Further with Knowledge Graphs. M. Alam et al. (Eds.)
AKA Verlag and IOS Press, 2021

doi:10.3233/SSW210036

Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor

Farshad BAKHSHANDEGAN MOGHADDAM [a,1], Carsten DRASCHNER [a], Jens LEHMANN [a], and Hajira JABEEN [b]
[a] *SDA Research Group, University of Bonn, Bonn, Germany*
[b] *University of Cologne, Cologne, Germany*

Abstract. The last decades have witnessed significant advancements in terms of data generation, management, and maintenance. This has resulted in vast amounts of data becoming available in a variety of forms and formats including RDF. As RDF data is represented as a graph structure, applying machine learning algorithms to extract valuable knowledge and insights from them is not straightforward, especially when the size of the data is enormous. Although Knowledge Graph Embedding models (KGEs) convert the RDF graphs to low-dimensional vector spaces, these vectors often lack the explainability. On the contrary, in this paper, we introduce a generic, distributed, and scalable software framework that is capable of transforming large RDF data into an explainable feature matrix. This matrix can be exploited in many standard machine learning algorithms. Our approach, by exploiting semantic web and big data technologies, is able to extract a variety of existing features by deep traversing a given large RDF graph. The proposed framework is open-source, well-documented, and fully integrated into the active community project Semantic Analytics Stack (SANSA). The experiments on real-world use cases disclose that the extracted features can be successfully used in machine learning tasks like classification and clustering.

Keywords. RDF Graph, Prepositionalization, Feature Extraction, Big Data, Distributed Computing, Scalable Analytics, SANSA

1. Introduction

With the rapidly growing amount of data available on the Internet, it becomes necessary to develop a set of tools to extract meaningful and hidden information from the online data. The Semantic Web enables a structural view of the existing data on the web and provides machine-readable formats [1] as the Resource Description Framework (RDF)[2]. RDF has been introduced by the World Wide Web Consortium[3] as a standard to model the real world in the form of entities and relations between them. RDF data are a collection of triples $< subject, predicate, object >$ which tend to have rich relationships, forming a potentially very large and complex graph-like structure. Figure 1 shows a sample RDF.

[1] Corresponding Author: Farshad Bakhshandegan Moghaddam, SDA Research Group, University of Bonn, Bonn, Germany; E-mail: farshad.moghaddam@uni-bonn.de

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

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

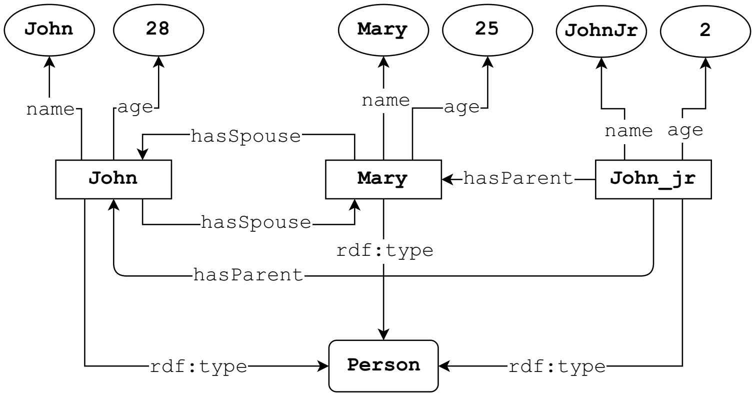


Figure 1. A Sample RDF Graph

Currently, many companies in the fields of science, engineering, and business, including bioinformatics, life sciences, business intelligence, and social networks, publish their data in the form of RDF [2][4,5]. In addition, the Linked Open Data Project initiative [3] helped the Semantic Web to gain even more attention in the past decade. Currently, the Linked Open Data (LOD) cloud comprises more than 10,000 datasets available online[6] using the RDF standard. Nowadays, RDF data can have sizes up to billions of triples[7].

