectorAssembler` allows multi-modal content-preserving representations

of feature vectors. Within various partner projects, the framework shows its enabling nature to create ML pipelines for multiple domains using a few lines of code .

---

```

1 val prediction: Dataframe = someMLmodel
2   .transform(df)
3 val ml2Graph = new ML2Graph()
4   .setEntityColumn("entityID")
5   .setValueColumn("prediction")
6 val metagraph: RDD[Triple] = ml2Graph
7   .transform(prediction)
8 metagraph
9   .saveAsNTriplesFile("/out/put/path")

```

---

Listing 6.5: Semantification of ML results

**Novelty** DistRDF2ML brings for the first time a generic preprocessing pipeline of RDF KGs to the Apache Spark world. The modules are easy to use as transformers and create explainable feature vectors. The extended SANSA stack provides end-to-end semantic ML pipelines for RDF KGs. This framework combines multiple complex technology stacks, such as semantic data processing, distributed scalable computing, generic ML, and data science pipelines.

**Availability** All the introduced building blocks of the DistRDF2ML pipeline are integrated into the SANSA framework within the ML layer<sup>11</sup>. The framework is also published as a Release<sup>12</sup> with attachments like the fat jar and data for the evaluation. DistRDF2ML is integrated into the SANSA Stack, an open-source GitHub repository under Apache 2.0 license (see SANSA licensing).

**Utility** The modules are fully documented on the code level<sup>13</sup> but also on how-to-use level<sup>14</sup>. The modules are implemented as a transformer to align with the structure of Apache Spark MLlib. All of the parameter adjustments have to be made on transformer level [137]. These settings are shown in Listings 6.2, 6.3. We made multiple sample pipelines available as an example class or as a Databricks Notebook [137] such that the try-out hurdle is majorly reduced. The common pipeline-interface should offer users from the three domains: semantic web developers, data scientists/ML engineers, and Apache Spark data analysts, an easy structure to build desired ML pipelines. The central idea is that this framework acts as an enabler between different technological worlds. We have appended the referred data and fat jar directly to the corresponding DistRDF2ML GitHub release for the reproducibility of results.

**Predicted Impact** The DistRDF2ML framework allows Data Scientists and Semantic Web developers to implement their ML pipelines with few lines of code. Apache Spark supports scaling them among Big Data requirements. This resource combines the strengths of multiple domains and offers an

<sup>11</sup> <https://github.com/SANSA-Stack/SANSA-Stack/tree/develop/sansa-ml>

<sup>12</sup> [https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1\\_DistRDF2ML](https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1_DistRDF2ML)

<sup>13</sup> <https://sansa-stack.github.io/SANSA-Stack/>

<sup>14</sup> <https://github.com/SANSA-Stack/SANSA-Stack/blob/develop/sansa-ml/README.md>

interface between them. This resource is a part of the SANSa stack, which is constantly being developed, and additional features are being added. The framework has six-month release cycles. All modules are covered by unit tests executed for every pull request within GitHub actions s.t. problems or errors become apparent fast and are resolved by the SANSa development team. With its usability over Databricks, the framework targets a large group of developers who want to develop end-to-end Spark ML pipelines operating on RDF KGs. With the increase of RDF data in the context of data integration and the rise of KG dataset sizes, a scalable framework offering accessible, hands-on opportunities to develop ideas directly within the SANSa framework is an important resource contribution.

**Use Cases** Generic scalable distributed preprocessing pipelines for RDF KGs are needed in almost all standard downstream ML and data analytics tasks which have to be operated on explainable feature vectors. The EU Horizon 2020 project PLATOON (Platform for Tools in Energy)<sup>15</sup> uses this framework to analyze large-scale energy RDF data analytics tasks. The database of the PLATOON project was made available for data integration from many data sources in RDF, which especially contain timestamp, categorical and numerical features. The data is of a large scale that individual systems can no longer process. Many use cases require convenient learning pipelines that cover various predictive models like regression and classification. Moreover, the project Simple-ML<sup>16</sup> plans to use the framework DistRDF2ML for processing RDF datasets. The significant advantage is the easy alignment of interfaces to other libraries like scikit-learn and the native operation on RDF data. SANSa is also an integral part of the data analysis in the Opertus Mundi<sup>17</sup> project. In addition, there is an application in the field of accident analytics in the Smart City area, in which the technologies presented here are used to prepare considerable amounts of data for further processing of standard ML models.

## 6.3 Experiment and Evaluation

### 6.3.1 Data Description

For the evaluation and reproducibility of DistRDF2ML usage, we use the openly available Linked Movie Database (LMDb) [6], which introduces a movie KG with information about movies, titles, runtimes, actors, genre, producers, origin country information, and much more. This LMDb data is interesting because it shows the high diversity of possible multi-modal data within a KG, like the natural language String of the title and categorical lists of Strings representing the movie genres. Alternatively, even the country's population, the movie's runtime as digit features can be incorporated into data analytics and ML pipelines. Figure 6.5 shows a small extract from the KG as a plot (extract from N-Triple file are available in the release dataset section [137], and List. 7.3).

### 6.3.2 Description Evaluation Dimensions

A significant aspect of this contribution is the usage of Apache Spark to allow distributed execution of ML pipelines starting from semantic data. This utilization of Spark offers scalability in memory and

---

<sup>15</sup> <https://platoon-project.eu>

<sup>16</sup> <https://simple-ml.de>

<sup>17</sup> <https://www.opertusmundi.eu>



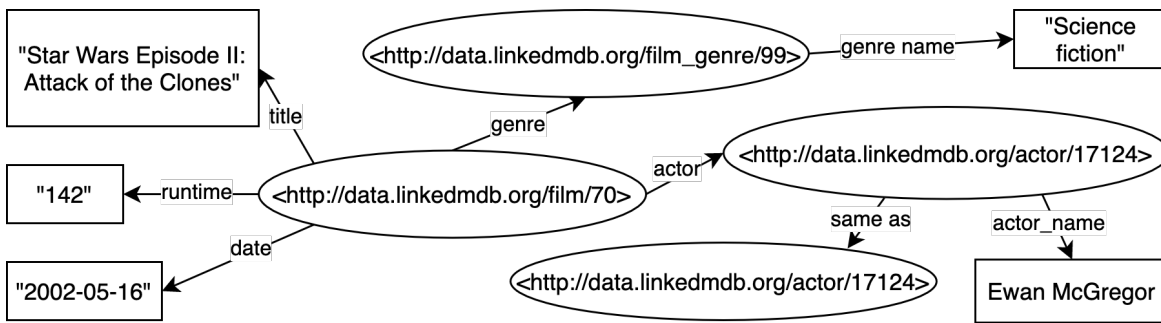


Figure 6.5: Sample graph snippet from Linked Movie Database

processing power, which would not be available in single-system implementations. Also, processing power, the feature extracting SPARQL complexity, and Spark cluster configuration influence the needed processing time. In this section, we offer an overview of the effect of these different (hyper-) parameters on the respective processing time. The plots in Figures 6.6 - 6.9 present the processing times of the most complex software modules as stacked bar charts. The green lower part of each full bar corresponds to the SparqlFrame transformer, and the blue upper part corresponds to the SmartVectorAssembler. The height of the bars always corresponds to the execution time in seconds.

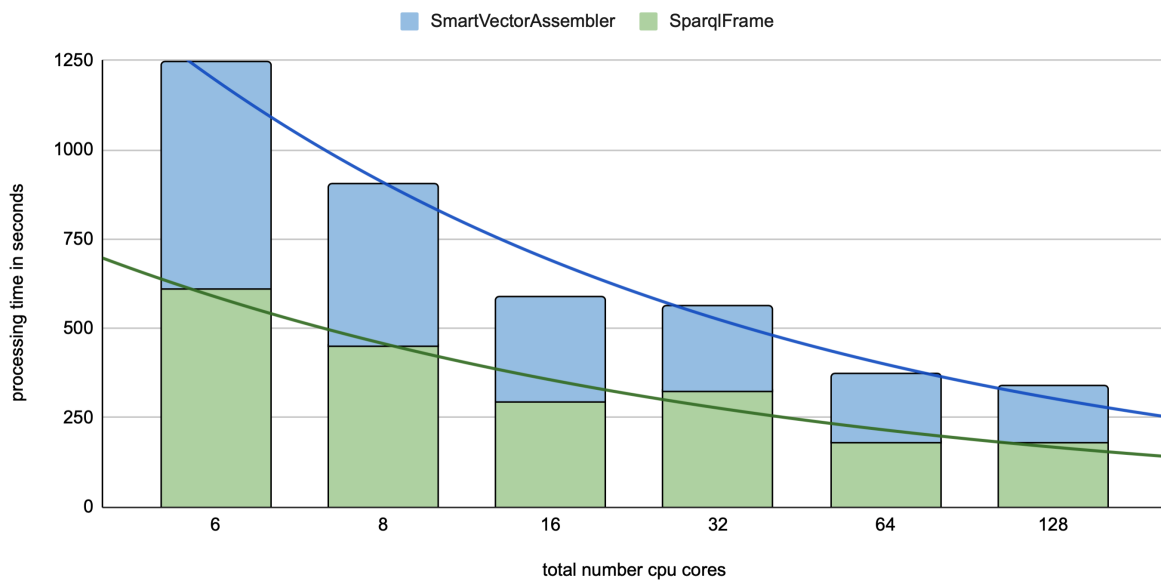


Figure 6.6: Processing power vs processing time

### 6.3.3 Processing Power vs Processing Time

The following paragraph shows how the available computational resources majorly influence the processing time. The substantial decrease in processing time over an increase in processing power allows high scalability among higher-performing clusters. For use cases where the processing time is critical, it is possible to reduce the processing time by increasing the number of executing cores,

showing the highly effective parallelism and distributed approach of DistRDF2ML.

### 6.3.4 SPARQL Complexity vs Processing Time

The starting point of creating the feature vectors is the respective SPARQL query. This query could have a considerable variety of complexity. The complexity is adjustable by the number of projection variables and the number of features. Additionally, the complexity is influenced by the traversal depth to reach a specific projection variable and the type of used feature. In the example of a linked movie database, a movie has assigned an arbitrarily long feature list of *actors' names* and a single-digit feature of the movie's *runtime*. The feature processing steps for a variable length string feature like the actors are more memory-intensive than the single digits representation of the *runtime*. We also compared the local execution of this pipeline to the cluster setup. Even in the more minor SPARQL queries, we perceive how fast the cluster usage outperforms a local Computer (6-Core, 16GB Ram, MBP 16 2019). All of the used hyper-parameters are available on the GitHub page as all of the code is available fully open source to allow a higher reproducibility of the experiments and find respective recommendations for other use cases [137].

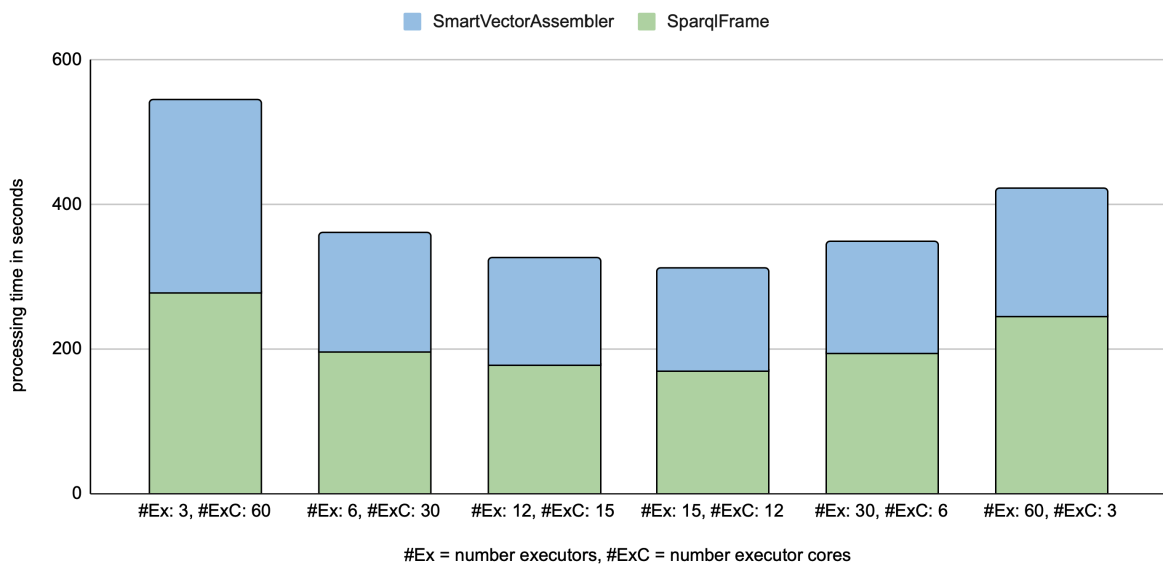


Figure 6.7: SPARQL complexity vs processing time

### 6.3.5 Dataset Size vs Processing Time

DistRDF2ML is also evaluated in terms of the scalability of different data sizes. This evaluation includes both local executions on a single local consumer notebook (6-Core, 16GB Ram, MBP 16 2019) and distributed on a three-node Apache Spark cluster (each having 64 cores and 256GB RAM). The data was generated synthetically compared to the other experiments that were based on LMDB. The synthetic datasets ensure that the data's exponential growth is achieved with consistent data and feature density. The evaluation shows that DistRDF2ML scales even with exponentially increasing data sizes. Also, we measured that for small data up to  $10^3$  movies, a local execution is

faster than a distributed execution. This behavior is because the data size does not justify the overhead of network communication and distributed execution management. However, from  $10^4$  movies, cluster computation offers not only performance-wise advantages but also from  $10^5$  movies, there are advantages of having sufficient main memory available that prevents out-of-memory problems (see figure 6.8). The case of out-of-memory was due to the sheer number of unique features that had to be indexed via Word2Vec embeddings and label encoding. The memory load can be reduced by setting min counts to greater than 1. We decided to use the worst edge case to measure an upper bound. The data used for the evaluation can be downloaded from the framework release page. The interval in which data sizes are visualized was chosen based on breakpoints. These breakpoints are the transition points at which cluster computation is superior both performance and RAM-wise to the local execution.

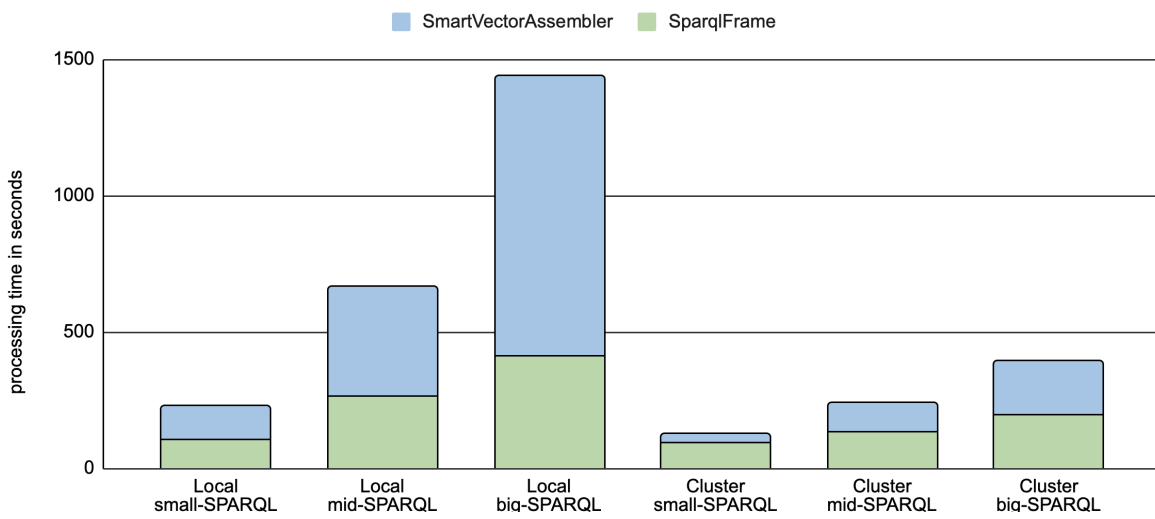


Figure 6.8: Dataset (-size) and number movies on cluster and local execution vs. processing time

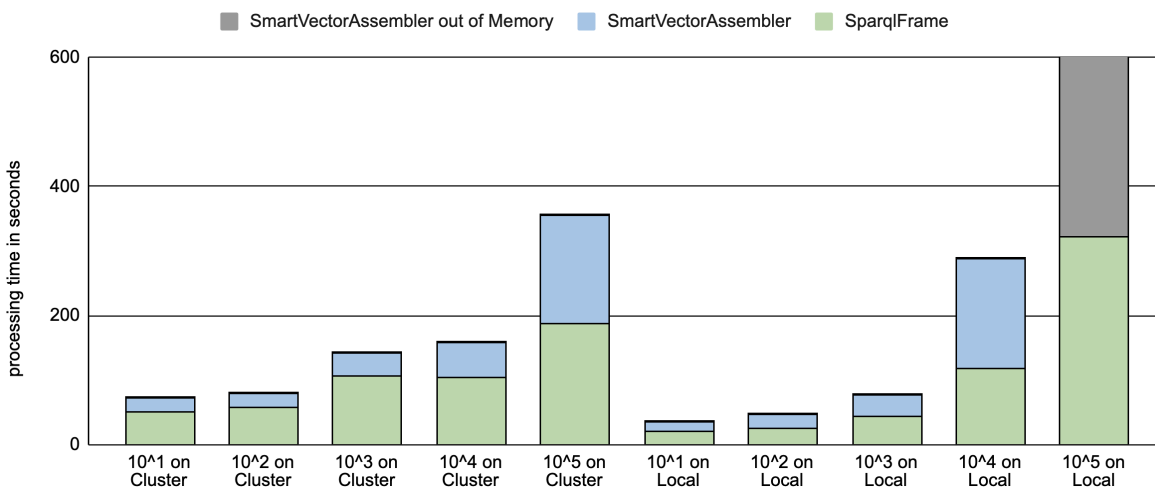


Figure 6.9: Spark setup vs processing time

### 6.3.6 Spark Setup vs Processing Time

In the Spark setup, we can configure the distribution of executors and their assigned executor cores and executor memories over the spark-submit. In a cluster of nodes, each having 64 cores and 256GB RAM, we can allow the Spark to have up to 60 cores and 240GB RAM. We do not use all capacities to allow background processes and Spark overhead to use the remaining capabilities. Now we have to decide which adjustment executors we want to make. The two edge cases are 3 Executors, each having 60 cores and 240GB of RAM, called fat executors, or we can go for the other edge case of 180 executors, each having one core and 4GB of RAM. Figure 6.9 depicts the optimization of execution time by balancing between fat and micro executors (best configuration in fourth bar: 15 executors, each having 12 executor cores).

### 6.3.7 Sample DistRDF2ML Pipelines

For demonstration purposes, we developed ML pipelines on RDF KGs and made those available within the SANSA stack GitHub repository and over Databricks notebooks [137]. Within the documentation, we propose the following:

- Generic Random Forest Regression ML pipeline to predict LMDB movie runtimes
- Generic RF Classification ML pipeline to predict LMDB movie genres
- Generic K-Means Clustering ML pipeline to group LMDB movies

### 6.3.8 Discussion

**Advantages of distributed processing:** In the experiments, we observe that the development and use of Apache Spark as one fundamental backbone offer high scalability far beyond the capabilities local computers can achieve. The significant advantage, in addition, is that the same code can be used on any Spark cluster instance and will make use of the available resources. In Figure 6.7, we have shown the first-hands-on guideline to optimize the spark setup, which can improve performance.

**Limits of local computation:** In a multitude of experiments in which we compared local and distributed cluster computation, we measured that the CPU processing on the cluster scales far better than local execution. Additionally, the cluster can easily be configured with more RAM and can handle much more complex in-memory processing tasks without getting out-of-memory issues.

**Scalability of approach:** The experiments show the scalable nature of Apache Spark and its modules. We stuck to Apache Spark design principles when creating the transformers. This programming principle paid off with excellent scalability behavior among various (hyper-) parameters (see Section 6.3).

**Generality of approach:** Within Section 6.3.7, DistRDF2ML shows its various application opportunities as an enabler and glue between large-scale RDF KGs and existing numeric feature-based Apache Spark MLib ML approaches. The usage of DistRDF2ML within the examples of *Clustering*, *Classification*, and *Regression* shows how easily applicable and adjustable these transformers are.

These transformers will majorly decrease the entry hurdle for creating baseline Apache Spark RDF ML pipelines. If the transformer does not suit the users' purpose, it can be used as starting point for further development because of its fully open-source nature.

## 6.4 Summary

DistRDF2ML provides the first end-to-end Apache Spark framework of generic modules to create explainable ML pipelines for KGs. DistRDF2ML can cover the requirements of a large set of use cases. The strengths lie in the high-level interface via transformers to build an intuitive creation of explainable feature vectors for ML models. By incorporating modules of the SANSa stack, Apache Spark and Apache Jena, DistRDF2ML operates natively on large-scale RDF datasets. Also, by providing the `Literal2Feature` function, the entry-level into the creation of feature extracting SPARQL is majorly simplified. We have evaluated the scalability of the approach. Through the results of experiments, we observed that the processing time is reduced by increasing resources (number of CPU cores, see Figure 6.6). Also, a distributed environment with increased memory allocation can significantly process more extensive datasets (see Figure 6.8). The variety of documentation from Scala docs cover example pipelines to the mapped functionalities in the unit tests allows a quick start in using the framework [137]. The alignment to the standard interfaces of Spark MLlib and transformer (see Listings: 6.2, 6.3) enables working with large-scale RDF data possible for many data scientists with less experience with working on semantic, linked open data, or Apache Spark. The availability of open source also makes necessary customization by users possible.



---

## Distributed Explainable Multi-Modal Similarity Estimation

---

**Acknowledgement** This chapter is based on our scientific publication: *SimE4KG: Distributed Explainable multi-modal Semantic Similarity Estimation for Knowledge Graphs*, Carsten Felix Draschner, Hajira Jabeen, Jens Lehmann, 2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), DOI: 10.1109/AIKE55402.2022.00007 [27]. Further details can be found in Section 1.4, especially in Section 1.4.2. This published paper got additionally invited by the IJSC Journal and is submitted and under review at the point of thesis submission.