Besides this, Machine Learning, a field of discovering how machines can perform tasks without being explicitly programmed to do so, is growing and finding its way in human daily life. Some prominent examples are autonomous driving, face detection, weather forecasting, etc. Recently with the rapid growth of computational power, training machine learning algorithms at scale is getting much more feasible. However, most of the well-known machine learning algorithms for classification, regression, and clustering need to work with a standard representation of data, i.e. a *feature matrix*. In this format, the data is mostly presented as a 2D matrix, in which rows present the data points and columns indicate the features. Normally, for supervised learning, one (or more) of the columns can be considered as a label (target) for the given row (data point).

Due to the complex graph nature of RDF data, applying standard machine learning algorithms to this data is cumbersome. Although there are efforts in the community to incorporate RDF graphs directly in the machine learning algorithms, they are mostly focused on the structural properties of RDF datasets [4,5,6,7] and offer limited support for RDF literals. Moreover, the challenges in the current big data era (limited computational resources) cause the traditional analytical approaches to mostly fail to operate on large-scale data.

Even though Knowledge Graph Embeddings (KGE) are getting popular as a paradigm to obtain low-dimensional feature representations of Knowledge Graphs (KG),

[4] http://www.openphacts.org

[5] https://ontop-vkg.org

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

[7] https://www.w3.org/wiki/DataSetRDFDumps

most of them do not exploit literals in their learning process (an exception is [8] for numerical literal values). Moreover, the feature vectors obtained by KGE models are latent features, which are not explainable. Latent embeddings lose locatable information such as the numerical annotation stored in literals, e.g., *salary of colleagues*, *timestamp of buying a certain item*, *runtime of a movie*. The loss of such numeric or timestamp features in multi-modal knowledge graphs need to be avoided in many use cases, especially those that use RDF for data integration and exploiting such features.

To tackle the aforementioned issues, we propose Literal2Feature, a generic, distributed, and scalable software framework which is able to automatically transform a given RDF dataset to a standard feature matrix (also dubbed *Prepositionalization*) by deep traversing the RDF graph and extracting literals to a given depth. Literal2Feature enables the use of a wide range of machine learning algorithms for the Semantic Web community. The proposed method is able to extract features automatically by creating a SPARQL query to produce the feature matrix. All steps are performed automatically without human intervention (details in Section 3). In addition, Literal2Feature is integrated into the SANSA stack [9] and interacts with the different SANSA computational layers. This integration enables sustainability, as SANSA is an actively maintained project, and uses the community ecosystem (mailing list, issue trackers, continuous integration, web-site, etc.). In contrast to KGEs, our proposed approach successfully utilizes literals as extracted features and provides high-level explainability for each feature. Moreover, our approach enables ML practitioners to select the features of interest, based on learning objectives and scenarios.

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

- Introducing a distributed generic framework that can automatically extract semantic features from an RDF graph
- Integrating the approach into the SANSA stack
- Covering the code by unit tests, documenting it in Scala docs [10] and providing a tutorial [11]
- Making Literal2Feature and the framework open source and publicly available on Github[8]
- Evaluation of the results over multiple datasets on classification and clustering scenarios, and comparing it with similar approaches
- Empirical evaluation of scalability

The rest of the paper is structured as follows: The related work is discussed in Section 2. Literal2Feature workflow, and implementation are detailed in Section 3. The use-cases are discussed in Section 4. Section 5 covers the evaluation of the Literal2Feature and demonstrates the scalability. Finally, we conclude the paper in Section 6.

2. Related Work

This section presents prior related studies on Prepositionalization, Graph Embeddings, and Machine Learning on Semantic Data.