### 7.1 Motivation

Many applications in data analytics and machine learning (ML) are based on entity pairwise similarity estimation. On the one hand, data-optimizing algorithms like entity disambiguation, entity resolution, and unsupervised classification can require semantic similarity estimation. On the other hand, some algorithms produce results to improve the accessibility of content or offer recommendations like clustering or recommendation systems. Interesting data sources for these analytic pipelines are heterogeneous knowledge graphs (KGs). The integrated KG data are multi-modal and require type-specific similarity estimations. In Chapter 4, we introduce DistSim, which offers scalable similarity estimation for RDF KGs. DistSim’s feature extraction handles all features as categorical features. The advances in property-based differentiation and handling of multi-modal features developed in Chapter 5 and Chapter 6 do not apply to the existing DistSim similarity estimators. Literal2Feature’s KG features exploration and DistRDF2ML’s fixed-length numeric feature creation allow explainable multi-modal feature vector generation. Treating features with modality-specific similarity estimation is advantageous, which is difficult on DistRDF2ML’s assembled feature tensors. Feature-specific similarity estimations can improve the interpretability of results. It is desirable to have explainable results. This explainability can guide adjusting the data processing to the desired outcome instead of being confronted with an ML black box. Further, the user should be capable of influencing the ML result to the desired behavior. Due to the sheer size of RDF KGs, which in the era of big data no longer fit into the main memory of classical consumer devices, and due to the quadratic complexity of *all-pair-similarity* (APS), the solution calls for generic, scalable distributed similarity estimation frameworks. The advantage of distributed computing architectures is that they allow

horizontal scaling of processing power. Another performance challenge is that the Literal2Feature and DistRDF2ML feature extraction can be time-consuming due to the distributed execution of SPARQL to SQL rewriting on large KGs. Providing a semantic data processing framework with good documentation and open source is crucial since this ensures transparency and adaptability. An easy-to-use open-source framework is missing that offers explainable and adjustable semantic similarity estimation for multi-modal KGs, simultaneously capable of scaling horizontally to a multi-node cluster.

The main contributions of this work are:

- Semantic similarity estimation SimE4KG algorithm for multi-modal RDF KGs.
- Novel generic distributed in memory downstream ML modules within the open-source SimE4KG framework, which is specially designed to scale with heterogeneous KGs. [138].
- Introduction of multi-dimensional weighting techniques of domain-aware similarity scores.
- Semantification of similarity estimation improves the explainability, reusability, and reproducibility, completing the original RDF KG.
- Documentation within ReadMe, Scala Docs, and sample pipelines with accessible notebooks, integrated into the holistic Semantic Analytics Stack SANSA [139].

The chapter is organized as follows: In Section 7.2, the SimE4KG architecture and the concept of modules are presented. Section 7.3 shows the framework's performance and scalability evaluation, while Section 7.4 concludes the efforts.

## 7.2 SimE4KG Approach

This section gives insight into the structure of the SimE4KG framework and the computation of new generic modules within the pipeline for scalable distributed domain-aware multi-modal semantic similarity estimation.

### 7.2.1 Pipeline Architecture

The SimE4KG framework execution is performed by a pipeline of generic newly developed modules. Figure 7.1 shows the structure of the pipeline. In the Sections 7.2.2 - 7.2.6, the approaches of the single steps are described, and technical insights into the implementation are given. The pipeline consists of seven steps (see Fig. 7.1). Initially, the data is Read-In. The relevant entities of the KG that should be within the similarity assessment can then be specified. Subsequently, a probabilistic approach is used to calculate promising candidates. Extensive and detailed features are extracted from the KG for these promising candidates. Similarity scores are calculated and accumulated for the promising candidate pairs based on the multi-modal features and weights. The results of the semantic similarity estimation are transformed into a semantic native form, including the pipeline metadata like the hyper-parameters. Finally, the RDF KG data is stored.



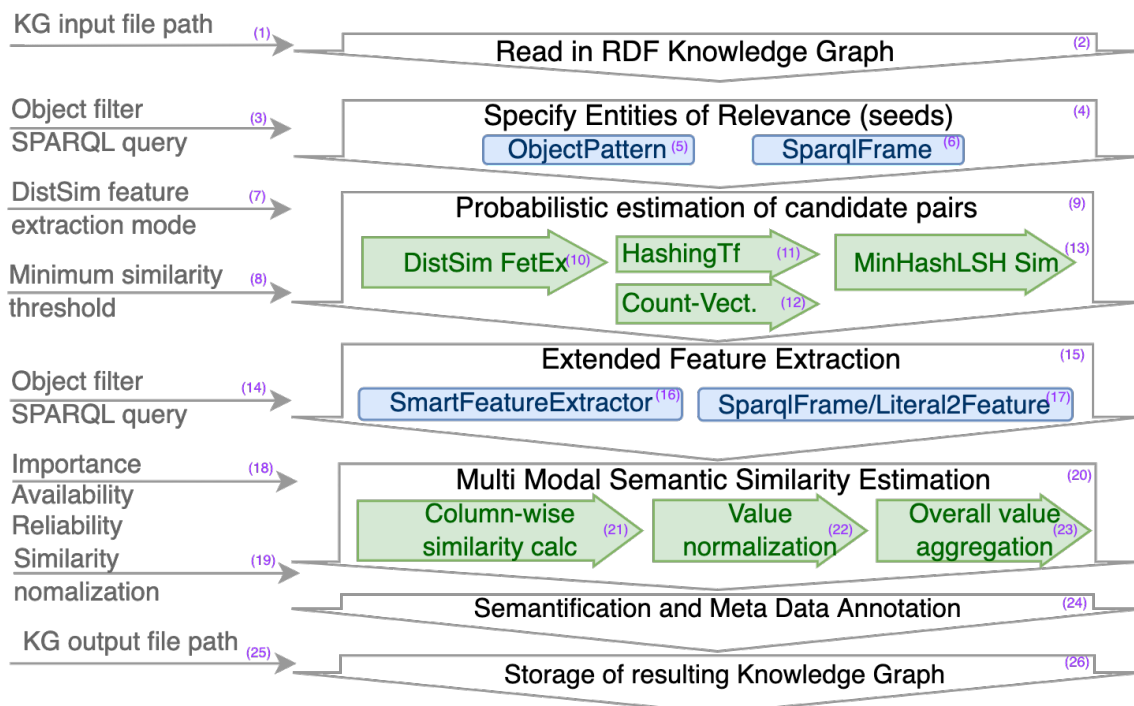


Figure 7.1: SimE4KG pipeline overview

**Pipeline Architecture - Technical Implementation Details** Many new modules were developed during the technical implementation based on existing widely used libraries and frameworks. Data management is implemented by the Hadoop File System (HDFS). All pipeline modules use Apache Spark [13] for distributed execution and Apache Jena [15] for native handling of Semantic RDF KGs. The reading and writing of RDF data are implemented by the SANSa RDF layer [14]. The initial optional filtering of the entities is implemented by the triple pattern matching options or the SparqlFrame Transformer [24]. The probabilistic gathering of candidate pairs reuses the DistSim [22] pipeline (see Sect. 7.2.2). For feature extraction, the significantly faster SmartFeatureExtractor was developed as an alternative to the existing SparqlFrame/Literal2Feature [23, 24] Transformer (see Sect. 7.2.3). The multi-modal semantic similarity estimation and the semantification of the results were implemented as new Transformers (see Sect. 7.2.4 - 7.2.5 and List. 7.1), which are aligned to the well-known libraries, e.g., Apache Spark MLlib and scikit-learn [16, 20].

### 7.2.2 Probabilistic Collection of Candidate Pairs

SimE4KG reads in the RDF KG by using the existing SANSa RDF layer API [14] (see Fig. 7.1 (1,2)). Afterward, the KG is available as a Spark Dataset of Jena Triples. Since the complexity (see Fig. 7.7 top left) of multi-modal and domain-aware similarity scores has, in principle, a quadratic running time, the probabilistic feature-based similarity score is performed based on the DistSim approach (see Sect. 4.2.3). The feature aggregation is optimized to the Map-Reduce operations of Apache Spark native operating on Apache Jena Triple data of the KG. Different options offer choices in which triple data should be considered as features. In our pipeline, we proposed majorly collecting associated nodes

for the first guess of overlapping features. The whole space of twelve available modes is described within DistSim [22] (see Sect. 4.2.3). For this feature extraction, the relevant entities are initially determined by a filter in the KG (see Fig. 7.1 (3-6)). A SPARQL query or a triple filter can be used based on the object, such as a concrete *rdfType*. SPARQL queries are executed by the SparqlFrame module of SANSA ML, which reuses SPARQLIFY a SPARQL to SQL rewriting executable within Apache Spark [17, 24]. The alternative object filter method is faster (see Evaluation and Fig. 7.10) but allows less complexity for the specification. For the filtered entities, features are extracted using the DistSim feature extractor. These features are transformed into numerical feature vectors using two different options. The options are either the HashingTF or the CountVectorizer of Spark [13, 16]. HashingTF has faster performance but lacks explainability of created vectors which is acceptable for first guesses compared to later full feature space similarity assessment. For the selected entities, the upper triangular matrix of similarity scores is computed and filtered by a minimum threshold of entity pair similarity. The similarity scores are generated using the MinHashLSH Apache Spark implementation [95] (see Fig. 7.1 (7-13)). Since the DistSim approach treats all KG features as categorical features within the MinHashLSH, this step is only used as a first guess to reduce the complexity of the multi-modal estimations (see Sect. 7.2.3 & 7.2.4). The result is the basis for all entity pairs to be computed in the multi-modal semantic similarity score procedure.

---

```

1 val dataset: Dataset[Triple] = spark.rdf(...)
2 val SimE4KG-Estimator = new DaSimEstimator()
3   .setObjectFilter(...)
4   .setDistSimFeatureExtractionMethod("os")
5   .setSimilarityValueStretching(true)
6   .setImportance(Map("actor_sim" -> 0.2[...])
7 val similarityDf: DataFrame = SimE4KG-Estimator
8   .transform(dataset)
9 similarityDf.show()

```

---

Listing 7.1: Example code usage of SimE4KG module

### 7.2.3 Feature Extraction

Based on the feature extractor implementations in the SANSA stack (*FeatureExtractorModel*, *SparqlFrame*), a feature extractor Transformer *SmartFeatureExtractor* was developed that combines the strengths of both. The speed is increased by using the parallelizable and efficient pivot function in Apache Spark [13], but also the resulting DataFrames are collapsed so that for each sample, a row exists [24] (see Fig. 7.1 (14-17) and Fig. 7.2 (1-3)). Collapsing means that for the arbitrary number of associated features for a KG entity, feature elements are grouped by their RDF predicate. This grouping leads to the fact that features can be single Strings or numbers but also a list of arbitrary entries. The columns are automatically annotated to the literal types [140] mounted when given so that native operations can be performed on the given values (see Table 7.1 and Fig. 7.2 (3,4)). If RDF literal XSD-type annotation is missing, the fallback is String which can be manually changed to the desired casting. This predicate-based and multi-modal feature extraction significantly improves the approach from DistSim. The similar functionality of the *SparqlFrame* requires a call to the underlying SPARQLIFY [17, 23] framework, which introduces significant time complexity, although this is essential due to the higher variability of a SPARQL query in certain use cases. So, for most use cases,

the new *SmartFeatureExtractor* enables faster (see Sect. 7.3) but also domain-aware and multi-modal feature extraction in contrast to the DistSim feature extractor model [22] and SparqlFrame Feature Extractor [24]. The *SmartFeatureExtractor* can be used outside the SimE4KG domain as it provides a convenient and distributed collection of RDF KG feature data. The *SmartFeatureExtractor* results are native Apache Spark Dataframes that can be further processed with the available Spark MLlib transformers to build desired ML pipelines.

```

1  val dataset: Dataset[graph.Triple] = [...]
2
3  /** Smart Feature Extractor */
4  val sfe = new SmartFeatureExtractor()
5    .setSparqlFilter("SELECT ?mov WHERE { [...] / film > .}")
6
7  /** Feature Extracted DataFrame */
8  val featureExtractedDF: DataFrame = sfe
9    .transform(dataset)
10 featureExtractedDF.show()

```

Listing 7.2: Example code usage of SmartFeatureExtractor

*Example:* In the case of the Linked Movie Database (LMDb) RDF KG, which we used for evaluation and demonstration purposes, we faced a wide variety of feature types (see Fig. 7.3). The feature types reach from a single short string representing the movie title. A list of URIs represents the movie's actor cast. This cast is an arbitrary long list of entries (see Table 7.1). The release date is annotated as timestamp, but due to missing annotation of XSD-type, we assign the Dataframe column type in a subsequent SimE4KG pipeline step (see sample SimE4KG Databricks notebooks) [138].

entity (URI)	runtime (Double)	actors (Array[URI])	release date (Timestamp)	title (String)	fN ...
https://.../film1	141.0	[htt//.../a4, ...a8]	2002-01-01	Catch Me If y...	...
https://.../film2	142.0	[htt//.../a1, ...a2]	1994-01-01	The Shawshan...	...
https://.../film3	189.0	[htt//.../a1, ...a4]	1999-01-01	Green Mile	...
https://.../filmN	...	...	...	...	...

Table 7.1: Sample multi-modal result SmartFeatureExtractor

### 7.2.4 Domain Aware and Multi-Modal Similarity Estimation

For a final assignment of a similarity score, individual feature-specific similarity scores are calculated based on all extracted features (see Fig. 7.1 and Fig. 7.2 (3-5)). Matching similarity scores are automatically assigned and used for different multi-modal feature types. For example, for categorical feature lists such as an arbitrary number of genre annotations, similarity scores are computed using the Jaccard index [22, 141]. A normalized distance across the whole feature distribution is calculated for continuous values like runtime. For lists of timestamps, a normalized mean distance is calculated as the similarity score based on a transformed *unix\_timestamp* representation of these features (see Fig. 7.2 (4) green and blue). The normalization supports domain-aware similarity estimations as data distribution can be different across the various features. Thus, all features are assigned a similarity

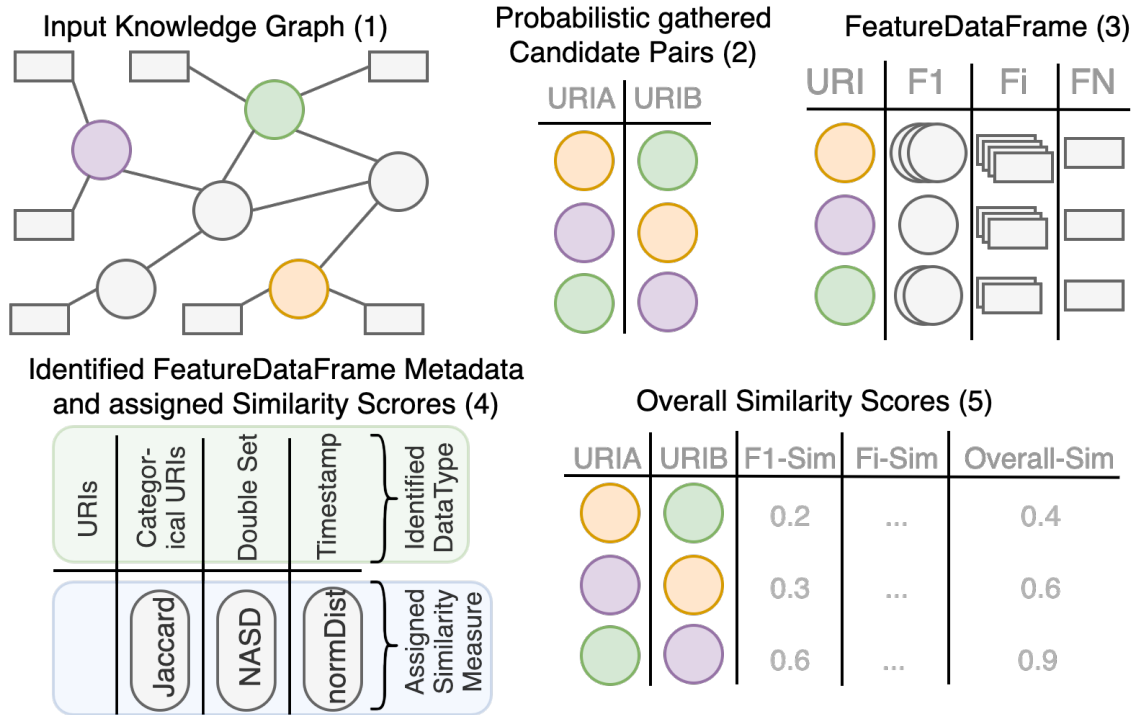


Figure 7.2: SimE4KG data transformations

score between 0.0 (no similarity) and 1.0 (exact similarity). Subsequently, all similarity scores can be normalized (see Listing 7.1) again to carry domain awareness correctly on the characteristics of the different KGs and similarity scores (see Fig. 7.2 (5)). The supported and mapped similarity scores can handle the following multi-modal KG data types: Boolean, String, TimeStamp, Double, Int, URI/IRI [13, 15]. An intermediate SimE4KG pipeline step provides all calculated similarity scores separated across feature type (see Fig. 7.2 (5)). These feature-specific similarity scores are used for the final similarity score assessment but also for the explainability information.

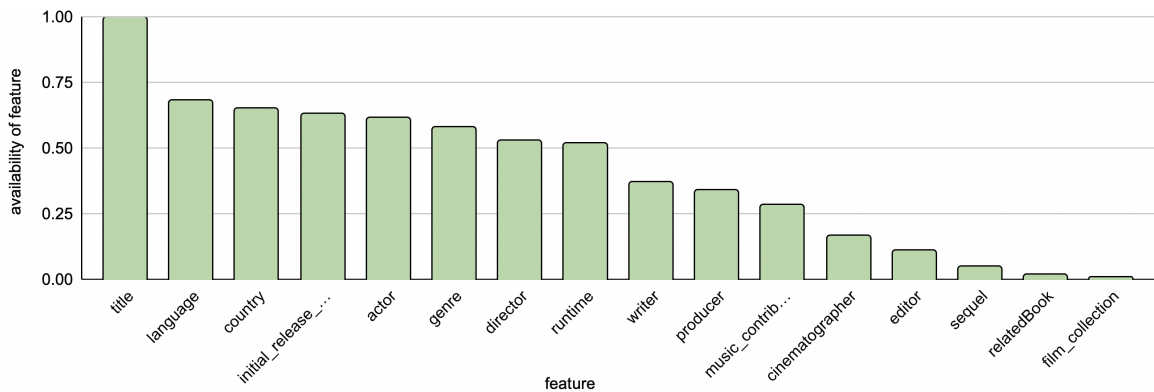


Figure 7.3: Extract of feature availability within LMDB dataset of type movies

### 7.2.5 Weighting of Features and overall Similarity Assessment

The decided feature similarities are aggregated in a final overall similarity score based on a weighted sum. Likewise, the most significant feature similarities for explainability are collected. However, individual similarity scores can also be weighted. Several weights can be specified by the user as parameters or are automatically derived from a holistic data analysis of the KG. The weights are the following:

**Importance** With the optional importance parameter, the user can give SimE4KG a personal preference for the relevance of features. For example, if this pipeline is an implementation of a recommendation system in the field of a movie database, the user can assign increased importance to concrete feature similarities such as a high similarity of genre and actors (see List. 7.1 line 6). The importance is optional; if it is not specified as a parameter, all features have equally distributed importance. If the importance is given, the respective values must sum up to one.

**Availability** The availability weight is automatically calculated. The non-null, non-empty, or non-nan values accumulate to the availability ratio for each feature. It is used in the case of weighting similarity scores when a score is calculated from two entities concerning a concrete feature. In some cases, the feature value is not provided for one of the entities. Due to the open-world assumption, this can always happen (see Fig. 7.3). However, if, in principle, a feature is often null (low availability), a low similarity in this feature should not negatively impact the overall similarity score.

**Reliability** It describes the likelihood of the information within a feature being correct; if a particular feature is less reliable, its respective similarity score can be reduced in the final influence of the overall result. The basic idea is that KGs are created by merging and integrating many data sources. The initial data sources building the data integrated KG are differently obtained, differently volatile, and differently ensured to be up to date. In contrast to availability, reliability must be handed over as a parameter because it needs expert knowledge about the data sources not stored within the actual data and not each user can provide.

### 7.2.6 Semantification of Results and Meta Data

The SimE4KG pipeline provides end-to-end RDF KG data handling (see Fig. 7.1). The results of the similarity estimations can be output as native RDF data. Not only the resulting similarity scores for pairs of entities are mapped (see Fig. 7.4, blue) but also other metadata like the hyper-parameter setup of the pipeline (see Fig. 7.4, green and yellow). This data is provided for the results' reusability, reproducibility, and explainability. Here the explainability is the annotation of the most influential factors (one or multiple), e.g., for *film1* and *film2*, the *most influential factor* is the overlapping *writer* (see Fig. 7.4 in yellow). Furthermore, the reproducibility is supported by including the hyperparameters of the whole pipeline. The reusability is made possible because the results coexist natively in the original KG (see Fig. 7.4 in grey) and can thus be included in queries or further analyses. The reusable semantic native RDF KG allows further semantic data analytics native through SPARQL queries. The enrichment of the source KG with similarity information empowers data analytics that needs similarity estimation but does not require an initial SimE4KG execution.