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

Prepositionalization. In the recent years, a variety of approaches have been proposed for generating features from LOD. Many of these methods assume a manual design of the feature selection mechanism and in most situations these methods require the user to create a SPARQL query to retrieve the features. For instance, LiDDM [12] enables the users to specify SPARQL queries for retrieving features from RDF graphs that can be used in various machine learning algorithms. Similarly, an automatic feature generation approach is proposed in [13], where the user has to define the type of features in the form of SPARQL queries. Another approach, RapidMiner[9] *semweb* plugin [14] preprocesses RDF data using user-specified SPARQL queries, such that the data can be handled in RapidMiner. Another similar approach is FeGeLOD [15] and its successor, the RapidMiner Linked Open Data Extension [16]. FeGeLOD is an unsupervised approach for enriching data with features derived from LOD. This approach uses six unsupervised feature generation techniques to explore the data and fetches the features.

Our approach differs from the above-mentioned methods since it does not require any predefined SPARQL query to directly extract the features. Moreover, our approach can be scaled horizontally over a cluster of nodes to handle large amounts of data.

Graph Embeddings. Beside the above-mentioned classical methods, our work is also related to graph embeddings such as [17,18,19,20]. Although these approaches convert entities to dense vectors, however, their result is usually not explainable due to latent representations of entities. Moreover, most of them ignore literals in their learning process. RDF2Vec [17] is an approach for learning latent entity representations in RDF graphs. It first converts RDF graphs into sequences of graph random walks and Weisfeiler-Lehman graph kernels, and then adopts CBOW and Skip-gram models on the sequences to build entity representations. TransE [18] is a geometric model that assumes that the tail embeddings are close to the sum of the head and relation embeddings, according to the chosen distance function. DistMult [19] is a matrix factorization model that allows all relation embeddings to be diagonal matrices, which reduces the space of parameters to be learned and results in a relatively easy model to train. SimplE [20], like DistMult, forces the relation embeddings to be diagonal matrices but extends it by a) associating two different embeddings, $\mathbf{e_h}$ and $\mathbf{e_t}$ with each entity e, depending on whether e is head or tail b) associating two distinct diagonal matrices, $\mathbf{r}$ and $\mathbf{r_{-1}}$, with each relation r, expressing the relation in its normal and inverse direction.

Our approach differs from the KGE methods, as it only utilizes literals from knowledge graphs to generate the feature vectors while demonstrating a high-level of explainability and relatedness.

Machine Learning on Semantic Data. There are numerous centralized machine learning frameworks and algorithms for RDF data. For example, TensorLog [21] and ProPPR [4] are recent frameworks for efficient probabilistic inference in first-order logic. AMIE [5] and AMIE+ [22] learn association rules from RDF data. DL Learner [7] is a framework for inductive learning on the Semantic Web. [6] provides a review of statistical relational learning techniques for knowledge graphs.

Although there are some efforts in the community to incorporate RDF data in machine learning pipelines, however, to the best of our knowledge, Literal2Feature is the first in a row which deeply extracts literals as a feature matrix from the RDF data in a distributed

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

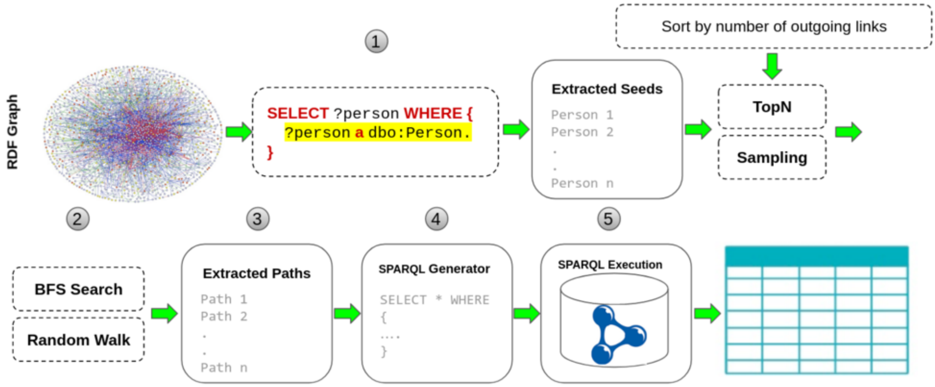


Figure 2. System Architecture Abstract Overview

and scalable manner. One may argue that these features can be extracted from the ontologies as well [23]. However, it should be noted that 1) ontologies can be large and not fully used in the data 2) there may not be any ontologies associated with a given RDF data set.