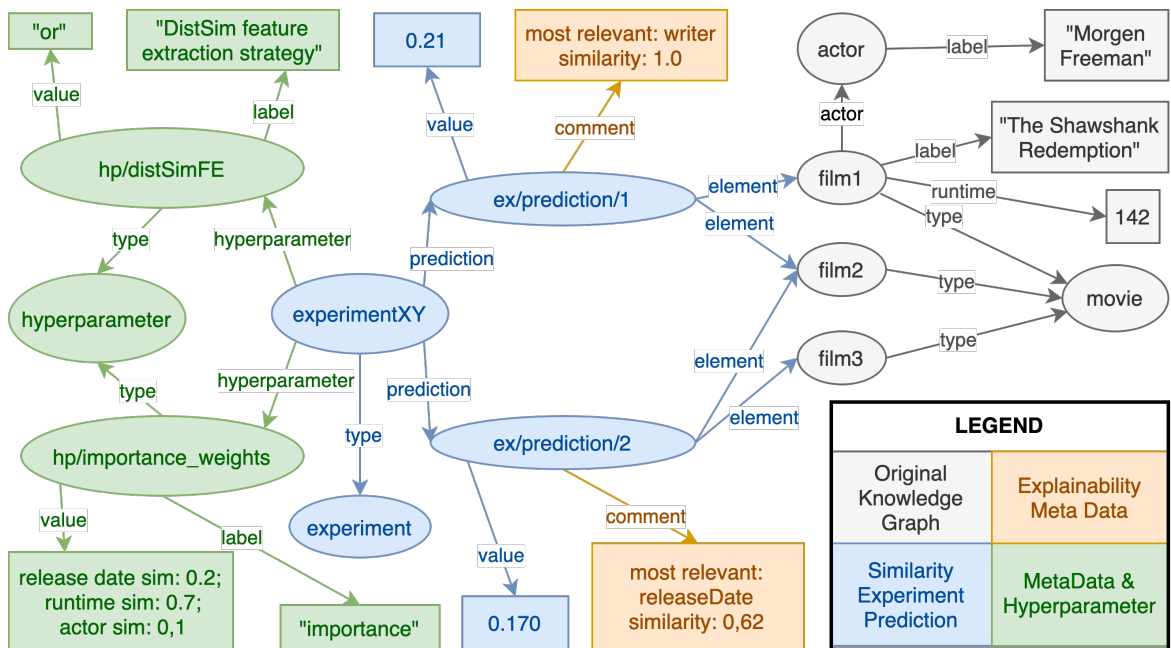


Figure 7.4: SimE4KG semantification example

### 7.2.7 Impact & Use Cases

The SimE4KG approach and framework have the potential for impact in the Semantic Web, Linked Data, Knowledge Graph, Data Science, and ML community and are already part of some use cases.

**Potential Impact:** SimE4KG is an interesting resource for the AI/ML, Knowledge Engineering, and Semantic Web community because it offers scalable and multi-modal semantic similarity estimation as an easy-to-use framework integrated into SANSa. All generic modules are adjustable and usable outside the intended context to create arbitrary downstream ML pipelines for RDF KGs. Due to its usability, this source offers a solution to use KGs and ML results for tools like recommendation systems. Also, the metadata encourages the reproducibility of ML experiments. It makes it easier to use semantic data cause of its distributed scalable approach. Working with generic transformers, which are well established in libraries and frameworks, also offers many data scientists access to all the interesting semantic data sources. By semantification of the ML pipeline parameters and results, we introduce how downstream ML and similarity estimation pipelines create native semantic results. These results become explainable and reproducible. This representation also offers many data integration advantages to the source KG. The resource improves and widens the opportunities of already developed frameworks like the DistSim approach that could not produce explainable results and operate on multi-modal data. At the same time, the user can set parameters like the presented weighting.

**Use Cases:** Generic, scalable, distributed semantic similarity estimation pipelines for RDF KGs are needed in several follow-up ML and data analytics tasks. The EU Horizon 2020 project PLATOON

(Platform for Tools in Energy<sup>1</sup>) uses this framework to analyze large-scale energy RDF data. The database of the PLATOON project was made available for data integration from many data sources in RDF, which especially contain multi-modal features within the literals. The data is of such a large-scale that individual systems can no longer process it. Another project, Simple-ML<sup>2</sup>, develops on top of the generic data analytics options of SANSA a convenient (low-code, no-code) opportunity to stack ML approaches. The SANSA stack supports the data analysis in the Opertus Mundi<sup>3</sup> project.

### 7.2.8 Quality, Reusability, and Availability

We provide our novel SimE4KG framework with a dedicated focus on quality, reusability, and availability.

**Reusability:** The resource is documented by ReadMes, where the framework’s ideas, thoughts, and evaluations are presented<sup>4</sup>. All novel modules are documented by scala docs. We provide Hands-On tutorials for sample pipelines within the framework and supply sample Databricks notebooks. Due to the open-source and generic pipeline development approach, the framework is extensible. Within multiple sources like README, GitHub Pages, and Databricks sample notebooks, we guide the application of the SimE4KG framework.

**Design and Technical Quality:** The resource modules are developed generically and are aligned with standard downstream ML pipeline modules like scikit-learn [20] and Apache Spark MLlib [16] transformers where all (hyper-) parameters are set over setters (see List. 7.1). The maintenance is supported by the Scala-docs, and unit tests are automatically called by GitHub actions. The ReadMes and Databricks Notebooks elaborate the downstream tasks to assist the use of specific modules to perform intended or adjusted similarity estimations. SimE4KG is integrated into the SANSA framework, and it reuses modules from the related approaches: DistSim [22], DistRDF2ML [24], SPARQLIFY [17] and Apache Spark MLlib MinHashLSH [95]. All the frameworks and data used within this tool, all the tools which are used, are openly available and mostly open source. The framework is evaluated in multiple domains, offering performance metrics in data size and processing power scalability, and presents the advantages and disadvantages of the novel introduced modules. The entire resource is available within the open source SANSA GitHub repository<sup>5</sup>. All modules are merged over a pull request and available within the linked SimE4KG SANSA release<sup>6</sup> [138]. The framework is available under the Apache License 2.0<sup>7</sup>. The framework goes through release cycles twice a year, and novel modules are earlier available in pre-releases.

---

<sup>1</sup> <https://platoon-project.eu>

<sup>2</sup> <https://simple-ml.de>

<sup>3</sup> <https://www.opertusmundi.eu>

<sup>4</sup> <http://sansa-stack.github.io/SANSA-Stack/>

<sup>5</sup> <https://github.com/SANSA-Stack/SANSA-Stack>

<sup>6</sup> <https://github.com/SANSA-Stack/SANSA-Stack/releases/>

<sup>7</sup> <https://www.apache.org/licenses/LICENSE-2.0>

## 7.3 Experiment and Evaluation

In this section, we evaluate the performance and characteristics of the proposed SimE4KG framework. SimE4KG is evaluated in the dimensions of scalability over increasingly complex pipelines (dataset size and hyperparameters) and the scalability of the approach over distributed cluster architectures. The newly developed approaches, such as feature extractor and similarity estimation, are compared with the distributed approaches from related work. All evaluations are presented here, and some further details are documented on our release page. The evaluations were performed on a three-node cluster, with each node having 64 cores and 256 GB RAM, configured with Spark 3.x, Scala 2.12, and Java 11. In addition, we ran our experiments on a Macbook Pro 16 (2019) with dataset sizes and hyperparameter configurations that did not exceed the available memory. The available benchmarking and evaluation scripts support reproducing the presented processing times. The experiments are evaluated on the multi-modal Linked Movie Data Base (LMDB) KG [6] resulting in  $10^9$  potential candidate pairs (see Fig. 7.5 and Fig. 7.7).

### 7.3.1 Dataset Description

The linked movie database (LMDB) is a multi-modal RDF KG describing detailed information about movies [6]. The associated features are, for example, actors (list of entities), genre (entities), movie ids (Integers), origin country (geo-information), producers (entities), runtimes (floating number), and titles (short NLP string). We selected the LMDB because of its multi-modal nature (see List. 7.3 and Fig. 6.5). In addition, we selected the dataset because similarity estimations on movie data can form a baseline for further recommendation systems or clustering, which are commonly needed analytical tasks for streaming and multi-media data. In contrast to other open available KGs, the LMDB is majorly clean in its data annotation. We only saw some weaknesses in the lack of annotation of XSD types (see List. 7.3 Line 1, 4). We present an application of SimE4KG's modules operating on the multi-nature of the LMDB dataset in the Databricks notebooks linked to the release [138].

---

```

1 <ht [...] db.org/film/70> <ht [...] date> "2002-05-16" .
2 <ht [...] db.org/film/70> <http://www.w3.org/2000/01/rdf-schema#label> "
   Star Wars Episode II: Attack of the Clones" .
3 <ht [...] db.org/film/70> <ht [...] #type> <ht [...] /film> .
4 <ht [...] db.org/film/70> <ht [...] /runtime> "142" .
5 <ht [...] db.org/film/70> <ht [...] /country> <ht [...] country/US> .
6 <ht [...] db.org/film/70> <ht [...] actor> <ht [...] 17124> .
7 <ht [...] db.org/actor/17124> <ht [...] actor_name> "Ewan McGregor" .
8 <ht [...] db.org/film/70> <ht [...] genre> <ht [...] db.org/film_genre/99> .
9 <ht [...] db.org/film_genre/99> <ht [...] film_genre_name> "Science
   fiction" .

```

---

Listing 7.3: Sample RDF snippet from Linked Movie Database

### 7.3.2 Dataset Size vs. Processing Time

The use of distributed technologies results from the fact that semantic data and KGs can exist in data sizes that exceed the available memory on local computers. However, since memory provides a fast



way to process data, the capabilities of distributed in-memory cluster computation are a significant advancement. In this evaluation (see Figs. 7.5, 7.6, 7.7), we show the scalability of SimE4KG over increasing dataset sizes. In Figure 7.5, we present that the quadratic complexity of all pair similarities is reduced by two orders of magnitude using the MinHash LSH approach (from  $10^9$  to  $10^7$  similarity scores for entity pairs). Figure 7.7 shows that the framework scales linearly among, in principle, quadratic increasing candidate pairs computed on the cluster (the 30k+ unique entities already correspond to  $10^7$  selected candidate pairs and  $10^9$  potential candidate pairs, see Fig. 7.5). The Spark overhead is also proportionally reduced in more extensive dataset size scenarios (see Fig. 7.6). We also evaluated the scalability of local execution between a linearly increasing number of movie entities and increasing quadratic candidate pairs (see Fig. 7.5). Additionally, the almost linear scaling is observed among quadratically increasing problems due to the MinHashLSH approach and the usage of Spark Cluster computation.

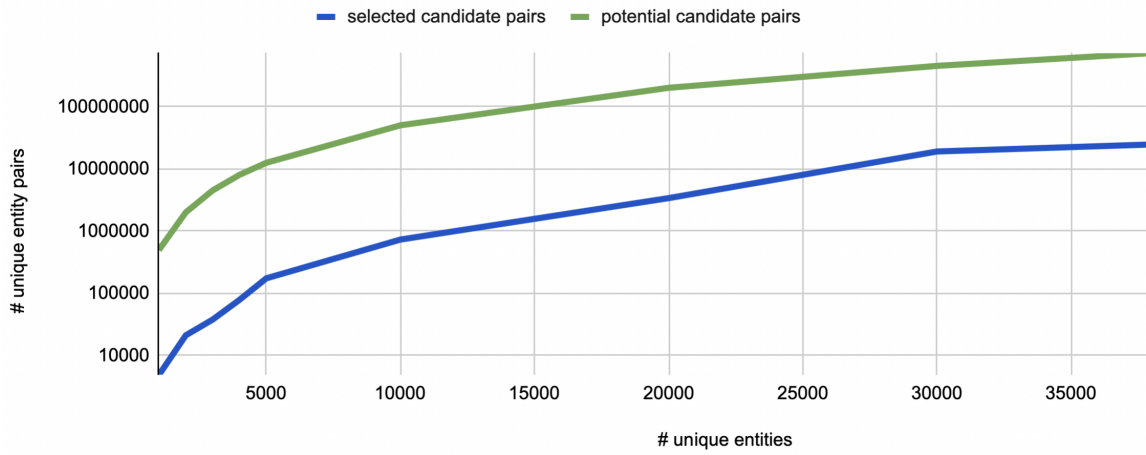


Figure 7.5: Evaluation dataset size scalability LSH effect

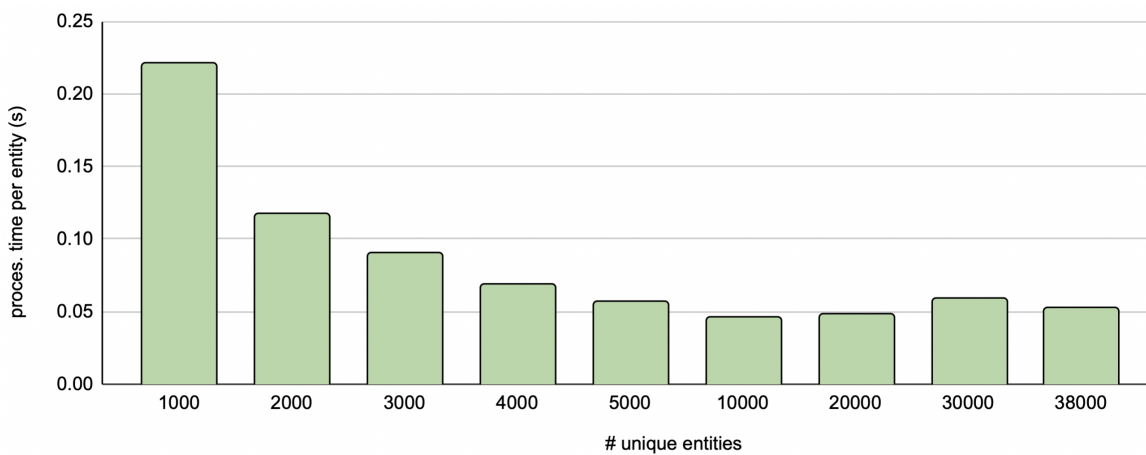


Figure 7.6: Evaluation of Spark Overhead by Dataset Size

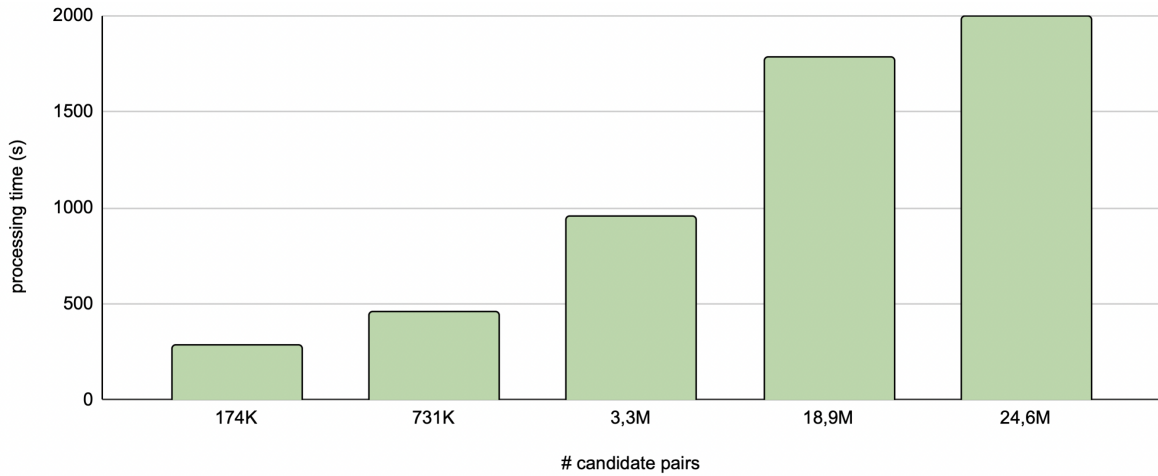


Figure 7.7: Evaluation SimE4KG dataset size scalability on LMDB subsets

### 7.3.3 Processing Power vs. Processing Time

The additional performance on cluster computation with the increased number of computation units (cores) can only be used by parallelizable and efficient algorithms. To evaluate the scalability of performance over increasing computational power, we configure our cluster with different numbers of executor cores so that a reduction of the processing time becomes transparent with constant LMDB KG data. Figure 7.8 shows how the processing time can be reduced with more processing power. In addition, we would recommend a well-balanced Spark cluster setup to even bring down the processing time with the same overall processing power (see Fig. 7.9). The more executors a cluster has, the more the implementation can be executed in parallel. Still, all those parallel jobs need to be managed and synced by a driver, creating an overhead. Our observation in Figure 7.9 shows that having multiple executors (more executors than nodes) but each having still quite a lot of processing power results in faster processing.

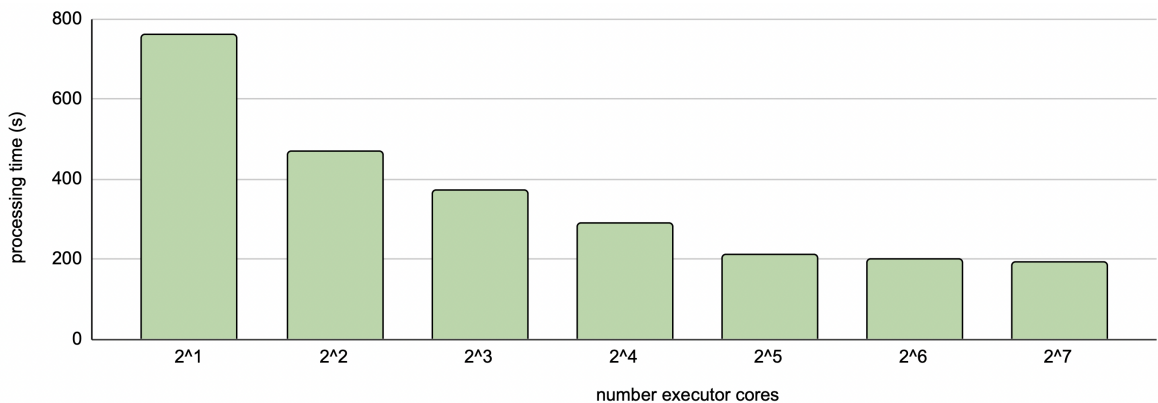


Figure 7.8: Evaluation processing power scalability

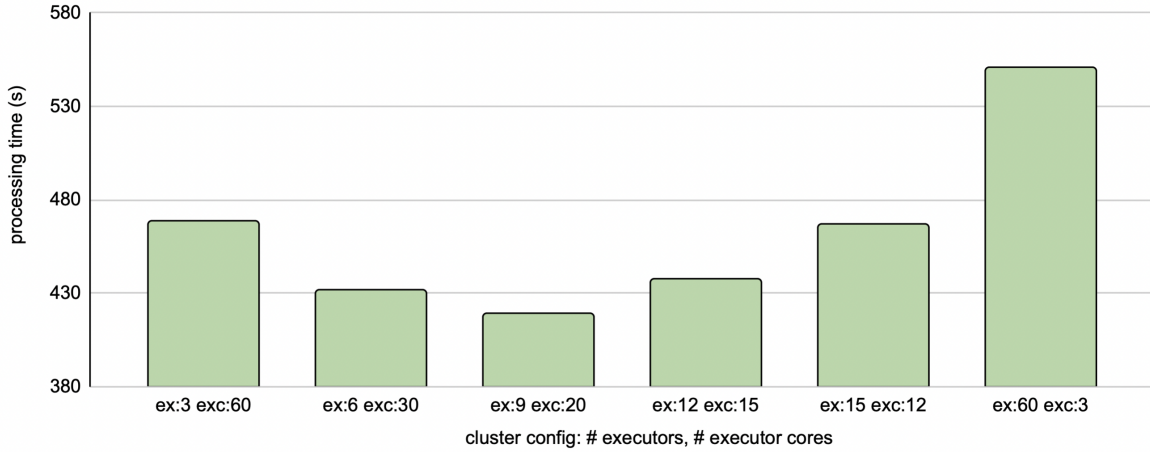


Figure 7.9: Evaluation spark cluster configuration effect

### 7.3.4 Comparison of SimE4KG to Distsim and DistRDF2ML

In contrast to existing DistSim frameworks, SimE4KG can deal with the multi-modal nature of RDF KG and produces explainable results that the user can better understand. Moreover, the ideas of the SPARQLIFY-based SparqlFrame are extended for the generic feature extractor. These and other modules have been tested through (hyper-) parameter evaluations. Table 7.2 presents that the novel SmartFeatureExtractor outperforms the SparqlFrame feature extractor in the full 26 features (projection variable = PV) with one layer depth. So if the extended feature extraction (see Fig. 7.1 (15,16,17)) is only interested in the features of distance one in the KG, the SmartFeatureExtractor from SimE4KG should be selected. If a precise and deep traversal of the KG structure is needed, the existing SparqlFrame can be used. In addition, the initial seed gathering (see Fig. 7.1 (4,5,6)) has been extended and evaluated. For the 3,5M triples containing information about roughly 40K entities of type movie, the initial time for collection of seed entities could be reduced from 12.99 seconds using the SparqlFrame, to 1.74 seconds with the object pattern matching based on the Apache Spark/Jena Triple matching (The processing time measures shown in Fig. 7.10 result from local execution on MP16-2019).

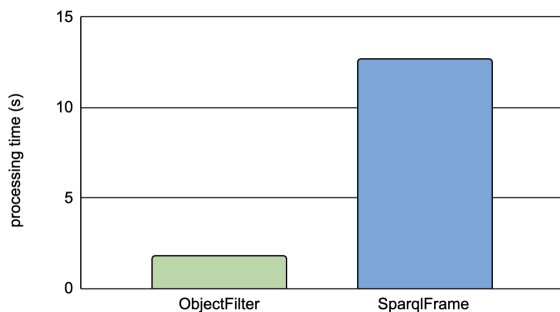


Figure 7.10: Evaluation and Comparison to available SANSa modules

#PV	#FeatureEx	prcstime(s)
3	SparqlFrame	41
4	SparqlFrame	49
26	SmartFeatureExtractor	50
5	SparqlFrame	55
6	SparqlFrame	66
10	SparqlFrame	123
26	SparqlFrame	1638

Table 7.2: Smart Feature Extractor vs SparqlFrame. Hyperparameter: lmbd, MBP16, 38kMov

### 7.3.5 Discussion

The evaluation has shown that the developments of SimE4KG are both scalable for increasingly complex data and scale well horizontally over increasing resources. This scalability enables more complex similarity estimations using the framework SANSA. The integration and documentation of SimE4KG and its scalable performance provide a reasonable basis for follow-up ML or data-analytic pipelines in SANSA. Based on the good results of Literal2Feature and DistRDF2ML, we observed that feature-based KG ML could produce great ML predictions [23, 24] (see Chapter 5 & 6). A benchmark dataset for large-scale RDF KG semantic similarity estimation for multi-modal data is missing. The annotation of human-perceived semantic similarity estimation scores is challenging to determine [72, 73], which is outside the scope and focus of the technical-driven approach. The pipelines can be used for implementing recommendation systems, entity matching, or clustering. In the multi-dimensional evaluation, it becomes clear that the various feature extractors of DistSim, DistRDF2ML, and SimE4KG differ in complexity, both in the set of possible features and their runtime. In the present use cases, the newly developed feature extractor compromises the fast but non-differentiated collection of features and the various configurable but computationally intensive SPARQL-based feature extractors. The design of the similarity estimator allows far more influence by the pipeline's user. Also, SimE4KG can cover many more possible multi-modal KG data types. There are fast probabilistic methods as well as more accurate non-probabilistic algorithms. Also, the improved annotation of the similarity estimations allows an increased explainability of the created results.

## 7.4 Summary

This work presents the SimE4KG framework integrated into the semantic analytics stack SANSA. SimE4KG is an explainable, scalable, distributed multi-modal in-memory semantic similarity estimation approach for RDF KGs. The similarity assessment is performed by a pipeline of generic, easily configurable software modules. All of them are aligned with the common usage of ML pipeline transformers, offering standard usability. In addition, the entire similarity estimation is optimized to fit the multi-modal nature, and the results are presented in an explainable, reproducible, and reusable native RDF format. The extensive evaluation of framework performance shows the significant scaling of distributed computing. The entire framework is open-source and available through the corresponding GitHub repository, while the documentation in the form of a tutorial like ReadMe, Scala-docs, hands-on example classes, and Databricks notebooks is presented. Due to the generic nature and the integration into the SANSA stack, SimE4KG can be extended, and the individual software modules can be used outside the initially intended usage.

## Implementation and Technical Setup

Chapters 4, 5, 6, and 7 present the algorithmic and technical approaches of DistSim, Literal2Feature, DistRDF2ML, and SimE4KG. These include feature-based data analytics and learning for RDF KGs. This chapter presents how these are standardized and made available as resources. Also, two sections show how these newly created tools and frameworks can be tested in Databricks and Zeppelin notebooks and integrated via RestAPI. Section 8.1 presents how we technically deployed the projects. Section 8.2 shows how the technologies can be tested and used in the browser through the PaaS provider Databricks. Section 8.3 shows how the technology can be deployed on-site in Zeppelin notebooks or as a Rest API architecture.

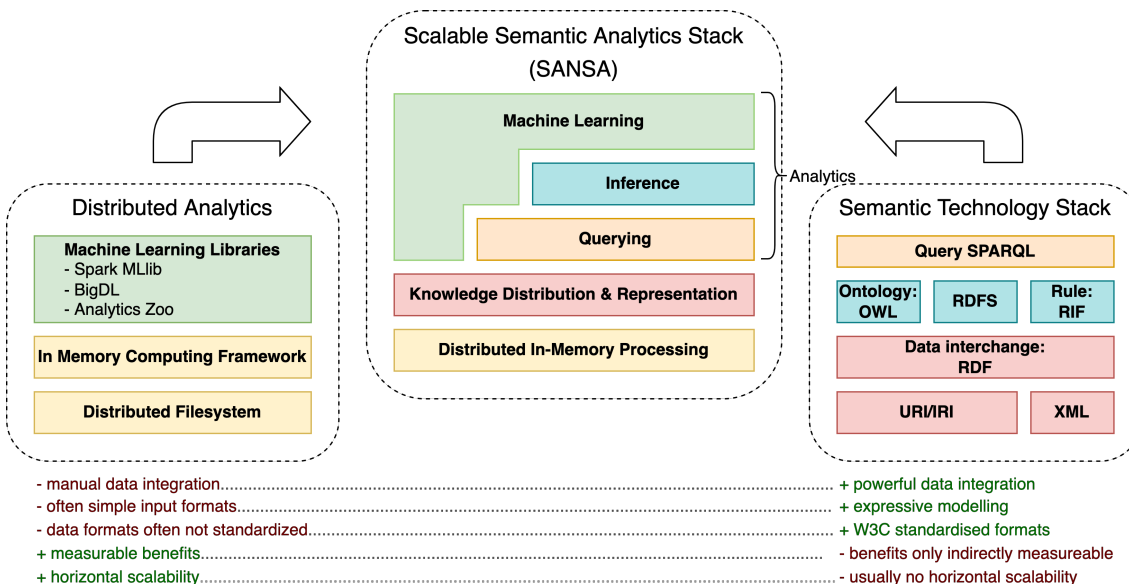


Figure 8.1: SANSA stack structure

## 8.1 Resources

Each project developed within this thesis has been created and published in an explicit schema. The technical components are fully open source and can be found in the GitHub repository. The documentation can be found on the website `sansa-stack.net` and in the repository. The repository documentation is mapped on the GitHub pages, in ReadMes, and in the corresponding release notes (see Figs. 8.2 - 8.4).

### 8.1.1 GitHub Repository

All of the work can be found in the GitHub repository of the SANSa stack<sup>1</sup>. The GitHub repository includes several projects. The major project is the central SANSa stack. This central repository replaces the historical projects separated by layer. Additional projects include the technical implementations of Zeppelin Notebook, Databricks, and Rest-API. The central repository provides documentation within the ReadMes and on the GitHub pages. Also in the GitHub repository the unit tests are executed through GitHub actions.

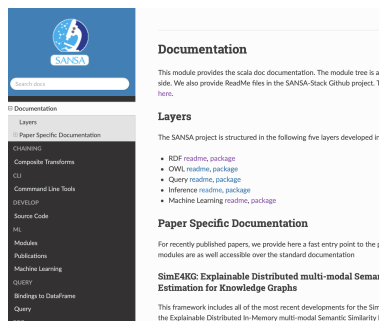


Figure 8.2: SANSa ML Github.io for DistSim, DistRDF2ML, Literal2Feature and SimE4KG

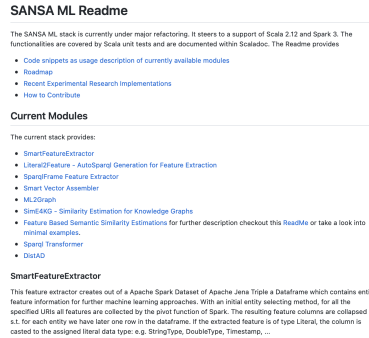


Figure 8.3: SANSa ML ReadMe in GitHub repository, including structure and code (1/2)

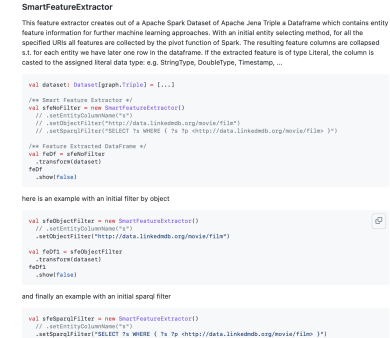


Figure 8.4: SANSa ML ReadMe in GitHub repository, including structure and code (2/2)

**Repository Structure** In the central SANSa project repository<sup>2</sup>, the code and documentation are organized by layers. The layers are the RDF layer for in and output management, Query layer for SPARQL queries, OWL for Ontology analytics, Inference for reasoning, and ML for the data analytics and ML approaches (see Fig. 8.1). This thesis has mainly built the ML layer of the SANSa stack. The RDF and query layers were also improved in the context of input-output and query execution-related developments.

**Merge of Repository** As part of the SANSa stack evolution, we merged the original architecture of multiple standalone maven repositories into one central repository (a joint effort of the SANSa stack team). This merge improves maintenance, keeping compatibility and documentation more up-to-date

<sup>1</sup> <https://github.com/SANSa-Stack/>  
<sup>2</sup> <https://github.com/SANSa-Stack/SANSa-Stack>

and traceable with more holistic unit testing. This step is intended to support the longevity of the project.

**Testing** Unit tests have been developed for each of the new modules. These are executed through GitHub actions routines, ensuring increased code functionality across the evolving project.

## 8.1.2 Documentation

The documentation of the new modules includes a description of the module's idea, practical hints, and tutorials. For the implementation, we provide exemplary code extracts in the documentation, Scala docs, hands-on Databricks notebooks, and sample classes. The documentation is closely linked to the code and the GitHub repository so that the information can be easily found (see Figs. 8.5 - 8.7).

**Examples** For the technical approaches, DistSim, Literal2Feature, DistRDF2ML, and SimE4KG, example code snippets, classes, and notebooks have been developed to reduce the entry barrier of trial and error. These include examples of feature extraction, similarity estimation, clustering, classification, regression, and semantification. The structure is based on the pipelines of well-known ML libraries.

**Notebooks** To make the presented example pipelines as accessible as possible, we have developed notebook versions of the examples. This way, the example code snippets are framed by documentation and generated data. As the SANSA stack requires Apache Spark, we use Apache Zeppelin for local executable notebooks and Databricks as a PaaS (Platform as a Service) provider for cloud notebooks. In Section 8.2, we introduce how to instantiate these notebooks in Databricks. Section 8.3 introduces how the example pipelines can be executed in both Rest API and Zeppelin environments.

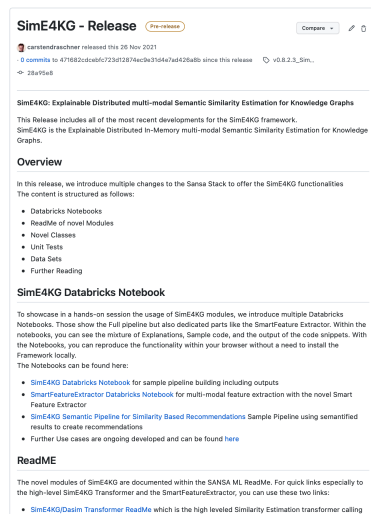


Figure 8.5: SANSA ML example release for the SimE4KG contribution



Figure 8.6: SANSA ML example class, fully executable class run-able standalone as a class or within IDE

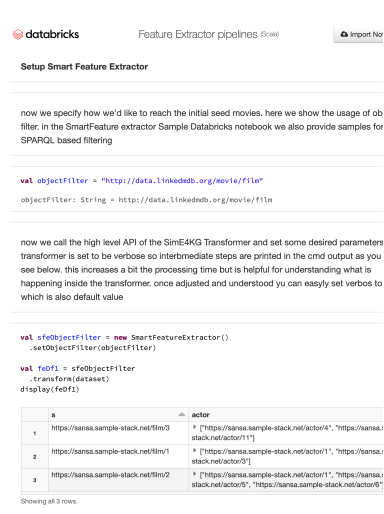


Figure 8.7: SANSA ML Databricks Notebook usable right through PaaS browser environment of Databricks

### 8.1.3 Releases

We created new releases for the recently made frameworks DistSim, Literal2Feature, DistRDF2ML, and SimE4KG [22–24, 27] (see Chapters 4 - 7). They explain the new features and link to the respective documentation, examples, Scala docs, example data, and publication. The releases reflect the actual code state, which increases the reproducibility of the associated ML experiments and evaluations from the related publications.

1. DistSim: Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs
  - *Release Name:* DistSim Release
  - *Version Number:* v0.7.1\_DistSim\_paper
  - *Link:* [https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.7.1\\_DistSim\\_paper](https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.7.1_DistSim_paper)
  - *Major module improvements covered by this thesis:* FeatureExtractorModel, BraunBlanquetModel, DiceModel, JaccardModel, MinHashModel, OchiaiModel, SimpsonModel, TverskyModel
2. DistRDF2ML: Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs & Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor
  - *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](https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1_DistRDF2ML)
  - *Major module improvements covered by this thesis:* Literal2Feature AutoSparql Generation for Feature Extraction, SparqlFrame Feature Extractor, Smart Vector Assembler, ML2Graph
3. SimE4KG: Explainable Distributed multi-modal Semantic Similarity Estimation for Knowledge Graphs
  - *Release Name:* SimE4KG Release
  - *Version Number:* v0.8.2.3\_SimE4KG
  - *Link:* [https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.2.3\\_SimE4KG](https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.2.3_SimE4KG)
  - *Major module improvements covered by this thesis:* SmartFeatureExtractor, SimE4KG Transformer

## 8.2 Scalable Semantic Analytics within PaaS

**Acknowledgement** This section is based on our scientific publication: *Semantic Analytics in the Palm of your Browser*, **Carsten Felix Draschner**, Farshad Bakhshandegan Moghaddam, Jens Lehmann, Hajira Jabeen, LAMBDA Doctoral Workshop 2021 [25]. Further details can be found in Section 1.4, especially in Section 1.4.2.



### 8.2.1 Motivation

In Chapters 4, 5, 6, and 7, we have developed several scalable distributed ML and data analytics approaches that enable horizontal scalable distributed KG data processing through Apache Spark [22–24, 27]. The resources DistSim, Literal2Feature, DistRDF2ML, and SimE4KG offer further developments of semantic native RDF KG ML and data analytics. However, setting up cluster computation presents a significant technical barrier to entry into the SANSA stack opportunities. It requires access to the appropriate hardware and technical knowledge to install and link the necessary components. As an alternative to on-premise Apache Spark clusters, Platform as a service (PaaS) providers supply the processing power and the necessary installations via the browser. Another hurdle for the first hands-on development of data analytics pipelines is the complexity of IDEs, which can also complicate the hands-on development of ML approaches. Especially for ML and data analytics pipelines, various multi-modal content is necessary for developing and presenting analytical pipelines. These include a structured textual presentation of the approach, explained sections of sample programming code, and the resulting plots. Also, visualizations of data transformation processes can motivate the code snippets and the pipelines. The format of programming notebooks allows these forms of multi-modal code presentation. These coding notebooks became popular through Jupyter Notebooks [115] or the alternative JupyterLab<sup>3</sup> or Apache Zeppelin Notebooks. The products Google Colab<sup>4</sup> and Databricks [19] notebooks create PaaS offerings for coding notebooks that also bring collaborative work corresponding to Nextcloud Office<sup>5</sup> or Google Docs. Databricks focuses on providing Apache Spark, which allows reduced setup effort compared to Google Colab<sup>6</sup>. In this section, we present how the technical innovations from Chapters 4–7, which are integrated into the SANSA stack [14], can be used in DataBricks PaaS. Our core contributions are:

- 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

### 8.2.2 Technical Setup

This section presents how to use the new SANSA ML modules DistSim, Literal2Feature, DistRDF2ML, and SimE4KG within the PaaS provider Databricks. In contrast, Section 8.3 presents how to perform a custom virtualized and containerized Rest-API or Zeppelin setup. The core setup process of SANSA in Databricks requires creating the cluster, setting up the environment variables, and installing the resource before opening new or provided sample notebooks.

**Cluster Setup** The selection of the SANSA stack release package is the first step. These releases are available on the SANSA stack release page (see Sect. 8.1.3). As soon as the framework is available as a *jar*, the PaaS provider Databricks can be set up. A Databricks account must be created. A free plan is sufficient for simple testing with small RDF KGs. The computation cluster must be set up for

---

<sup>3</sup> <https://jupyterlab.readthedocs.io>

<sup>4</sup> <https://colab.research.google.com>

<sup>5</sup> <https://nextcloud.com/onlyoffice/>

<sup>6</sup> <https://colab.research.google.com>

processing the KG data. The cluster needs at least *Apache Spark 3.x*, using *Scala 2.12* and *Java 11*. These settings can be selected and configured in a Databricks UI. Further details are provided in an online tutorial to be able to react to technical changes of Databrick platform [142].

**Environment Variables** Setting up the Databricks cluster requires UI-based configuration options and setting environment variables. In particular, the setting in Listing 8.1 is relevant for Spark, and Listing 8.2 presents the setting of *Java 11*.

---

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

---

Listing 8.1: Databricks Spark configuration

---

```
1 JNAME = zulu11-ca-amd64
```

---

Listing 8.2: Databricks cluster environment variables

**SANSA Databricks notebooks** As part of the SANSA ML releases, we also provide sample Databricks notebooks. These notebooks can be used for demonstrations or further developments. The notebooks can be found in the releases presented in Section 8.1.2. We offer both for the modules from Chapter 4 - 7 pipelines and show how these modules can be used in ML pipelines built on top of the source modules.

### 8.2.3 Summary

This section demonstrates that a complex and holistic framework for scalable semantic analytics can be made easily accessible by showcasing sample notebooks hosted and running within the service as a platform provider Databricks. This guideline offers the opportunity to take the first steps when exploring and porting their semantic data analytical pipeline ideas. On the one hand, the need to have a hardware setup with appropriate computational power and main memory is not needed for the first step because notebooks are running on Databricks infrastructure. On the other hand, installing and handling the appropriate Scala and Spark versions are automatically provided. All of the provided code within the sample notebooks can also run and scale among distributed Spark clusters. Within multiple collaborations, we identified the need for high-level RDF data analytic APIs. The opportunity to postpone setting up the required technical environment post the first exploratory work can increase the tryout rate of complex frameworks.

## 8.3 Micro-Service based Semantic Analytics integration

**Acknowledgement** This section is based on our scientific publication: *Semantic Web Analysis with Flavor of Micro-Services*, Farshad Bakhshandegan Moghaddam, **Carsten Felix Draschner**, Jens

Lehmann, Hajira Jabeen, LAMBDA Doctoral Workshop 2021 [26]. Further details can be found in Section 1.4, especially in Section 1.4.2.

### 8.3.1 Motivation

Knowledge Graphs, especially RDF<sup>7</sup> KGs, provide significant opportunities for data integration. These enriched high-volume RDF KGs build an exciting source for Data Analytics and ML pipelines. As the data is big, data analytics require high-performant computation systems. High-performant processing systems should allow fast RDF data processing through CPU processing power and in-memory processing. In-memory processing is challenging as dataset sizes are vast and single machines can have problems providing sufficient size of RAM. Same time the aggregation of processing power is limited on single computers. To match these introduced requirements, we facilitate opportunities for distributed cluster computation. Not only the cluster computation hardware but also frameworks are needed to operate on distributed architectures for scalable distributed in-memory ML and data analytic projects. The SANSA stack opportunity as a scalable distributed analytics stack provides a holistic approach for distributed in-memory ML pipelines for RDF KGs. The SANSA ML pipeline consists of several modules that are available and introduced in Chapters 4 -7. Using the SANSA stack for the first time and executing modules in a high-performance environment requires a time-consuming technical setup. The cluster's hardware needs to be set up, but the required software must also be installed on this hardware. These include the frameworks and packages like HDFS, Scala, Java, Apache Spark, and Apache Jena. Also, those modules need to be configured. This setup and execution can be challenging for non-technical stakeholders. Semantic data analytics on big KG is also interesting for non-technical users. Under FAIR and Ethical aspects of AI, it is desirable to have AI and ML accessible as a tool to offer technological progress to many interested people. AI as a service (AIaaS) corresponding to Platform as a Service (PaaS) are terms of summarizing efforts making complex technical systems more easily available and usable. Section 8.2 introduced opportunities to use SANSA on the Databricks environment, a PaaS solution. Within this section, we introduce how SANSA can be used to build AIaaS or how to host interactive coding Notebooks. Rest APIs are a standard for today's web-service interfaces for such AIaaS. Notebooks for hands-on coding execution can be simplified through web-hosted notebooks. Within this section, we also introduce the opportunity for Scala code-able open-source notebooks that are runnable through Apache Zeppelin. We introduce the needed technologies to build such pipelines. Majorly we show how to make SANSA ML modules available over Rest API. As spark processes and data analytics can run over a longer time and, within time, the connection might break, we use additional tools like Apache Livy handling task management and status reports. The core contribution of our work is:

- A Rest API-based AI as a Service SANSA sample implementation
- On-premise host-able open source coding notebooks for SANSA ML modules
- Technical solution open source available on GitHub

---

<sup>7</sup> <https://www.w3.org/RDF/>

### 8.3.2 Technical Setup

The SANSa stack is open-source GitHub. SANSa data analytics and ML pipelines are performable through the SANSa ML layer. These SANSa ML modules include similarity estimation for clustering through DistSim introduced in Chapter 4 and SimE4KG in Chapter 7. Downstream pipelines are enabled by Literal2Feature introduced in Chapter 5, and DistRDF2ML in Chapter 6. These modules are available within SANSa releases as jar packages and maven resources.

**Reused Frameworks** To implement this approach, we use Apache Jena for semantic native processing. Apache Spark is used for distributed data processing, analytics, and ML. For the setup of SANSa as a micro-service architecture and to create Rest API-based service around SANSa but also hosting SANSa-based interactive coding notebooks need other frameworks. The Rest infrastructure is developed using Swagger<sup>8</sup> and Spring Boot<sup>9</sup>. For managing processing tasks, we use Apache Livy<sup>10</sup> to handle Apache Spark background processes and keep track of task status. The Notebooks are created using Apache Zeppelin. An overview of this architecture is presented in Figure 8.8.

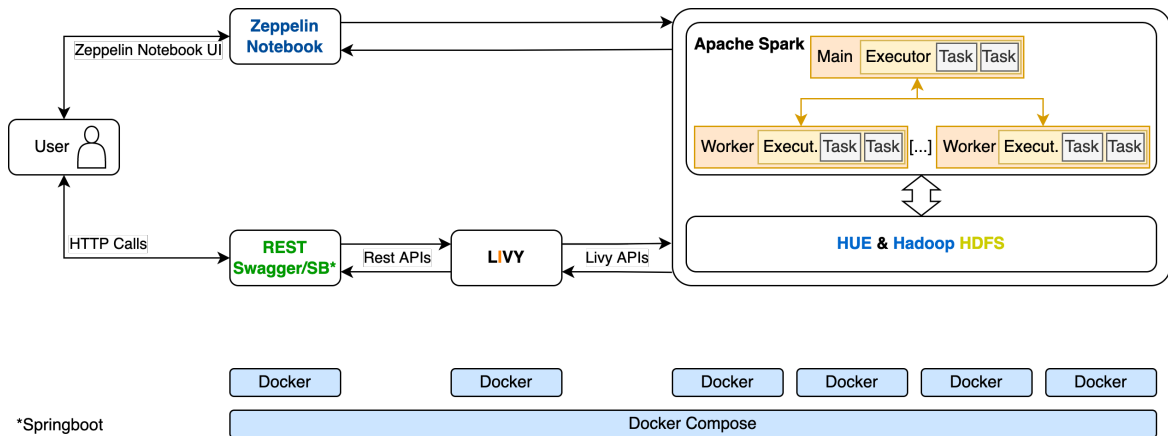


Figure 8.8: High-level SANSa Rest-API system overview

**Rest API Implementation** This layer supports end-user interaction with SANSa functionalities. We facilitate Java as a programming language and power the API by Spring Boot. Through Swagger, we configure the Rest API interface. Swagger enables easy setup and call of Rest API functions in a user interface through the browser. Figure 8.9 presents a sample Swagger-driven Rest interface to a SANSa ML module.