3. Literal2Feature

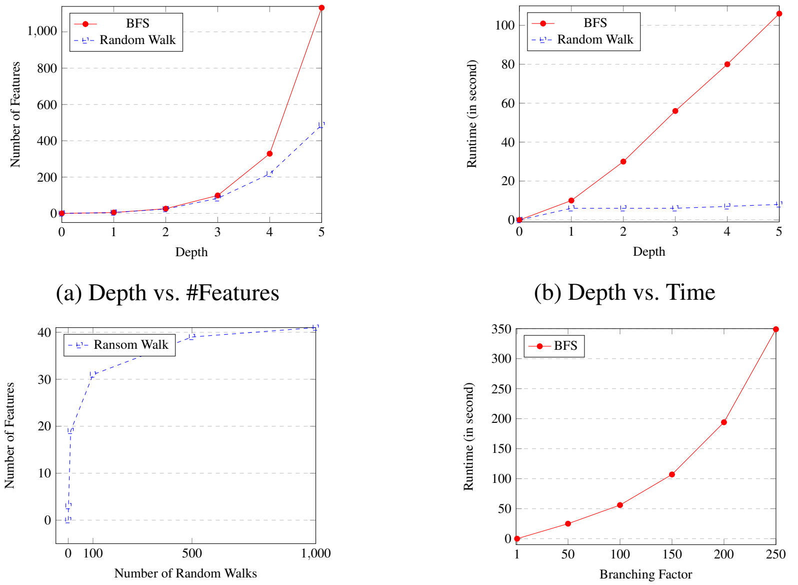
In this section, we present the system architecture of Literal2Feature. The main goal of the framework is to retrieve literals for each entity based on the predefined graph depth. In other words, Literal2Feature, by deeply traversing the RDF graph, is able to gather literals for each entity up to a predefined level and consider them as a feature vector for the given entity. We believe that literals contain valuable information for each entity which can be used for any subsequent machine learning pipeline. Although other features such as a number of specific predicates (e.g. `foaf:knows` which can count the number of friends of each person) or existence of a type relation (e.g. a boolean value which is `True` if a specific type relation exists such as `rdf:type dbo:Person`) can also be retrieved from an RDF graph, however, in Literal2Feature, we neglect them and only focus on the literals.

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

3.1. Components

3.1.1. Seed Generator Component

The input of the system is an RDF graph $\mathscr{G}$ and a set of RDF triples $\mathscr{T}$. The triples define the entities that the user is interested to generate features for. $\mathscr{T}$ will automatically formulate a `SELECT` SPARQL query which is used to generate only starting points for the deep search. In other words, this query specifies the entities for which the user is interested to extract features for (e.g. in Figure 1, all the persons). By executing the `SELECT`



(a) Depth vs. #Features

(b) Depth vs. Time

(c) #Walks vs. #Features

(d) Branching Factor vs. Time

Figure 3. Impact of different factors on the runtime and the number of extracted features (*a*, *b*, and *c* are experimented on the Engie dataset (Section 5.1) and *d* on a synthetic data on a single machine)

query over the given RDF data, the starting points of search (seeds) are generated. These seeds are sorted based on the number of outgoing links. The higher the number of outgoing links, the higher chance of generating more features (see Section 3.1.2). To execute the query, Sparklify [24], a SANSA built-in distributed SPARQL executor is used.