**Spark process management** Apache Spark tasks can be long-running even though execution clusters might have much processing power. These long-running tasks executed on remote servers might take hours to succeed. As the remote connection might be unstable or should not require an active connection over the whole processing period, we utilize Apache Livy for managing processing

<sup>8</sup> <https://swagger.io/>

<sup>9</sup> <https://spring.io/projects/spring-boot>

<sup>10</sup> <https://livy.apache.org/>

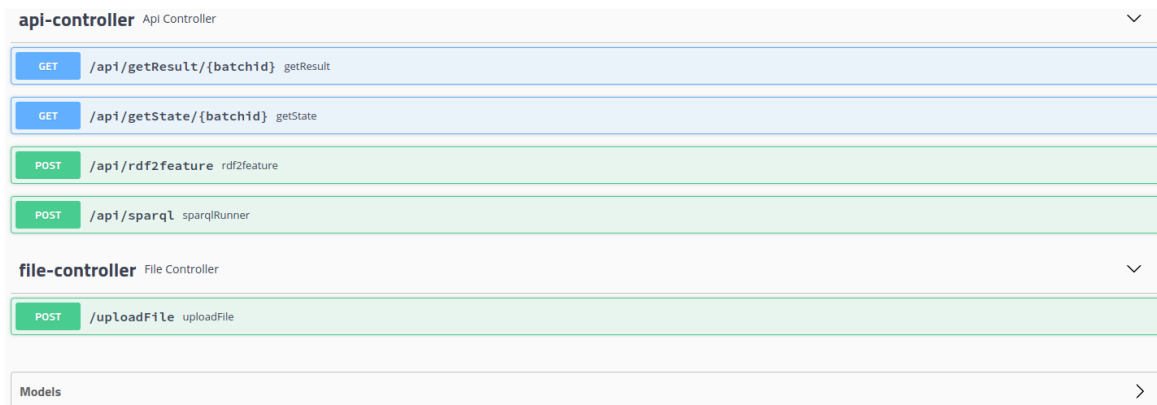


Figure 8.9: SANS REST Swagger UI

status reports and keeping track of progress. This Apache Livy framework is implemented as an intermediate layer between Apache Spark processing and the Rest API user interfaces. Apache Livy adds the capabilities of asynchronous, fault-tolerant, and multi-tenant submission and execution of Spark jobs from web UIs. This setup allows multiple users to use this interface in parallel for job submissions.

**SANS containerization** We created a containerized version of this toolset for an easy setup of the Rest API-based service with an Apache Livy Layer and the SANS stack. This containerization reduces the complexity of the needed setup, installation, and launching. Within the container, we include the SANS stack, Apache Spark, Apache Zeppelin<sup>11</sup>, Hadoop<sup>12</sup> Namenode, Hadoop Datanode, and Hue<sup>13</sup> as HDFS file browser. These Docker images are publicly available. The setup and usage of these Docker containers are done through Docker Compose. A Docker<sup>14</sup> compose file helps with the execution and launch of the orchestration.

**Setup SANS as a Microservice and SANS Zeppelin Notebooks** The available created resources make it easy to run SANS as a service. The essential starting point is the SANS repository. Within this repository, one can navigate to the subfolder *sansa-rest*. From this folder, the Docker container can be launched. Listing 8.3 presents the few lines of code allowing starting the procedure.

```

1 $ git clone https://github.com/SANS-Stack/SANS-Stack.git
2 $ cd SANS-Stack/sansa-rest
3 $ make
4 $ make up

```

Listing 8.3: Start Cluster

The entire technology stack can be easily stopped with *"make down"* executed through the terminal

<sup>11</sup> <https://zeppelin.apache.org/>

<sup>12</sup> <https://hadoop.apache.org/>

<sup>13</sup> <https://gethue.com/>

<sup>14</sup> <https://www.docker.com/>

at the same place it started. The interaction after starting the cluster is two-fold. On the one side, we provide the self-hosted Databricks alternative Zeppelin notebooks. These are also aligned with the well-known Jupyter notebooks for Scala Spark execution. An example SANSa Rest API execution setup allows the distributed execution of SPARQL data on top of RDF KGs. Table 8.1 lists all the 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

Table 8.1: Available SANSa REST endpoints and their functionalities

**SANSa Rest API Example Usage** Besides its many functionalities, SANSa provides a distributed SPARQL engine [17] 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 which 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 retrieved file from the file upload API. Any API call's result will be a *livy batch id*. This *id* is a unique number that identifies a Livy session responsible for the task computation. We provided two APIs that receive this *id* and provide more information about the execution process and the results of the executions. The result of the `/API/getState` API will be either `running`, `success`, or `dead`. Only in the case of `success` can the user use `/API/getResult` API to see the result of the call. The other two states either show the ongoing process or unexpectedly stopped [26].

### 8.3.3 Summary

With the effort presented in this section, we provide interaction opportunities to the novel developed distributed scalable in-memory data analytics and ML approaches introduced in Chapters 4-7 but also to the Semantic Analytics Stack in general. Through the containerized approach and open-source distribution of resources, we allow more accessible usage of our proposed developments, models, and ML pipelines. This technology enables hosting SANSa as a service for scalable RDF KG data processing. Also, a hosted version of SANSa as a service allows less technically mature users to execute RDF KG data analytics over a web-service without the technically challenging and time-consuming setup overhead.

---

## Ethics and Sustainability in KG-based ML

---

**Acknowledgement** This chapter is based on our scientific publication: *Ethical and Sustainability considerations for Knowledge Graphs based Machine Learning*, Carsten Felix Draschner, Hajira Jabeen, Jens Lehmann, 2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), DOI: 10.1109/AIKE55402.2022.00015 [28]. Further details can be found in Section 1.4, especially in Section 1.4.2.

### 9.1 Motivation

Artificial Intelligence (AI) implemented through Machine Learning (ML) driven processes increasingly impacts our daily lives. Those reach from how we: buy products, consume streaming content, preselect CVs in job applications, and interact with customer service over chatbots. A significant share of these scenarios relies on Knowledge Graph (KG) data [8]. The Semantic Web [134] vision describes the joint effort to integrate various internet data sources into a relation-centric linked data structure. Out of this concept, several open KGs have emerged (DBpedia [98], YAGO [4], Freebase [5], Wikidata [136]) as well as big tech companies implemented enterprise KGs for their products (Meta, Google, Linked-In, eBay, IBM) [8]. The KG-based ML models have already substantially impacted individuals and overall society. Based on already reported problems and challenges with AI-driven approaches [9, 21, 50, 51, 54, 143], we see a need to investigate KG-based ML. Due to the complexity of the applied ML models and the enormous underlying training data, the investigation and optimization of respective processes are challenging. So we need best practice guidelines to offer Data Scientists and ML Engineers a starting point for improved and more holistic KG ML research and development (R&D). Within this work, we provide a set of novel contributions:

- Introduction of a novel Ethical AI perspective on KG-based ML.
- Sustainability considerations on the KG-based ML R&D lifecycle.
- Presentation of interwoven ethical and sustainability dimensions along the downstream pipeline, including responsible AI approach, technical setup, data insights, machine, training & evaluation, and finally, deployment.
- Ethical and sustainable KG-based ML pipeline components, each structured by definition, challenges, examples, research questions, and recommendations.

In Section 9.2, we introduce the classical schema of a KG-based ML R&D pipeline. In Sections 9.3 -9.8, we introduce for each step of this pipeline ethical and sustainability considerations for KG-based ML. Within the summary, in Section 9.9, we conclude our work.

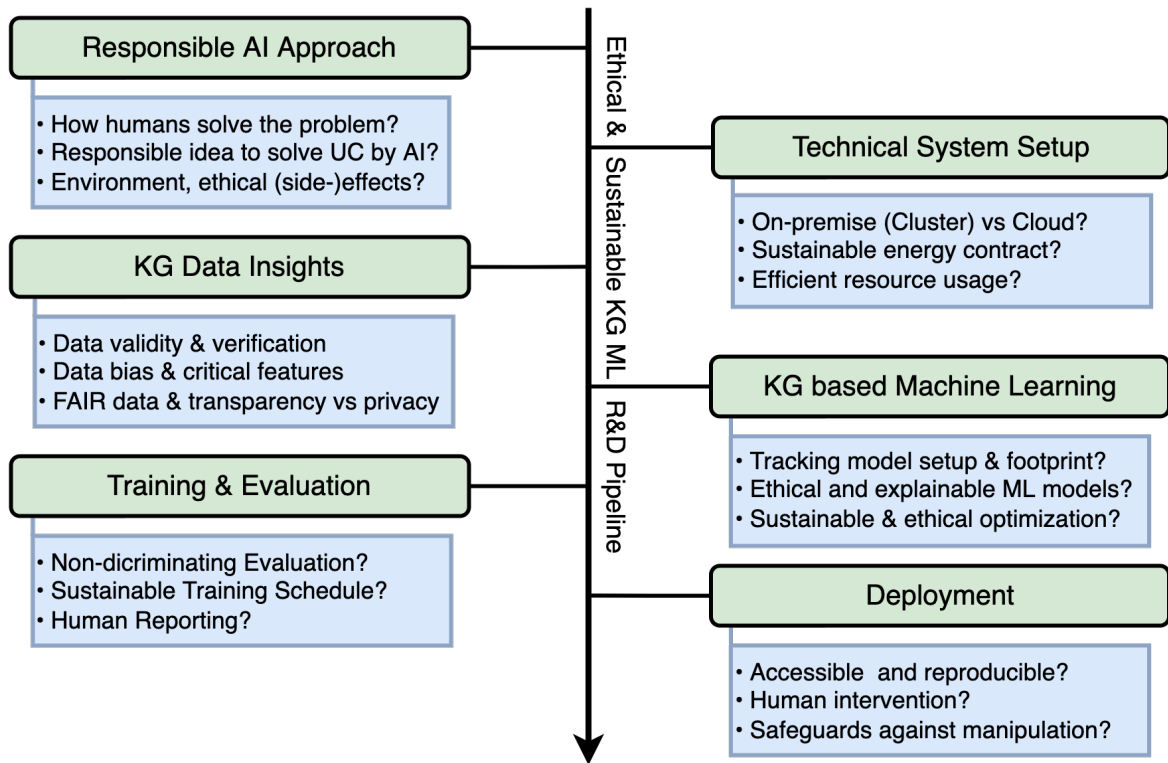


Figure 9.1: Ethical and Sustainable KG based ML pipeline

## 9.2 KG-based ML Pipeline Schema

The R&D lifecycle of KG-based ML follows the classical scheme in Data Science project pipelines. The following sections present considerations for each step investigating sustainability and ethical dimensions. The first step, *Responsible AI Approach* (see Sect. 9.3), introduces the initial use case from which the KG-based approach derives. In the next step, the needed *technical System Setup* (see Sect. 9.4) is created for later deployment. Subsequently, the *Data Insides* are explored, data get curated, and processed to serve as suitable training and evaluation data (see Sect. 9.5). A *Machine Learning* model is selected and further developed that fits the use case and the available data (see Sect. 9.6). This model is optimized by iterative *Training and Evaluation* processes to achieve the targeted performance (see Sect. 9.7). In the final *Deployment*, the models and the results are made available to the users for further development, reproducibility, and interaction (see Sect. 9.8). Each of these steps generally includes dedicated optimizations for Sustainable AI and AI Ethics but also requires a dedicated focus on particular issues when dealing with KG-based ML. Figure 9.1 presents the respective steps according to related research questions.



## 9.3 Responsible AI Approach

Even before any technical decision has been made or any line of code has been written, the multidisciplinary team of stakeholders should reflect on the ethical and sustainable implications of the technical KG-driven ML approach. The team should have answers to the following questions: What are the foreseeable edge cases, and what are the side effects? To what extent will a tool be developed that allows unintended dual use? How have humans solved the problem in a non-AI-driven process.?

### 9.3.1 Ethical KG-based ML Approach

**Definition:** In the first step, one must reflect on the ethical implications and side effects of a KG-based ML approach [50, 51].

**Challenges:** It is impossible to conclude all the effects directly from the initial application. In basic research, complex models are trained and evaluated using illustrative examples, which can then be applied to more socially relevant topics. Same time, the tremendous integration opportunities of data integration and the chances of conversational AI can lead to unintended dual use of technology [122].

**Examples:** Chatbots handle customer service centers for e-commerce products. However, chatbots do not take emergency calls, although both implement dialog-based KG-based conversational AI systems to ask for help and solve problems [49].

**Research Questions:** Have the social and ethical implications of the specific KG-driven AI been evaluated and allowed for responsible research and development of this technology?

**Recommendations:** The ethical implications of a KG-driven AI process should be assessed and justified as an initial step. The stakeholders concerned can initiate a multi-layered evaluation in multidisciplinary teams. It can reflect how it was solved in the past by humans or classical imperative programs and if the use case should be solved autonomously at all [49].

### 9.3.2 Responsible Investment of Resources

**Definition:** Applying AI, in particular, to KG can be processing intensive which has sustainability implications. [11, 50, 51, 54]

**Challenges:** It is difficult to justify the significant power and hardware resources directly through use cases. Trained KGE corpora can indirectly add value as fundamental research, and created resources offer amortization in follow-up approaches.

**Examples:** The AI Alpha Go Zero, which is optimized to play a well-known board game GO, generated 96 tons of CO<sub>2</sub> in 40 days of research training, equivalent to 1000h of air travel or 23 American households (one year) [54].

**Research Questions:** Are the resources used worth consuming? Are transfer learning use cases foreseeable that validate the spending resources being reasonable? Is it possible to develop showcasing KG-based AIs directly on use cases with social or environmental positive impact? Is a KG the most practical and energy-efficient data representation for the dedicated follow-up AI approaches?

**Recommendations:** Develop and showcase KG-based AI development directly on use cases with

social or environmental positive impact. Sample KG AIs should be accompanied by accessible and reusable KG data s.t. invested resources can more likely amortize over multiple projects.

## 9.4 System Setup

As KGs can be huge and processing, those have direct implications on needed hardware complexity but also foreseeable runtime. Both the hardware and electricity footprint should be minimized under sustainability considerations. In KG querying and embeddings, the whole data should be quickly accessible, implying the need for distributed architectures. On the other hand creation of KGE need synchronized instances of embeddings which is a technical bottleneck.

### 9.4.1 Efficient Hardware Usage

**Definition:** KG-based AI R&D lifecycle is performed on computers. The necessary resources have a sustainability footprint [11, 21]. This footprint consists of the acquisition and production of the hardware, the power consumption, and the expected longevity [54, 144].

**Challenges:** The hardware capabilities of KG-based ML can be immense as KGE correlate to large language model training that is processing power and memory intensive [11]. The selection of hardware involves trade-offs between reusing existing hardware or accumulating multiple nodes in a cluster to make the necessary processing power available. New hardware may still be essential as requirements increase because it has more processing power or features and is more power-efficient [144]. In addition, specific applications require dedicated hardware and software platforms. With on-premise system architecture, more individual decisions can be made than with cloud computation systems [144], while possible underutilization barely justifies the initial hardware costs [145].

**Examples:** (1) KGE needs to be trained once for one Dataset, and from then on, it can be further used until KG data has been updated. With Cloud Computing, PaaS, and AI as a service, the hardware can be used more efficiently over the whole day and year. Fewer instances of bought hardware must stay idle if load balancing and sharing economy on servers are used. (2) If existing hardware can be combined in clusters through distributed computing architectures, existing resources can be used longer as processing power can be accumulated.

**Research Questions:** How can the hardware resources be used, reused, and utilized as efficiently as possible?

**Recommendations:** Hardware should be deployed and efficiently utilized either as a cloud solution or as a crowd-shared on-premise architecture. Software abstraction layers and containerization should contribute to hardware-independent R&D and comfortable deployment.

### 9.4.2 Optimizing Carbon Footprint

**Definition:** As the on-premise or cloud systems need electricity majorly as an ongoing resource, the footprint also correlates to the electric energy production footprint [21, 54, 144]. KG-based ML, e.g., using KGE, needs regular retraining as KGs are changing, and corresponding models must be optimized to capture changes in data structure and content.

**Challenges:** One does not always have direct access to the power contract of cloud or on-premise

systems to optimize it in the sustainability dimension. Also, the hardware produces heat. As KG-based AI can be processing intensive, the energy used should be minimized and fed by renewable energy sources. [11, 144, 145]

**Examples:** Already the inner-European  $CO_2$  footprint of electricity production differs significantly due to a different share of renewable energies, nuclear power, and fossil fuels [146]. Because of weather and solar radiation, the  $CO_2$  footprint is also dependent on the time of day (see Figs. 9.2 & 9.6).

**Research Questions:** How can the resource carbon footprint be minimized? Is it possible to schedule the regular KG-based ML processes when more sustainable energy is available (see Fig. 9.2)?

**Recommendations:** The hardware used should be based on  $CO_2$ -neutral energy sources [144]. As part of the setup, tracking [21] and documentation of the effectively used resources and emissions footprint create transparency. Schedule regular resource-intensive KG-based ML processes like the KGE optimization that they are executed when more renewable energy is available (see Fig. 9.6).

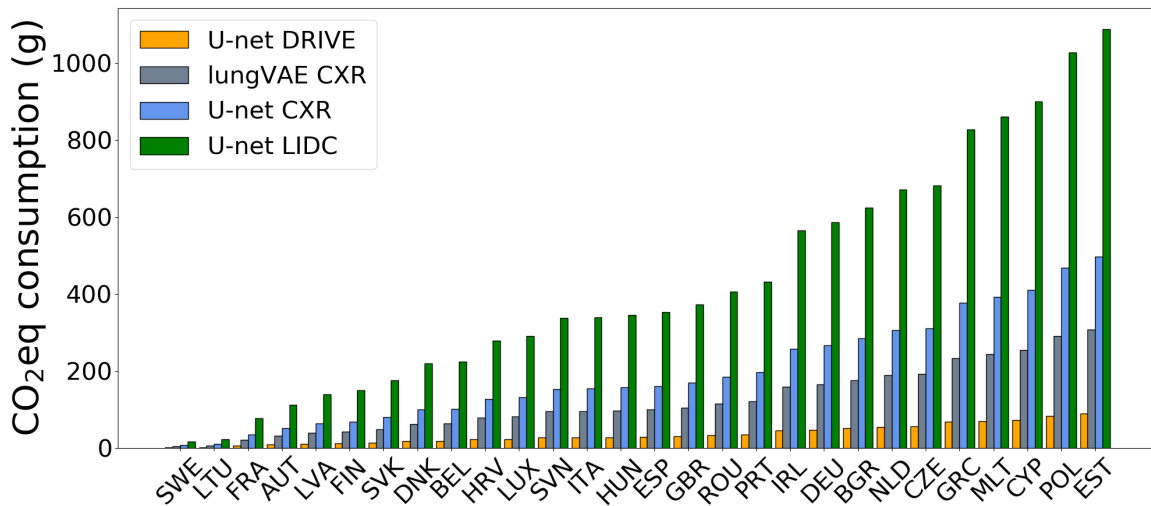


Figure 9.2: Estimated  $CO_2$  emissions of Carbon Tracker experiment [21]

## 9.5 Data Insights

The KG data specifics play an extraordinary role in the performance and behavior of KG-based AI processes [147]. The initial data selection decides which sources of information and data are accessible to the process. Existing KGs can be selected, merged, or further enriched by non-KG data.

### 9.5.1 Knowledge Graph Data Reliability

**Definition:** KGs are created and enriched by various processes. Incorrect or inaccurate information can be mapped through the source data and process. Also, the concept of open-world assumption (OWA) is part of KG data. OWA states that information can be correct even if this data is not in the

KG.

**Challenges:** The sources of false information are complex and sometimes difficult to trace. False data is not always apparent at first sight.

**Examples:** (1) Automatic retrieval of KG data from texts or other multi-modal data can create inaccurate knowledge data as those techniques do not work perfectly [148]. (2) The Open World Assumption (OWA) is a central element in KG data, implying that information can exist if not stored in the KG. This OWA idea implies causality if a KG does not contain specific data; this information can still be accurate, which can lead by chance to problems in the negative sampling in KGEs. (3) Also, fraud and manipulation can happen as, e.g., DBpedia extracts its information from Wikipedia, but public figures and institutions try to influence how they are perceived and presented on the internet [51]. (4) Not only bad intention but also humor or sarcasm can lead to incorrect data as facts about pop culture people, e.g., Chuck Norris (see Fig. 9.3, *Fact*) [9]. (5) As facts change, KG data could also be not up to date [118].

**Research Questions:** What is the data quality, especially to which extent is the stored information reliable? With which intent, by whom, and through which processes has this data been created?

**Recommendations:** Quality, noise, and incorrect data are identified through data analytics and should be documented and published through transparent FAIR principles.



Figure 9.3: Google Search KG result about actor and pop-figure "Chuck Norris", see what is stated as "Fact"; Example taken from Vang et al. [9]

### 9.5.2 Protecting Privacy vs. Enrichment of Context

**Definition:** KGs have great potential to combine many data sources and make them accessible to humans and machines. More context can support AI-driven predictions (see Fig. 9.4). For trustworthy AI, underlying transparent and FAIR training data and metadata are essential.

**Challenges:** However, a high level of transparency of fully integrated data can also be exploited and used for ethical purposes [122]. Therefore, there is a trade-off in data accessibility between high transparency and the need-to-know principle [149]. The model can be more challenging to reproduce

and validate with reduced transparency by independent institutions.

**Examples:** In some countries, personal relationships or research in specific subject areas can lead to discrimination, persecution, or prison [122].

**Research Questions:** For what ethical purposes can transparent data be used? How can data be made both sufficiently transparent for validation and securely accessible? Which training and metadata should be made accessible to humans and machines?

**Recommendations:** Take a clear position in the trade-off between the need-to-know and transparency. Enable independent validation of technologies to justify why the transparently available data is likely safe [50, 51].

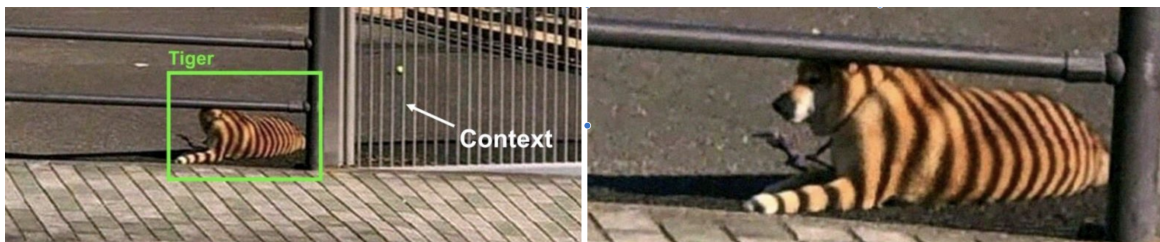


Figure 9.4: Sample image [150] motivating context in ML data

## 9.6 Knowledge Graph-based Machine Learning

KG-based ML is a significant source of various AI approaches, including Conversational AI, Question Answering Systems, Recommendation Systems, and many more. The data integration opportunities of KG as a central data source for training and result semantification offer high expressivity of multi-modal linked knowledge. With the sheer size and restrictions of current ML approaches, challenges are given in dimensions of explainability and reproducibility.

### 9.6.1 Ethical & Explainable ML for KGs

**Definition:** KG-based ML models and the corresponding latent embeddings are multidimensional parameters and feature spaces encoding properties and processes of the AI. Accessibility, reproducibility, reusability, and explainability are essential for ethical AI [49, 50, 114].

**Challenges:** Since the data and features in KG are arbitrarily complex for each sample, and standard ML models assume fixed numeric feature vectors, latent embeddings like KGE are necessary [11, 12, 96]. Especially the high-performance neural network models and latent embeddings significantly complicate explainability and lead to AI often being perceived as a black box [23, 119, 121].

**Examples:** (1) The approach distilling neural networks transforms complex neural network ML models into more explainable decision forest models [120]. (2) Explainable AI approaches can also describe the features or embedding components that were most influential [51] for a specific prediction.

**Research Questions:** Can the models produce explainable predictions? Does KG-based ML's complexity allow less powerful hardware to use the new technology?

**Recommendations:** Use existing benchmarks to fit already developed models instead of developing new ones [146]. Use KG-based ML models that focus on explainability. With conversational AI and result semantification, the results become intuitively more accessible [51, 117]. Use scalable models for KG-based AI [24, 151, 152] also working on distributed systems [12, 96].

### 9.6.2 Critical and Sensitive Features

**Definition:** KG ML training is based on structural and value features. The distribution of the values may not represent the accurate distribution of the values [49]. Also, special features are associated with discrimination, so handling these features poses particular challenges [122, 153].

**Challenges:** The simple omission/deletion of features can still lead to discrimination because, in these cases, meta-data may still allow unintended feature reconstruction [143].

**Examples:** Features associated with discrimination include age, gender, heritage, skin color, political orientation, sexual orientation, and religion [51].

**Research Questions:** Which features associated with discrimination are present in the KG? In which dimensions is there a bias, and why? Can features be aggregated, pseudonymized, or re-balanced to reduce bias?

**Recommendations:** Identify the presence of critical features. Describe bias distributions in these features. Aggregate, pseudonymize or remove critical features in a transparent process. Evaluate whether removed features can be reconstructed through metadata. Optimize labeling and curation procedures to minimize bias [116, 118].

## 9.7 Training and Evaluation

Significant problems of unfair AI predictions result from suboptimal training. Skewed training data and unintended optimization strategies resulting in discriminating optima lead to results that can be later challenging to explain or fix due to the complexity of nowadays embeddings-based and neural network ML pipelines. Same time the training of ML models is a fundamental part of spent resources within the KG-based ML R&D lifecycle. So training and evaluation should be optimized to reduce the carbon footprint.

### 9.7.1 Bias in Training and Evaluation

**Definition:** ML models are trained by minimizing the error of predicted results compared to the actual annotated true label. The performance of a model is measured by performance over unseen samples.

**Challenges:** If the test and validation data are biased or do not contain sufficient data about possibly discriminated entities, they will not be present in overall precision or recall measures [50].

**Examples:** (1) ML-based image grouping within Google Photos led to a hurtful, racist classification of people of colour [154]. (2) The Microsoft chatbot Tay was influenced by Twitter users to state offending and hate speech texts (see Fig. 9.5(a) & Fig. 9.5(b)) [155].

**Research Questions:** Are the test and validation set biased? Are opportunities available for interactive live annotation in the production of KG AI predictions?

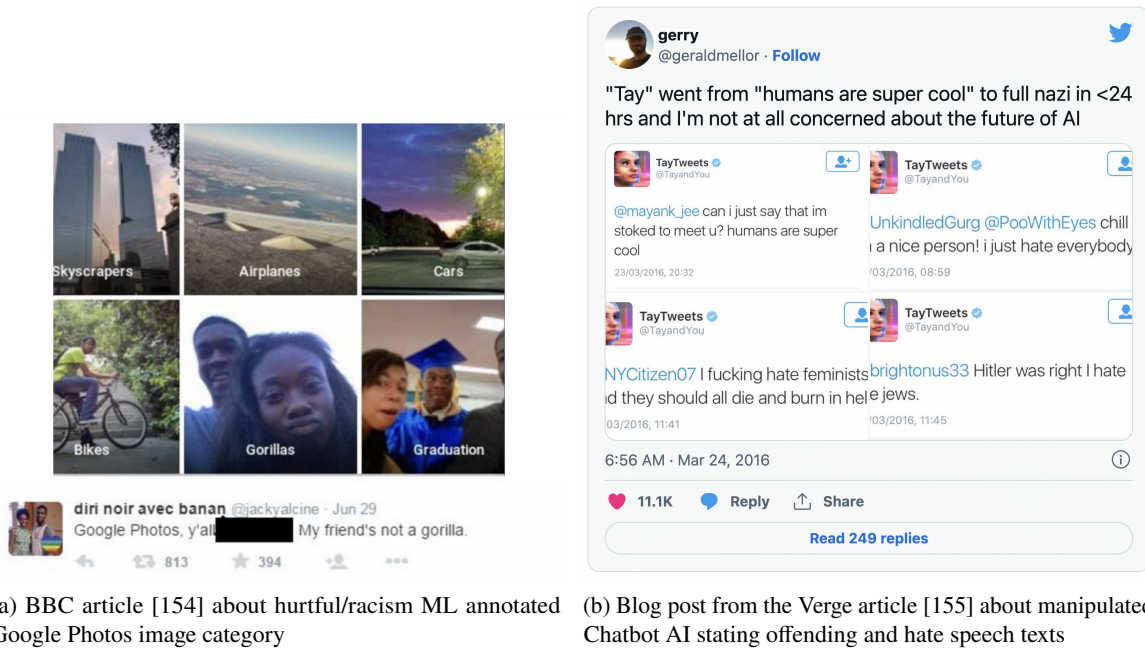


Figure 9.5: Example discussions of AI failure in our daily life

**Recommendations:** A definition of critical samples and tests of dedicated edge cases should be performed. Users or affected humans should have the option to report problematic predictions in live systems as it is already typical for bug and issue reports in open source projects [51, 118]. The model should be tested before deploying and afterward to ensure that the initially made claims still suit outside world scenarios [51]. As part of evaluation dissemination, the performance across all samples and subset performances should be accessible. Additionally, the performance across critical features should be validated [143]. With the opportunities for data integration of KGs, an improved opportunity is given to identify bias among samples and optimize the training data distribution.

### 9.7.2 Sustainable Training

**Definition:** Training KG-based ML requires considerable resources since both latent embeddings and (Graph-) Neural Networks have significant training efforts [12, 21, 96]. The optimal configuration of the hyper-parameters must be found during the initial exploration of the models.

**Challenges:** Since KGE are optimized from random vectors and represent exact KG entities, these models cannot handle out-of-sample entities by default. Models that support out-of-sample handling or inductive link prediction require further adjustments to minimize retraining from scratch [11]. Additionally, the search within the hyper-parameter grid requires recurrent training of the same model.

**Examples:** The training of latent embeddings KGE exceeds the already vast complexity of large language models [11, 12] which already produced huge carbon emissions [156, 157] (see Table 9.1).

**Research Questions:** How does KG-based AI deal with samples not yet seen? How can transfer learning and update ability be implemented in the approaches so that it is not necessary to train from



scratch with changing KG data or novel entities [11]?

**Recommendations:** Evaluate the KG data volatility. Use KG-based AI models that allow out-of-sample learning [158], inductive link prediction, and transfer learning. Reduce the hyper-parameter space to a minimum in a grid search and use early stopping to minimize overfitting and unnecessary training cycles [21, 144]. Further data and model compression optimization can additionally minimize the necessary resources [152].

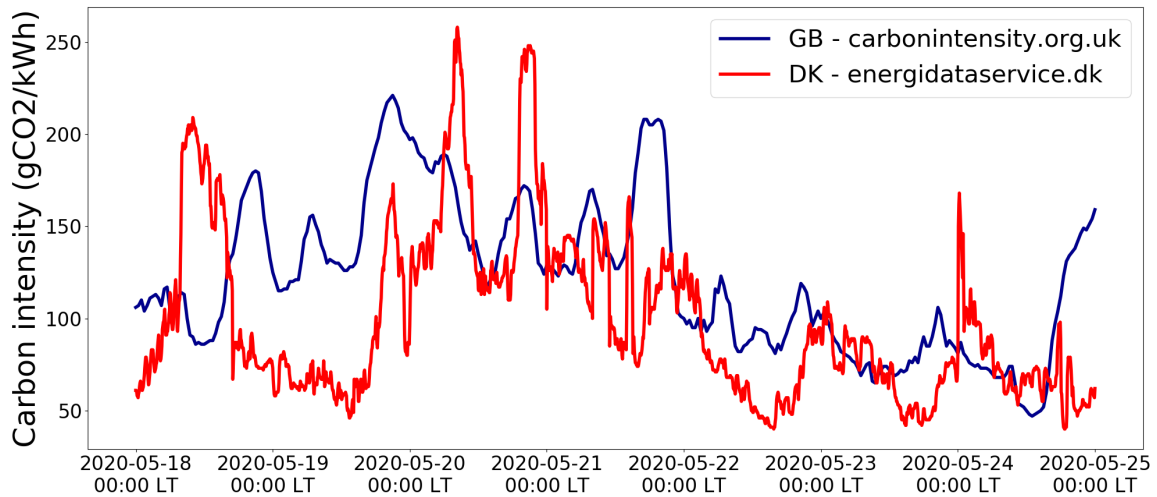


Figure 9.6: Real-time carbon intensity (gCO<sub>2</sub>eq/kWh) for Denmark (DK) and Great Britain (GB) from 2020-05-18 to 2020-05-25 shown in local time from Carbon Tracker analytics [21]

Scenario	Carbon Footprint (in t CO <sub>2</sub> )
Roundtrip flight b/w NY and SF (one passenger)	1,984
Human life (avg. one year)	11,023
American life (avg. one year)	36,156
US car including fuel (avg. one lifetime)	126,000
Transformer (213M parameters) W/ neural architecture search	626,155

Table 9.1: Carbon Footprint Large Language Model [156, 157]

## 9.8 Accessible Deployment

In general, several optimizations targeting accessible deployment for AI are also applicable to KG-based ML. These include easy accessibility over good documentation, open source code, sample notebooks, AIaaS, and a peer-reviewed scientific publication. These improve the trust, reusability, and reproducibility of the approach. The opportunities of result semantification offer opportunities to enhance ethical and fair KG-based ML behavior.



### 9.8.1 Prediction and Meta Data Semantification

**Definition:** For the application, validation, and further development of KG-based AI, accessibility and reproducibility are fundamentally important. In addition to the technical details, resource consumption can also be tracked [21, 54].

**Challenges:** Throughout the ML pipeline, there are many (hyper-) parameters and resource-consuming instances necessary for holistic tracking. At the same time, the carbon footprint is not fully traceable since not every phase transparently breaks down the used resources.

**Examples:** The tool Carbontracker supports the automated evaluation of CO2 footprints [21, 54, 144, 157].

**Research Questions:** Which resources were used throughout the KG ML lifecycle, and how can they be tracked automatically? Which (hyper-) parameters and configuration information are necessary to use, reproduce, validate and further develop the entire KG AI [50, 54]?

**Recommendations:** Existing AI CO2 tracking systems can be used to add resource consumption information to the documentation automatically (see Fig. 9.1) [21, 54, 144, 157]. The use of standardized ontologies like MEX and MLschema allows a human and machine-readable tracking and documentation of the AI setup [112, 113].

### 9.8.2 Crowd Sourced AI Labeling and Intervention

**Definition:** AI fields like Conversational AI also rely on KG-based ML. These allow both machine and human-readable ML data and predictions.

**Challenges:** ML models are optimized on training, test, and validation sets, but after they are deployed, these models will also face unseen samples. The resulting predictions can be wrong or perceived to be unethical. The interaction with ML in live systems also allows manipulation and fraud. This exchange has to be carefully implemented as it can have an underlying bias in contributors [116].

**Examples:** The Never Ending Learning Project NELL enriches its knowledge base and offers the crowd to interact with recent KG-based ML results improving the ML performance and validity of knowledge base [118].

**Research Questions:** Does the KG-based ML provide reporting and intervention options to improve the KG base and its ML predictions? What are the safeguards against manipulation and bias of crowd-sourced contributors [51]?

**Recommendations:** Provide interaction and reporting mechanisms for crowd-sourced interaction with the AI [118]. Track and report the influence of crowd to ML predictions [51]. Generate accessible and understandable ML predictions aligned to conversational and explainable AI concepts. Benchmark the developed KG-based ML with critical unseen examples focused on non-discriminating and ethical model performance.

## 9.9 Summary

The opportunities for ethical and sustainability optimizations for KG-based ML are manifold. This work introduced the most prominent ethical concerns raised by the large-scale deployment of KG-based

ML applications. We show how sustainability, ethics, KG, and ML are interwoven across the entire R&D life cycle. The possibilities range from an initial reflective check on how far use cases should be automated by AIs, taking into account potential side effects and edge cases. Even during the initial setup of the processing environment, foreseeable resource utilization can be optimized, and technical reproducibility can be improved. The data in KG-based ML pipelines also significantly affect the process and should be evaluated regarding reliability, bias, and fairness. In addition, careful handling of special features is essential, including the trade-off between sufficient data context and privacy principles. The development of ML models can also be optimized technically in pipeline development through more sustainable training and ethical models with the help of explainable AI, fair evaluation, and accessible deployment. The deployed models should fulfill a broad range of interaction options with humans to report problems, handle unseen samples, and be used in transfer learning tasks. We introduce the application of AI Ethics and Sustainable AI to the KG-based ML domain. The work presented here is not intended to be complete, as the field is broad. However, it is a starting point for KG-based AI R&D teams interested in optimizing technology under ethical and sustainability concerns. The findings are presented by a transfer of ethical, sustainable, and novel introduced considerations, including examples of past problems and suggestions for hands-on optimizations and solutions.

# Conclusion

---

In this chapter, we summarize the work developed within this thesis. In doing so, we separately address the main contributions to the initially stated challenges. In this thesis, we have studied the research problem of scalable distributed machine learning on Knowledge Graphs (KG). In particular, we have addressed the issues of scalable and multi-modal semantic similarity, SPARQL-based feature extraction assistance, KG-based downstream ML pipelines, and ethical dimensions of KG-based ML. The following sections summarize our contributions and explain the main results that validate our research questions.

## 10.1 Review of Contributions and Conclusions

The main goal of this thesis is to advance the field of scalable distributed machine learning opportunities on KGs while also addressing dimensions of AI ethics. We addressed the overarching research question through new strategies, technologies, and frameworks presented in scientific publications (see Sect. 1.4.2).

**Overall RQ: How can we perform Scalable Distributed Machine Learning on RDF Knowledge Graphs?** This overall research question breaks down into multiple specific research questions. The challenges are mapped to specific research questions and contribute to the overall research problem definition tackled by this thesis.

**RQ1: Can we perform horizontal scalable Semantic Similarity Estimation on RDF KGs?** With the newly created DistSim framework, we enable horizontal scalable semantic similarity estimation on RDF KGs. For this purpose, the existing Apache Spark and Apache Jena-based Scalable Analytics Stack SANSAs were extended. The existing feature-based similarity estimation approaches for tabular data were implemented. We extended the probabilistic and scalable similarity estimation approach MinHash Locality Sensitivity Hashing that corresponds to the Jaccard index. For KG-based feature extraction, new downstream feature extraction transformers were developed. These are generic, modular, and easy to configure. We enhanced the pipeline with the semantification of the results. So original data remains linked to the pipeline's results and metadata. These integrated and semantified results increase reusability and reproducibility. Through sample data, sample pipelines, and integration into the open-source SANSAs GitHub repository, increased accessibility is guaranteed, complemented

by Scala docs, ReadMes, and unit tests. Through our developments, we provide a scalable native RDF KG similarity estimation pipeline.

**RQ2: How can we assist SPARQL query-based feature extraction for RDF KGs?** Our development of the Literal2Feature module supports the creation of explainable feature vectors by automatically generating feature-extracting SPARQL queries. For this purpose, the corresponding SPARQL query structure is derived by automatically traversing the RDF KG. The resulting SPARQL query uses URI information, especially the relation patterns, to derive comprehensible, understandable, and explainable projection variables. The resulting feature matrices can be used for classical downstream machine learning (ML) pipelines. The approach was successfully evaluated on a multi-node cluster over large amounts of data. We were able to present excellent results for literal-feature-based ML pipeline predictions. This approach makes it much easier for beginners and advanced users to obtain data of interest to ML from source RDF-KGs. Mainly we were capable of simplifying the otherwise complex and error-prone creation of feature-extracting SPARQL queries even in setups without existing ontologies.

**RQ3: Can we perform scalable KG literal value-based downstream ML pipelines?** Scalable downstream ML pipelines on RDF KGs are possible even with the addition of multi-modal literal values. We framed the existing approaches for scalable distributed ML pipelines through Apache Spark MLlib by the developments of DistRDF2ML to operate natively on RDF KGs. The feature extracting SPARQL queries automatically generated in Literal2Feature can map the KG structure and associated information alongside manually created ones. Through our new SparqlFrame Transformer, these SPARQL queries will result in Apache Spark DataFrames. In particular, SPARQLIFY is used as SPARQL-to-SQL rewriting. For the transformation of the features to the required fixed-length numeric feature vectors, we introduce the newly developed SmartVectorAssembler. This new transformer allows the subsequent arbitrary stacking of further established ML methods, allowing pipelines for classification or regression. A newly created module links the prediction data with the output KG data and the setup of the downstream ML pipeline. This semantification supports reusability, reproducibility, explainability, and accessibility. The approach has been successfully evaluated with real-world data on multi-node clusters and shows scalability over variable data sizes. Setup and hyper-parameters were also evaluated for their effect on efficiency and scalability, and we derived and presented best practices. This newly created resource is integrated into the open-source GitHub repository. The resulting new version is available as a separate release, as well as related documentation, ReadMes, Scala-Docs, sample pipelines, sample Databricks notebooks, and unit tests. With the DistRDF2ML enhancements, we enable many interested stakeholders to develop distributed scalable ML pipelines on KGs and run them on existing Apache Spark environments.

**RQ4: Can we perform Explainable Semantic Similarity Estimation on large-scale multi-modal RDF KGs?** We achieved explainable distributed semantic similarity estimation for multi-modal KGs with our new SimE4KG resource. SimE4KG combines several existing techniques and extends them. The feature extraction of the DistSim similarity estimation approach has been replaced by the ideas of DistRDF2ML such that the multi-modal features of the KG are now available in Apache Spark DataFrames. At the same time, we improved the performance by alternative feature extraction transformers without needing significant SPARQL query execution. Based on the vector construction

concepts of the SmartFeatureExtractor and different similarity scores, feature modality-specific scores can be calculated. The user can also adjust these to produce an explainable overall similarity score. This score is then associated with the pipeline specifics and the source data by an extended semantification. Through our novel weighting schema, we can handle domain-specific, data-specific, and user-specific knowledge and aggregate this information according to the calculated similarities within an overall score. In the process, explainable meta information is also mapped via the scores. These semantified results include the most influential factor for a similarity score. We evaluated the SimE4KG approach over large datasets on a multi-node cluster. We measured the scalability over different processing power and dataset configurations through various experiments. The modular structure and the integration into the GitHub repository SANSA allow reasonable access to the resource. Our deployed resource consists of the open-source code, documentation, release, Scala doc, Databricks notebooks, sample pipeline files, and unit tests.

**RQ5: Which Ethical and sustainable dimensions can we consider in KG-based ML?** With our work on ethical and sustainability considerations for KGs-based machine learning in Chapter 9, we elaborate on the individual dimensions using the typical R&D lifecycle. Ethical and sustainability implications can be found in each of the R&D pipeline steps. By an initial definition, highlighting challenges, giving examples, stating research questions, and suggesting optimizations, we show how the research and development of KG-based ML can be optimized holistically. In doing so, many of the considerations presented address ML's unique requirements and circumstances on KG data that distinguish it from other AI ethics approaches. Nevertheless, we list points that should not be neglected in the development and can be generally assigned to Sustainable AI and AI Ethics. This review forms the starting point for evaluation frameworks and supporting tools for optimizing KG-based ML (see Section 10.2).

## 10.2 Future Work

In addition to the broad research, developments, and answers to the research questions, we would like to present future directions in this section.