3.1.2. Graph Search Component

The next step executes a graph search algorithm starting from the seed nodes. Without loss of generality, in this framework we use two different strategies, a) full graph search and b) approximated graph search. For the full graph search, we use Breadth-First Search (BFS) [25] to be able to traverse the entire graph. For the approximation, we use Random Walk model [26]. In both approaches, the user will have full control over the depth, the number of walks, and direction of the search (downward, upward). There is a relation between extracted feature completeness and the search execution time with the search strategy. Full graph search is able to extract all the existing features, but the execution time is higher than the random walk model. The three factors that have an impact on the number of extracted features and the execution time are a) the depth of search b) the average branching factor of nodes of the graph c) the number of random walks (only in the Random Walk model). Figure 3 depicts the impact of each factor. It can be seen that most of the parameters have an exponential impact on the full graph search. However, the Random Walk model depicts a linear and logarithmic behavior in terms of time and the number of extracted features.

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

- Reaching a node which is a literal node

- Exceeding the pre-configured length of the walk

3.1.3. Path Generator Component

By considering the nodes and edge labels (RDF entities and properties) we could generate property paths. Each properties path encodes all needed properties to reach a literal from the given seed node. For example, based on Figure 1, the sequence of `John_jr->`*`hasParent`*`->Mary->`*`age`* encodes the path which can fetch the "age of the mother of `John_jr`". Each walk of the search algorithm generates a properties path. Due to the probabilistic nature of the algorithms (in the Random Walk method) and repetition in the RDF graphs, there is a chance of having duplicated paths. The final output of this component is distinct properties paths.

3.1.4. SPARQL Generator Component

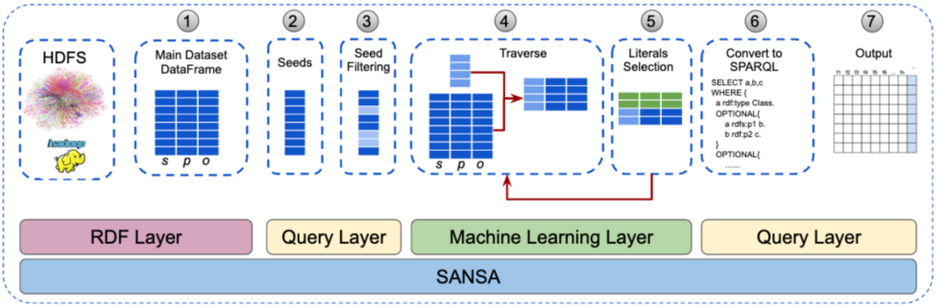
By gathering all property paths, ignoring the entities, and only keeping properties, each path is transformed automatically to a SPARQL query. For instance, the above-mentioned example will be transformed to `->`*`hasParent`*`->`*`age`* and then to the following SPARQL query (Listing 1, prefixes have been omitted for simplicity). Due to the data sparsity issue and as all the seeds may not have all the possible properties, each properties-path is wrapped with an `Optional` block to ensure the successful generation of the final result. To have explanatory names for the projected variables, the prefix part of each RDF property is omitted and after splitting based on underscore character, the last part of each property is selected and concatenated (see Listing 1). These names are human-readable and demonstrate explainability for each feature.

Listing 1: Sample result of the generated SPARQL query

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

3.1.5. SPARQL Executor Component

After generating the SPARQL query based on the generated properties paths, we use the built-in SPARQL engine in SANSA to execute the query over the given RDF data. The result of this component is the desired feature matrix. Due to the structure of the graph, the SPARQL query may result in multiple rows for a single entity. Based on Figure 1, this behavior can be seen when we want to extract the age of a parent of `John_jr` as he has two parents. In such a case, one of the rows is selected randomly and kept for the subsequent machine learning pipeline. In future work, we plan to use other strategies for merging different rows by applying aggregator functions such as taking an average, median, etc.



```
SELECT a,b,c
WHERE {
 a rdf:type Class.
 OPTIONAL{
   a rdfs:p1 b.
   b rdf:p2 c.
 }
 OPTIONAL{
   ......
```