### Scalable Distributed Machine Learning for KGs

- **Knowledge Graph Embedding:** In KG-based ML, we see the potential to merge the explainable and KGE embedding approaches.
  - *Hybrid KGE:* This could lead to a hybrid KGE and query-based approach that more clearly addresses the values in the literals. One approach could be the further development of LiteralE [57] or determining the starting points of the embeddings not randomly but by the developed extracted features.
  - *Explainable KGE:* As common KGE approaches start from random vectors and optimize the vector representation to meet the structural representation of the graph by minimizing a mathematically defined loss, the resulting KGE vector entries are not explainable. Approaches like distilling neural networks [120] try to replace latent weighting matrices with explainable decision trees. An identification process of KGE vector entry effects can support later identification of the importance of certain vector representations. This

assignment of feature vector entries to their effect in KG-based ML tasks can assign the existing latent embeddings an explainable meaning.

- *Out-of-sample KGE*: Especially for large-scaled KGEs, it is essential that changing data does not require a KGE creation from scratch. This capability is described in the context of semantic-linked data and KG by inductive link prediction or out-of-sampling KGE. It would be desirable to combine these approaches with scalable distributed systems.
- **Scalable Graph ML Frameworks**: For this purpose, existing frameworks can be used and further developed. There are developments of distributed KG embedding like Pytorch BigGraph, and DGL-KE, which come from the enterprise KG using companies Facebook and Amazon. In addition, BigDL, the framework developed by Intel, is available. BigDL makes complex ML optimization models, such as Deep Neural networks, possible on Apache Spark.
  - *Multi modal features*: Different data types may be covered in the downstream ML pipelines and similarity estimations in the future concerning multi-modal endpoints. KGs might also include data modalities like images, audio, or video, which can be included in the feature vector representations. This integrated vector representation could allow improved, e.g., recommendation systems using one feature vector for all multi-modal data.
  - *Recommendation systems*: In conjunction with the exciting data sources, the resulting various downstream ML applications can be used in data analytics pipelines such as recommendation systems. More accurate and user-adjustable recommendations and predictions can be performed through the introduced weighting schemas (see Sect. 7.2.5).
  - *Scalability optimizations*: For further evaluation of scalability optimizations, several options can be explored. These include partitioning and data compression approaches like the initial compression of source KG [151] but also compression [11, 152] of KGE as default processed steps within the ML pipeline modules. Special optimizations in the code can increase the efficiency of scalable distributed processing. These include partitioning, shuffling, and caching. Partitioning describes the data distribution strategies over several nodes in the cluster so that the number of necessary communications between the nodes is minimal. Shuffling is about data aggregation and redistribution strategies across the cluster between process steps.
  - *Benchmark of distributed ML tools for KGs*: As a starting point, one can evaluate novel and existing frameworks available to implement scalable KG-based ML. Both scalability and the set of available features should be compared. Besides the already mentioned options in the Apache Spark landscape with Apache Spark MLlib, Apache Spark GraphX, and BigDL, there are also options like Apache Giraph based on distributed scalable computing.
  - *Probabilistic graph algorithms*: Probabilistic approaches can substitute algorithmic approaches that are not sufficiently scalable. If used correctly, these can enable a trade-off between processing time and required accuracy.

### Ethical and Sustainability dimensions of KG-based ML

- **AI Ethics and Sustainability Evaluation Framework KG-based ML:** Chapter 9 presents the dimensions and considerations of ethical implications and sustainability optimizations for KG-based ML. A variety of generic libraries and well as frameworks have been developed to create KG-based ML tools. Likewise, a variety of use cases are implemented that use KG-based ML. With the help of an evaluation framework, we want to create a standard for evaluating the individual dimensions and thus create transparency for the technical implementation and improvement potentials. In particular, the dimensions of data, model, sustainability, and semanticization will be evaluated.
  - *Critical features in KG:* In the context of the data collection, which critical features are contained in the source data? What efforts have been made to clean or pseudonymize the data? To what extent do the features exhibit a bias across the dataset? How was the data obtained, automatically, synthetically, curated, or manually?
  - *ML model characteristics:* For the model specifications, propose an evaluation scheme for the following questions: To what extent is the model explainable? Can the model handle out-of-sample data? Can the model perform online learning? Can humans influence ML behavior and provide intervention or feedback?
  - *Sustainability:* Another set of evaluation options is related to sustainability optimizations of the KG-ML model. Has the model been pre-trained on a system powered by renewable energy?
  - *Result Semanticization:* Finally, we provide a set of evaluation dimensions related to the data semanticization: Does the model offer results as semantic native data enriching the original KG? Does the process export semantic ML pipeline metadata to improve reproducibility?
- **Critical Features and Bias ML Data Analytics Tool:** Based on the existing tools for tracking ML CO2 footprint and the known data analytics tools, we would like to develop a library that can be integrated into classical downstream pipelines and develop KG-based ML under AI ethics. The main goal is to offer an analysis and improvement of KGs as source data in the ML pipeline. Also, transparent handling of the data and its weaknesses should be supported by reporting possibilities. In particular, the tool should have the following functions:
  - *Critical feature analysis:* Collection of information associated with discrimination or personal data such as gender, age, political or sexual orientation, and ethnicity.
  - *Privacy protection mechanism:* Ability to remove, pseudonymize, or randomize these critical features.
  - *Bias analytics:* Highlight feature bias across sample sets.
  - *Report generation:* Automatically generate a report of the source data, data curation, and feature-data bias.
  - *Bias reduction:* Train-test-split sampling options to reduce bias in the training data.

### 10.3 Closing Remarks

The growing number of Semantic Linked Data sources and RDF KGs is a valuable resource for AI, data analytics, and ML processes. Our work shows how scalable learning can be implemented on KGs by combining distributed processing, semantic technologies, and ML. In particular, we developed five main contributions: 1) scalable semantic similarity estimation, 2) automatic feature extracting SPARQL query generation, 3) distributed downstream ML pipelines for KGs, 4) explainable multi-modal similarity assessment for KGs, and finally, 5) ethical and sustainability dimensions overview in KG based ML approaches. The contributions can be found in the open-source established project SANSa. This SANSa ML stack is applied in several European projects and thus influences the Semantic Web community. The mixture of technical implementations and broad theoretical dimensions like impact optimization can lay a foundation for new developments in KG-based ML.



## Bibliography

---

- [1] T. Berners-Lee and J. Hendler, *Publishing on the semantic web*, *Nature* **410** (2001) 1023 (cit. on pp. 1, 14, 26).
- [2] E. Miller, *An introduction to the resource description framework*, *Bulletin of the American Society for Information Science and Technology* **25** (1998) 15 (cit. on pp. 2, 14).
- [3] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, *Dbpedia-a crystallization point for the web of data*, *Journal of web semantics* **7** (2009) 154 (cit. on p. 2).
- [4] F. M. Suchanek, G. Kasneci, and G. Weikum, “Yago: a core of semantic knowledge,” *Proceedings of the 16th international conference on World Wide Web*, 2007 697 (cit. on pp. 2, 57, 97).
- [5] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: a collaboratively created graph database for structuring human knowledge,” *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ACM SIGMOD, 2008 1247 (cit. on pp. 2, 29, 97).
- [6] O. Hassanzadeh and M. P. Consens, “Linked movie data base,” *LDOW*, 2009 (cit. on pp. 2, 24, 29, 66, 82).
- [7] G. A. Miller, *WordNet: a lexical database for English*, *Communications of the ACM* **38** (1995) 39 (cit. on pp. 2, 29).
- [8] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, *Industry-scale knowledge graphs: lessons and challenges*, *Communications of the ACM* **62** (2019) 36 (cit. on pp. 2, 97).
- [9] K. J. Vang, *Ethics of Google’s Knowledge Graph: some considerations*, *Journal of Information, Communication and Ethics in Society* (2013) (cit. on pp. 2, 97, 102).
- [10] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, “Translating embeddings for modeling multi-relational data,” *Neural Information Processing Systems (NIPS)*, NIPS, 2013 1 (cit. on pp. 3, 24–26, 32, 51, 57).
- [11] M. Galkin, E. Denis, J. Wu, and W. L. Hamilton, “NodePiece: Compositional and Parameter-Efficient Representations of Large Knowledge Graphs,” *International Conference on Learning Representations*, 2021 (cit. on pp. 3, 24, 31, 32, 99–101, 103, 105, 106, 112).

- [12] D. Zheng, X. Song, C. Ma, Z. Tan, Z. Ye, J. Dong, H. Xiong, Z. Zhang, and G. Karypis, “Dgl-ke: Training knowledge graph embeddings at scale,” *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020 739 (cit. on pp. 3, 29, 30, 32, 103–105).
- [13] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al., *Apache Spark: a unified engine for big data processing*, *Communications of the ACM* **59** (2016) 56 (cit. on pp. 4, 19, 29, 30, 32, 75, 76, 78).
- [14] J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, I. Ermilov, S. Bin, N. Chakraborty, M. Saleem, A. C. Ngonga Ngomo, and H. Jabeen, *Distributed semantic analytics using the SANSA stack*, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **10588 LNCS** (2017) 147, ISSN: 16113349 (cit. on pp. 4, 17, 32, 33, 46, 57, 75, 91).
- [15] A. J. Foundation, *Apache Jena*, <https://jena.apache.org/index.html>, 2022 (cit. on pp. 4, 17, 30, 75, 78).
- [16] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., *Mllib: Machine learning in apache spark*, *The Journal of Machine Learning Research* **17** (2016) 1235 (cit. on pp. 4, 19, 24, 25, 29, 32, 57, 63, 75, 76, 81).
- [17] 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*, ed. by C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, Springer International Publishing, 2019 293, ISBN: 978-3-030-30796-7 (cit. on pp. 4, 11, 24, 30, 32, 55, 60, 76, 81, 96).
- [18] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao, *Ontop: Answering SPARQL queries over relational databases*, *Semantic Web* **8** (2017) 471 (cit. on p. 4).
- [19] Databricks-Inc., *Databricks platform*, 2021, URL: <https://databricks.com/product/data-lakehouse> (visited on 2021) (cit. on pp. 5, 31, 91).
- [20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., *Scikit-learn: Machine learning in Python*, *the Journal of machine Learning research* **12** (2011) 2825 (cit. on pp. 5, 24, 75, 81).
- [21] L. F. W. Anthony, B. Kanding, and R. Selvan, *Carbontracker: Tracking and predicting the carbon footprint of training deep learning models*, arXiv preprint arXiv:2007.03051 (2020) (cit. on pp. 5, 31, 32, 97, 100, 101, 105–107).

- 
- [22] C. F. Draschner, J. Lehmann, and H. Jabeen, “DistSim-Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs,” *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, IEEE, 2021 333 (cit. on pp. 9, 23, 25, 33, 75–77, 81, 90, 91).
- [23] F. Bakhshandegan Moghaddam, C. Draschner, J. Lehmann, and H. Jabeen, “Literal2Feature: An automatic scalable RDF graph feature extractor,” *Further with Knowledge Graphs, Proceedings of the 17th International Conference on Semantic Systems, SEMANTICS 2021, Amsterdam, The Netherlands, September 6-9, 2021*, IOS Press, 2021 74 (cit. on pp. 10, 11, 23, 25, 45, 57, 60, 64, 75, 76, 86, 90, 91, 103).
- [24] 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*, Association for Computing Machinery, 2021 4465, ISBN: 9781450384469 (cit. on pp. 10, 23, 25, 26, 55, 57, 75–77, 81, 86, 90, 91, 104).
- [25] C. F. Draschner, F. B. Moghaddam, J. Lehmann, and H. Jabeen, *Semantic Analytics in the Palm of your Browser*, Big Data Analytics 3rd Summer School **3** (2021) (cit. on pp. 10, 90).
- [26] F. B. Moghaddam, C. F. Draschner, J. Lehmann, and H. Jabeen, *Semantic Web Analysis with Flavor of Micro-Services*, Big Data Analytics 3rd Summer School **3** (2021) (cit. on pp. 10, 93, 96).
- [27] C. F. Draschner, H. Jabeen, and J. Lehmann, “SimE4KG: Distributed Explainable multi-modal Semantic Similarity Estimation for Knowledge Graphs,” *2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, IEEE, 2022 1 (cit. on pp. 10, 23, 25, 40, 73, 90, 91).
- [28] C. F. Draschner, H. Jabeen, and J. Lehmann, “Ethical and Sustainability Considerations for Knowledge Graph based Machine Learning,” *2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, IEEE, 2022 53 (cit. on pp. 11, 23, 97).
- [29] L. Yu, “Linked open data,” *A Developer’s Guide to the Semantic Web*, Springer, 2011 409 (cit. on pp. 13, 29, 33).
- [30] H. Paulheim, *Knowledge graph refinement: A survey of approaches and evaluation methods*, *Semantic web* **8** (2017) 489 (cit. on p. 13).
- [31] O. Hartig, “Foundations of RDF\* and SPARQL\*:(An alternative approach to statement-level metadata in RDF),” *AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017*. Vol. 1912, Juan Reutter, Divesh Srivastava, 2017 (cit. on p. 13).
- [32] J. Pérez, M. Arenas, and C. Gutierrez, *Semantics and complexity of SPARQL*, *ACM Transactions on Database Systems (TODS)* **34** (2009) 1 (cit. on p. 14).
- [33] W3C, *RDF Ntriple W3C Examples*, 2022, URL: <https://www.w3.org/TR/n-triples/> (visited on 12/25/2022) (cit. on p. 15).

- [34] D. L. McGuinness, F. Van Harmelen, et al., *OWL web ontology language overview*, W3C recommendation **10** (2004) 2004 (cit. on pp. 15, 16).
- [35] N. Guarino, D. Oberle, and S. Staab, “What is an ontology?” *Handbook on ontologies*, Springer, 2009 1 (cit. on p. 15).
- [36] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati, “Linking data to ontologies,” *Journal on data semantics X*, Springer, 2008 133 (cit. on p. 15).
- [37] M. Uschold and M. Gruninger, *Ontologies: Principles, methods and applications*, *The knowledge engineering review* **11** (1996) 93 (cit. on p. 15).
- [38] D. Fensel, “Ontologies,” *Ontologies*, Springer, 2001 11 (cit. on p. 15).
- [39] M. N. Mami, *Strategies for a Semantified Uniform Access to Large and Heterogeneous Data Sources*, PhD thesis: Universitäts-und Landesbibliothek Bonn, 2021 (cit. on p. 15).
- [40] T. Berners-Lee, *Linked Data Design Issues*, URL: <https://www.w3.org/DesignIssues/LinkedData.html> (visited on 10/20/2022) (cit. on p. 15).
- [41] 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)*, Ieee, 2010 1 (cit. on p. 18).
- [42] A. S. Foundation, *Apache Spark*, <https://spark.apache.org>, 2021 (cit. on pp. 19, 57).
- [43] J. Dean and S. Ghemawat, *MapReduce: simplified data processing on large clusters*, *Communications of the ACM* **51** (2008) 107 (cit. on p. 19).
- [44] A. S. Foundation, *Apache Spark MLlib*, <https://spark.apache.org/mllib/>, 2021 (cit. on p. 19).
- [45] A. Spark, *Spark MLlib ML Pipeline 3.3.0*, 2022, URL: <https://spark.apache.org/docs/latest/ml-pipeline.html> (visited on 12/25/2022) (cit. on p. 19).
- [46] M. Odersky, L. Spoon, and B. Venners, *Programming in scala*, Artima Inc, 2008 (cit. on p. 20).
- [47] O. E. Dictionary, *Artificial Intelligence*, 2022, URL: <https://www.oed.com/viewdictionaryentry/Entry/271625> (visited on 05/23/2022) (cit. on p. 21).
- [48] O. E. Dictionary, *Machine Learning*, 2022, URL: <https://www.oed.com/view/Entry/111850?redirectedFrom=machine+learning+#eid38479194> (visited on 05/23/2022) (cit. on p. 21).
- [49] D. Greene, A. L. Hoffmann, and L. Stark, “Better, nicer, clearer, fairer: A critical assessment of the movement for ethical artificial intelligence and machine learning,” *Proceedings of the 52nd Hawaii international conference on system sciences*, 2019 (cit. on pp. 22, 99, 103, 104).

- 
- [50] S. L. Piano, *Ethical principles in machine learning and artificial intelligence: cases from the field and possible ways forward*, Humanities and Social Sciences Communications **7** (2020) 1 (cit. on pp. 22, 31, 32, 97, 99, 103, 104, 107).
- [51] L. Floridi, J. Cowls, T. C. King, and M. Taddeo, “How to design AI for social good: Seven essential factors,” *Ethics, Governance, and Policies in Artificial Intelligence*, Springer, 2021 125 (cit. on pp. 22, 31, 32, 97, 99, 102–105, 107).
- [52] N. J. Goodall, *Ethical decision making during automated vehicle crashes*, Transportation Research Record **2424** (2014) 58 (cit. on p. 22).
- [53] H. Ye, L. Liang, G. Y. Li, J. Kim, L. Lu, and M. Wu, *Machine learning for vehicular networks: Recent advances and application examples*, ieev vehicular technology magazine **13** (2018) 94 (cit. on p. 22).
- [54] A. van Wynsberghe, *Sustainable AI: AI for sustainability and the sustainability of AI*, AI and Ethics (2021) 1 (cit. on pp. 22, 31, 32, 97, 99, 100, 107).
- [55] M. Agravante, *MIT moves toward greener, more sustainable artificial intelligence*, 2020, URL: <https://inhabitat.com/mit-moves-toward-greener-more-sustainable-artificial-intelligence/> (visited on 05/31/2022) (cit. on p. 22).
- [56] E. Alpaydin, *Introduction to machine learning*, MIT press, 2020 (cit. on p. 24).
- [57] A. Kristiadi, M. A. Khan, D. Lukovnikov, J. Lehmann, and A. Fischer, “Incorporating literals into knowledge graph embeddings,” *International Semantic Web Conference*, Springer, 2019 347 (cit. on pp. 24, 111).
- [58] 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*, vol. 813, CEUR Workshop Proceedings, CEUR-WS.org, 2011, URL: <http://ceur-ws.org/Vol-813/ldow2011-paper07.pdf> (cit. on pp. 24, 32).
- [59] 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 pp. 24, 32).
- [60] 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. 24).
- [61] 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*, ACM SIGKDD, 2016 855 (cit. on pp. 24, 25).
- [62] 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*, 2014 701 (cit. on pp. 24, 25).

- [63] P. Ristoski and H. Paulheim, “Rdf2vec: Rdf graph embeddings for data mining,” *International Semantic Web Conference*, Springer, 2016 498 (cit. on pp. 24, 25, 32, 51, 57).
- [64] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, *Weisfeiler-lehman graph kernels.*, *Journal of Machine Learning Research* **12** (2011) (cit. on pp. 24, 25).
- [65] B. Yang, S. W.-t. Yih, X. He, J. Gao, and L. Deng, “Embedding Entities and Relations for Learning and Inference in Knowledge Bases,” *Proceedings of the International Conference on Learning Representations (ICLR) 2015*, 2015 (cit. on pp. 24, 25, 32, 51, 57).
- [66] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, *The anatomy of the facebook social graph*, arXiv preprint arXiv:1111.4503 (2011) (cit. on p. 24).
- [67] Z. Wang, T. Chen, J. Ren, W. Yu, H. Cheng, and L. Lin, “Deep reasoning with knowledge graph for social relationship understanding,” *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018 1021 (cit. on p. 24).
- [68] P. Ristoski, C. Bizer, and H. Paulheim, *Mining the Web of Linked Data with RapidMiner*, *J. Web Semant.* **35** (2015) 142 (cit. on pp. 24, 25, 32).
- [69] 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. 24, 51).
- [70] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781 (2013) (cit. on pp. 25, 26, 61).
- [71] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp, and J. Lehmann, *PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings.*, *J. Mach. Learn. Res.* **22** (2021) 1 (cit. on pp. 26, 29, 30, 32).
- [72] H. Rubenstein and J. B. Goodenough, *Contextual correlates of synonymy*, *Communications of the ACM* **8** (1965) 627 (cit. on pp. 26, 86).
- [73] G. A. Miller and W. G. Charles, *Contextual correlates of semantic similarity*, *Language and cognitive processes* **6** (1991) 1 (cit. on pp. 26, 86).
- [74] V. M K and K. K, *A Survey on Similarity Measures in Text Mining*, *Machine Learning and Applications: An International Journal* **3** (2016) 19 (cit. on p. 26).
- [75] J. J. Lastra-Diaz, A. Garcia-Serrano, M. Batet, M. Fernandez, and F. Chirigati, *HESML: A scalable ontology-based semantic similarity measures library with a set of reproducible experiments and a replication dataset*, *Information Systems* **66** (2017) 97 (cit. on pp. 26, 29).
- [76] D. Sánchez, M. Batet, D. Isern, and A. Valls, *Ontology-based semantic similarity: A new feature-based approach*, *Expert systems with applications* **39** (2012) 7718 (cit. on pp. 26–28, 38).

- 
- [77] T. Slimani, *Description and Evaluation of Semantic Similarity Measures Approaches*, International Journal of Computer Applications **80** (2013) 25 (cit. on p. 26).
- [78] G. Zhu and C. A. Iglesias, *Computing semantic similarity of concepts in knowledge graphs*, IEEE Transactions on Knowledge and Data Engineering **29** (2016) 72 (cit. on p. 26).
- [79] R. Rada, H. Mili, E. Bicknell, and M. Blettner, *Development and application of a metric on semantic nets*, IEEE transactions on systems, man, and cybernetics **19** (1989) 17 (cit. on p. 27).
- [80] R. Richardson, A. Smeaton, and J. Murphy, *Using WordNet as a knowledge base for measuring semantic similarity between words*, 1994 (cit. on p. 27).
- [81] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, 1994 133 (cit. on p. 27).
- [82] D. Lin, "Principle-based parsing without overgeneration," *31st annual meeting of the association for computational linguistics*, 1993 112 (cit. on p. 27).
- [83] D. Lin et al., "An information-theoretic definition of similarity.," *Icml*, vol. 98, 1998, 1998 296 (cit. on p. 27).
- [84] P. Resnik, *Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language*, Journal of artificial intelligence research **11** (1999) 95 (cit. on p. 27).
- [85] J. Braun-Blanquet, *Pflanzensoziologie: grundzüge der vegetationskunde*, Springer-Verlag, 2013 298 (cit. on pp. 27, 28, 38).
- [86] T. A. Sorensen, *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons*, Biol. Skar. **5** (1948) 1 (cit. on pp. 27, 28, 38).
- [87] P. Jaccard, *Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines*, Bulletin de la Société Vaudoise des Sciences Naturelles **37** (1901) 241, ISSN: 03600300 (cit. on pp. 27, 28, 38).
- [88] A. Ochiai, *Zoogeographical studies on the soleoid fishes found in Japan and its neighbouring regions-I*, Bull. Jpn. Soc. scient. Fish. **22** (1957) 522 (cit. on pp. 27, 28, 38).
- [89] G. G. Simpson, *Mammals and the nature of continents*, American Journal of Science **241** (1943) 1 (cit. on pp. 27, 28, 38).
- [90] A. Tversky, *Features of similarity.*, Psychological review **84** (1977) 327 (cit. on pp. 27, 28, 38).
- [91] A. Z. Broder, "On the resemblance and containment of documents," *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, IEEE, 1997 21 (cit. on pp. 28, 29, 32, 38).

- [92] A. Gakhov, *Probabilistic Data Structures and Algorithms for Big Data Applications*, BoD–Books on Demand, 2019 (cit. on pp. 28, 29, 32, 34).
- [93] S. Harispe, D. Sánchez, S. Ranwez, S. Janaqi, and J. Montmain, *A framework for unifying ontology-based semantic similarity measures: A study in the biomedical domain*, *Journal of biomedical informatics* **48** (2014) 38 (cit. on p. 29).
- [94] J. Wang, H. T. Shen, J. Song, and J. Ji, *Hashing for similarity search: A survey*, arXiv preprint arXiv:1408.2927 (2014) (cit. on p. 29).
- [95] R. B. Zadeh and A. Goel, *Dimension independent similarity computation*, *The Journal of Machine Learning Research* **14** (2013) 1605 (cit. on pp. 29, 32, 38, 76, 81).
- [96] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, *Pytorch-biggraph: A large scale graph embedding system*, *Proceedings of Machine Learning and Systems* **1** (2019) 120 (cit. on pp. 29, 30, 32, 103–105).
- [97] J. J. Dai, Y. Wang, X. Qiu, D. Ding, Y. Zhang, Y. Wang, X. Jia, C. L. Zhang, Y. Wan, Z. Li, et al., “Bigdl: A distributed deep learning framework for big data,” *Proceedings of the ACM Symposium on Cloud Computing*, ACM, 2019 50 (cit. on pp. 29, 63).
- [98] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al., *Dbpedia—a large-scale, multilingual knowledge base extracted from wikipedia*, *Semantic web* **6** (2015) 167 (cit. on pp. 29, 57, 97).
- [99] M. Needham and A. E. Hodler, *Graph algorithms: practical examples in Apache Spark and Neo4j*, O’Reilly Media, 2019 (cit. on p. 30).
- [100] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, “Pregel: a system for large-scale graph processing,” *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010 135 (cit. on p. 30).
- [101] C. Martella, R. Shaposhnik, D. Logothetis, and S. Harenberg, *Practical graph analytics with apache giraph*, vol. 1, Springer, 2015 (cit. on p. 30).
- [102] J. S. Andersen and O. Zukunft, “Evaluating the scaling of graph-algorithms for big data using graphx,” *2016 2nd International Conference on Open and Big Data (OBD)*, IEEE, 2016 1 (cit. on p. 30).
- [103] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. Hellerstein, “GraphLab: a new framework for parallel machine learning,” *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, 2010 340 (cit. on p. 30).
- [104] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank citation ranking: Bringing order to the web.*, tech. rep., Stanford InfoLab, 1999 (cit. on p. 30).



- 
- [105] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, *One trillion edges: Graph processing at facebook-scale*, Proceedings of the VLDB Endowment **8** (2015) 1804 (cit. on p. 30).
- [106] X. Han, S. Cao, X. Lv, Y. Lin, Z. Liu, M. Sun, and J. Li, “Openke: An open toolkit for knowledge embedding,” *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations*, 2018 139 (cit. on p. 30).
- [107] 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*, 2019 2494 (cit. on p. 30).
- [108] G. Sejdiu, *Efficient Distributed In-Memory Processing of RDF Datasets*, PhD thesis: Universitäts- und Landesbibliothek Bonn, 2020 (cit. on pp. 30, 57).
- [109] G. Sejdiu, I. Ermilov, J. Lehmann, and M. N. Mami, *Distlodstats: Distributed computation of rdf dataset statistics*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **11137 LNCS** (2018) 206, ISSN: 16113349 (cit. on p. 30).
- [110] 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),” *European Knowledge Acquisition Workshop*, Springer, 2018 534 (cit. on p. 30).
- [111] R. Dadwal, D. Graux, G. Sejdiu, H. Jabeen, and J. Lehmann, “Clustering pipelines of large RDF POI data,” *European Semantic Web Conference*, Springer, 2019 24 (cit. on p. 30).
- [112] D. Esteves, D. Moussallem, C. B. Neto, T. Soru, R. Usbeck, M. Ackermann, and J. Lehmann, “MEX vocabulary: a lightweight interchange format for machine learning experiments,” *Proceedings of the 11th International Conference on Semantic Systems*, 2015 169 (cit. on pp. 31, 32, 107).
- [113] G. C. Publio, D. Esteves, A. Ławrynowicz, P. Panov, L. Soldatova, T. Soru, J. Vanschoren, and H. Zafar, “ML schema: exposing the semantics of machine learning with schemas and ontologies,” *ICML 2018 Workshop on Reproducibility in Machine Learning*, 2018 (cit. on pp. 31, 32, 107).
- [114] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al., *The FAIR Guiding Principles for scientific data management and stewardship*, Scientific data **3** (2016) 1 (cit. on pp. 31, 32, 103).
- [115] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al., *Jupyter Notebooks—a publishing format for reproducible computational workflows*. Vol. 2016, 2016 (cit. on pp. 31, 91).

- [116] N. M. Barbosa and M. Chen, “Rehumanized crowdsourcing: a labeling framework addressing bias and ethics in machine learning,” *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019 1 (cit. on pp. 31, 32, 104, 107).
- [117] I. Tiddi and S. Schlobach, *Knowledge graphs as tools for explainable machine learning: A survey*, *Artificial Intelligence* **302** (2022) 103627 (cit. on pp. 31, 32, 104).
- [118] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, et al., *Never-ending learning*, *Communications of the ACM* **61** (2018) 103 (cit. on pp. 31, 102, 104, 105, 107).
- [119] M. Palmonari and P. Minervini, *Knowledge graph embeddings and explainable AI*, *Knowledge Graphs for Explainable Artificial Intelligence: Foundations, Applications and Challenges*, IOS Press., Amsterdam (2020) 49 (cit. on pp. 31, 32, 103).
- [120] N. Frosst and G. Hinton, *Distilling a neural network into a soft decision tree*, arXiv preprint arXiv:1711.09784 (2017) (cit. on pp. 31, 32, 103, 111).
- [121] F. Lecue, *On the role of knowledge graphs in explainable AI*, *Semantic Web* **11** (2020) 41 (cit. on pp. 31, 32, 103).
- [122] R. Huber and J. Klump, “The Dark Side of the Knowledge Graph-How Can We Make Knowledge Graphs Trustworthy?” *EGU General Assembly Conference Abstracts*, 2020 13071 (cit. on pp. 31, 99, 102–104).
- [123] I. Abdelaziz, R. Harbi, Z. Khayyat, and P. Kalnis, *A survey and experimental comparison of distributed SPARQL engines for very large RDF data*, *Proceedings of the VLDB Endowment* **10** (2017) 2049 (cit. on p. 33).
- [124] 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 p. 45).
- [125] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek, *Fast rule mining in ontological knowledge bases with AMIE+*, *VLDB J.* **24** (2015) 707 (cit. on p. 45).
- [126] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, *A Review of Relational Machine Learning for Knowledge Graphs*, *Proceedings of the IEEE* **104** (2016) 11, URL: <https://doi.org/10.1109/JPROC.2015.2483592> (cit. on p. 45).
- [127] L. Bühmann, J. Lehmann, and P. Westphal, *DL-Learner - A framework for inductive learning on the Semantic Web.*, *J. Web Semant.* **39** (2016) 15 (cit. on p. 45).
- [128] C. F. Draschner and F. Bakhshandegan Moghaddam, *Literal2Feature Tutorial*, 2021, URL: <https://github.com/SANSA-Stack/SANSA-Stack/blob/develop/sansa-ml/README.md#literal2feature-autosparql-generation-for-feature-extraction> (cit. on p. 46).

- 
- [129] E. Moore, *The Shortest Path Through a Maze*, Bell Telephone System. Technical publications. monograph, Bell Telephone System., 1959 (cit. on p. 47).
- [130] F. Xia, J. Liu, H. Nie, Y. Fu, L. Wan, and X. Kong, *Random Walks: A Review of Algorithms and Applications*, IEEE Transactions on Emerging Topics in Computational Intelligence **4** (2020) 95 (cit. on p. 47).
- [131] P. Westphal, L. Bühmann, S. Bin, H. Jabeen, and J. Lehmann, *SML-Bench - A benchmarking framework for structured machine learning*, Semantic Web **10** (2019) 231 (cit. on p. 50).
- [132] S. M. Kazemi and D. Poole, “Simple Embedding for Link Prediction in Knowledge Graphs,” *Advances in Neural Information Processing Systems*, 2018 (cit. on p. 51).
- [133] 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. 51).
- [134] T. Berners-Lee, J. Hendler, and O. Lassila, *The semantic web*, Scientific american **284** (2001) 34 (cit. on pp. 57, 97).
- [135] J. Z. Pan, “Resource description framework,” *Handbook on ontologies*, Springer, 2009 71 (cit. on p. 57).
- [136] D. Vrandečić and M. Krötzsch, *Wikidata: a free collaborative knowledgebase*, Communications of the ACM **57** (2014) 78 (cit. on pp. 57, 97).
- [137] C. F. Draschner, *DistRDF2ML Release*, [https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1\\_DistRDF2ML](https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1_DistRDF2ML), 2021 (cit. on pp. 63–66, 68, 70, 71).
- [138] C. Draschner, *SimE4KG GitHub Release*, [https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.2.3\\_SimE4KG](https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.2.3_SimE4KG) (cit. on pp. 74, 77, 81, 82).
- [139] J. Lehmann, *SANSA GitHub Repository*, <https://github.com/SANSA-Stack> (cit. on p. 74).
- [140] W3C, *RDF XSD types*, 2022, URL: <https://www.w3.org/TR/swbp-xsch-datatypes/> (visited on 12/25/2022) (cit. on p. 76).
- [141] P. Jaccard, *Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines*, Bull Soc Vaudoise Sci Nat **37** (1901) 241 (cit. on p. 77).
- [142] C. F. Draschner, *Semantic analytics in the palm of your browser - Slides*, 2021, URL: <https://github.com/SANSA-Stack/SANSA-Databricks> (visited on 2021) (cit. on p. 92).
- [143] J. Dustin, *Amazon scraps secret AI recruiting tool that showed bias against women*, 2018, URL: <https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G> (visited on 05/31/2022) (cit. on pp. 97, 104, 105).

- [144] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, *Quantifying the carbon emissions of machine learning*, arXiv preprint arXiv:1910.09700 (2019) (cit. on pp. 100, 101, 106, 107).
- [145] C. Burnicki, *Cloud Computing and Carbon Footprint*, URL: <https://www.innoq.com/en/blog/cloud-computing-and-carbon-footprint/> (visited on 05/31/2022) (cit. on pp. 100, 101).
- [146] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, *Towards the systematic reporting of the energy and carbon footprints of machine learning*, *Journal of Machine Learning Research* **21** (2020) 1 (cit. on pp. 101, 104).
- [147] A. Saltelli and S. Funtowicz, *When all models are wrong*, *Issues in Science and Technology* **30** (2014) 79 (cit. on p. 101).
- [148] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer, *Quality assessment for linked data: A survey*, *Semantic Web* **7** (2016) 63 (cit. on p. 102).
- [149] P. B. De Laat, *Algorithmic decision-making based on machine learning from big data: can transparency restore accountability?* *Philosophy & technology* **31** (2018) 525 (cit. on p. 102).
- [150] Reddit.com, *Data without context is noise! (With Zoom)*, URL: [https://www.reddit.com/r/datascience/comments/tq93vt/data\\_without\\_context\\_is\\_noise\\_with\\_zoom/](https://www.reddit.com/r/datascience/comments/tq93vt/data_without_context_is_noise_with_zoom/) (visited on 05/23/2022) (cit. on p. 103).
- [151] J. Urbani, S. Dutta, S. Gurajada, and G. Weikum, “KOGNAC: Efficient encoding of large knowledge graphs,” *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2016, 2016 3896 (cit. on pp. 104, 112).
- [152] M. Sachan, “Knowledge graph embedding compression,” *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020 2681 (cit. on pp. 104, 106, 112).
- [153] N. Shadowen, “Ethics and bias in machine learning: A technical study of what makes us “good”,” *The Transhumanism Handbook*, Springer, 2019 247 (cit. on p. 104).
- [154] BBC.com, *Google apologises for Photos app’s racist blunder*, URL: <https://www.oed.com/view/Entry/111850?redirectedFrom=machine+learning+#eid38479194> (visited on 05/23/2022) (cit. on pp. 104, 105).
- [155] T. Verge, *Twitter taught Microsoft’s AI chatbot to be a racist asshole in less than a day*, URL: <https://www.theverge.com/2016/3/24/11297050/tay-microsoft-chatbot-racist> (visited on 05/23/2022) (cit. on pp. 104, 105).
- [156] technologyreview.com, *Training a single AI model can emit as much carbon as five cars in their lifetimes*, URL: <https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/> (visited on 05/23/2022) (cit. on pp. 105, 106).

- 
- [157] E. Strubell, A. Ganesh, and A. McCallum,  
“Energy and Policy Considerations for Deep Learning in NLP,”  
*Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*,  
2019 3645 (cit. on pp. 105–107).
- [158] M. Albooyeh, R. Goel, and S. M. Kazemi,  
*Out-of-sample representation learning for multi-relational graphs*,  
arXiv preprint arXiv:2004.13230 (2020) (cit. on p. 106).



## List of Publications

---

- *Conference Papers (Peer-Reviewed):*

1. **Carsten Felix Draschner**, Jens Lehmann and Hajira Jabeen, *DistSim - Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs*, 2021 IEEE 15th International Conference on Semantic Computing (ICSC), 2021, pp. 333-336, DOI: 10.1109/ICSC50631.2021.00062. <https://ieeexplore.ieee.org/document/9364473>
2. Farshad Bakhshandegan Moghaddam, **Carsten Felix Draschner**, J. Lehmann and H. Jabeen, *Literal2Feature: An automatic scalable RDF graph feature extractor*, in: Further with Knowledge Graphs, International Conference on Semantic Systems, (SEMANTICS), IOS Press, 2021, pp. 74–88. DOI: 10.3233/SSW210036, <https://ebooks.iospress.nl/volumearticle/57407>
3. **Carsten Felix Draschner**, Claus Stadler, Farshad Bakhshandegan Moghaddam, Jens Lehmann, and Hajira Jabeen. 2021. *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. DOI: 10.1145/3459637.3481999, <https://dl.acm.org/doi/10.1145/3459637.3481999>
4. **Carsten Felix Draschner**, Hajira Jabeen, Jens Lehmann, "*SimE4KG: Distributed Explainable multi-modal Semantic Similarity Estimation for Knowledge Graphs*", 2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pp. 1-8, DOI: 10.1109/AIKE55402.2022.00007, <https://ieeexplore.ieee.org/document/9939143>
5. **Carsten Felix Draschner**, Hajira Jabeen, Jens Lehmann, "*Ethical and Sustainability considerations for Knowledge Graphs based Machine Learning*", 2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pp. 53-60, DOI: 10.1109/AIKE55402.2022.00015, <https://ieeexplore.ieee.org/document/9939282>

- *Workshops, Demos, and Doctoral Consortium (Peer-Reviewed):*

6. **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>
7. 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>





# List of Figures

---

1.1	Research dimensions, considerations and outcomes . . . . .	3
1.2	Research question and contributions . . . . .	7
1.3	Chapter, Research Questions and Publication Overview . . . . .	9
2.1	KG example with similar naming, inverse relations, and relation with time property .	14
2.2	Example RDF Graph correspond to file presented in Listing 2.2 . . . . .	15
2.3	Apache Spark Components . . . . .	18
2.4	Apache Spark MLLib pipeline <i>fit</i> . . . . .	19
2.5	Apache Spark MLLib pipeline <i>transform</i> . . . . .	20
4.1	DistSim pipeline architecture . . . . .	34
4.2	Sample resulting meta graph representing semantic similarity estimations . . . . .	35
4.3	Feature extraction in mode AT . . . . .	36
4.4	Feature extraction in mode AN . . . . .	37
4.5	Feature extraction in mode AR . . . . .	37
4.6	Feature extraction in mode AS . . . . .	38
4.7	Sample similarity estimation with Jaccard . . . . .	39
4.8	Synthetic generated sample data snippets . . . . .	41
4.9	DistSim processing power scalability on 10E4 movie data . . . . .	42
4.10	Data size scalability of All-Pair-Similarity estimation . . . . .	42
4.11	Effect of CountVectorizer maximal vocabulary size (maxVocabSize) . . . . .	43
4.12	Effect of CountVectorizer maximal vocabulary size (maxVocabSize) for similarity estimation mode Jaccard . . . . .	43
4.13	Effect of CountVectorizer minimal document frequency (minDf) . . . . .	44
4.14	Effect of different number of HashTables . . . . .	44
5.1	A sample RDF graph . . . . .	46
5.2	Literal2Feature execution pipeline . . . . .	47
5.3	Literal2Feature approach processing overview . . . . .	49
5.4	Depth impact on the classification result and number of extracted features . . . . .	52
5.5	Literal2Feature evaluation: depth vs. #features on the ENGIE dataset . . . . .	52
5.6	Literal2Feature evaluation: depth vs. time on the ENGIE dataset . . . . .	53
5.7	Literal2Feature evaluation: #walks vs. #features on the ENGIE dataset . . . . .	53
5.8	Literal2Feature evaluation: branching factor vs. time on synthetic data on a single machine . . . . .	53
5.9	Literal2Feature evaluation: processing power scalability on DS 4 dataset . . . . .	54
5.10	Literal2Feature evaluation: dataset size up performance evaluation over 64 Cores . .	55

## List of Figures

---

6.1	DistRDF2ML Pipeline Overview . . . . .	58
6.2	DistRDF2ML SPARQL specification options . . . . .	60
6.3	DistRDF2ML feature vector creation pipeline through SparqlFrame and SmartVec- torAssembler . . . . .	62
6.4	Semantically annotated ML result in original KG . . . . .	64
6.5	Sample graph snippet from Linked Movie Database . . . . .	67
6.6	Processing power vs processing time . . . . .	67
6.7	SPARQL complexity vs processing time . . . . .	68
6.8	Dataset (-size) and number movies on cluster and local execution vs. processing time	69
6.9	Spark setup vs processing time . . . . .	69
7.1	SimE4KG pipeline overview . . . . .	75
7.2	SimE4KG data transformations . . . . .	78
7.3	Extract of feature availability within LMDB dataset of type movies . . . . .	78
7.4	SimE4KG semantification example . . . . .	80
7.5	Evaluation dataset size scalability LSH effect . . . . .	83
7.6	Evaluation of Spark Overhead by Dataset Size . . . . .	83
7.7	Evaluation SimE4KG dataset size scalability on LMDB subsets . . . . .	84
7.8	Evaluation processing power scalability . . . . .	84
7.9	Evaluation spark cluster configuration effect . . . . .	85
7.10	Evaluation and Comparison to available SANSA modules . . . . .	85
8.1	SANSA stack structure . . . . .	87
8.2	SANSA ML Github.io for DistSim, DistRDF2ML, Literal2Feature and SimE4KG . .	88
8.3	SANSA ML ReadMe in GitHub repository, including structure and code (1/2) . . . .	88
8.4	SANSA ML ReadMe in GitHub repository, including structure and code (2/2) . . . .	88
8.5	SANSA ML example release for the SimE4KG contribution . . . . .	89
8.6	SANSA ML example class, fully executable class run-able standalone as a class or within IDE . . . . .	89
8.7	SANSA ML Databricks Notebook usable right through PaaS browser environment of Databricks . . . . .	89
8.8	High-level SANSA Rest-API system overview . . . . .	94
8.9	SANSA REST Swagger UI . . . . .	95
9.1	Ethical and Sustainable KG based ML pipeline . . . . .	98
9.2	Estimated CO2 emissions of Carbon Tracker experiment [21] . . . . .	101
9.3	Google Search KG result about actor and pop-figure "Chuck Norris", see what is stated as "Fact"; Example taken from Vang et al. [9] . . . . .	102
9.4	Sample image [150] motivating context in ML data . . . . .	103
9.5	Example discussions of AI failure in our daily life . . . . .	105
9.6	Real-time carbon intensity (gCO <sub>2</sub> eq/kWh) for Denmark (DK) and Great Britain (GB) from 2020-05-18 to 2020-05-25 shown in local time from Carbon Tracker analytics [21] 106	

## List of Tables

---

4.1	Overview of available DistSim feature extraction modes . . . . .	36
4.2	Feature set-based semantic similarity estimation formulas . . . . .	38
5.1	Literal2Feature evaluation dataset statistics (GT = ground truth) . . . . .	50
5.2	Synthetic dataset description . . . . .	50
5.3	Literal2Feature F1-Measure evaluation results . . . . .	51
5.4	Silhouette coefficient . . . . .	54
7.1	Sample multi-modal result SmartFeatureExtractor . . . . .	77
7.2	Smart Feature Extractor vs SparqlFrame. Hyperparameter: lmbd, MBP16, 38kMov .	85
8.1	Available SANSa REST endpoints and their functionalities . . . . .	96
9.1	Carbon Footprint Large Language Model [156, 157] . . . . .	106