Figure 4. Literal2Feature Execution Pipeline (Best viewed in color)

3.2. Implementation

As the programming language of SANSA is Scala[10], we have selected this language and its APIs in Apache Spark to provide the distributed implementation of Literal2Feature. Moreover, we benefit from SANSA IO and Query layers. Technically, Literal2Feature can be divided into the following steps 1) Read RDF data as a data frame 2) generate seeds 3) filter seeds 4) traverse the graph by joining data frames up to a certain depth 5) select paths ending with literals 6) convert the paths to SPARQL 7) execute the SPARQL and output the result, as shown in Figure 4 which depicts the framework execution pipeline.

4. Use Cases

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

PLATOON Digital PLatform and analytic TOOls for eNergy (PLATOON[11]) is a Horizon 2020 Project aiming to deploy distributed/edge processing and data analytics technologies for optimized real-time energy system management in a simple way for the energy domain. PLATOON uses SANSA Stack as a generic data analytics framework in which Literal2Feature is an integral module in the pipeline. Literal2Feature makes it possible for non-experts to deduce the features and enrich their use case related data.

Engie Engie SA[12] is a French multinational electric utility company that operates in the fields of energy transition, electricity generation and distribution, natural gas, nuclear, renewable energy, and petroleum. Together with Engie we are working on a dataset related to accidents that occurred in France. One of the major challenges is the prediction and classification of the accidents in an effective and scalable manner. In order to perform this task efficiently and effectively, Literal2Feature integrated into SANSA stack is an integral module for classification task.

[10] https://www.scala-lang.org/
[11] https://cordis.europa.eu/project/id/872592/de
[12] https://www.engie.com

5. Experimental Results

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

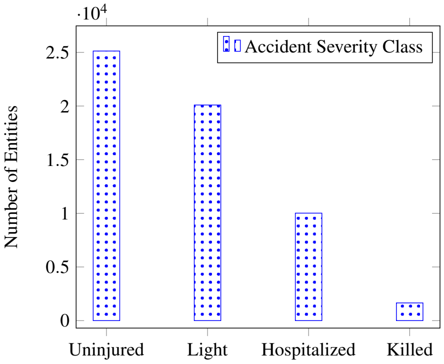


Figure 5. Entity Distribution (Accident Classification 1)

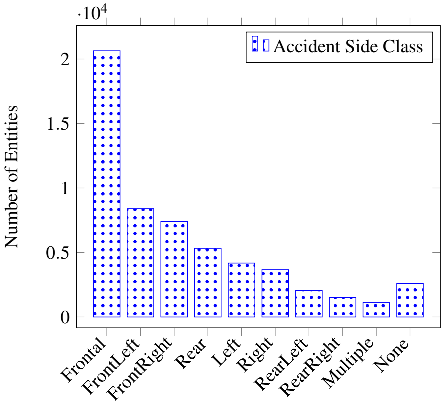


Figure 6. Entity Distribution (Accident Classification 2)

5.1. Experiment A: Assessment of the extracted Features

Literal2Feature can be used in many scenarios, from statistical analysis to classification and clustering. In this section, we analyze the quality of the extracted features for classification and clustering scenarios. To this end, two datasets have been exploited. Engie accident dataset[13] and Carcinogenesis dataset [27]. The accident dataset contains the data about accidents occurred in France in 2018. The Carcinogenesis dataset contains information about drug molecules and their features. An overview of the datasets is given in Table 1.

5.1.1. Classification

For the above-mentioned datasets, three classification scenarios have been defined as follows (all scenarios are multiclass single-label classification problem):

1- Accident classification based on how dangerous was an accident (4 classes).
2- Accident classification based on which side of the vehicle was shocked in the accident (10 classes).
3- Carcinogenic drugs classification (2 classes)

As a baseline for feature vectors, FeGeLOD [15], RDF2Vec [17], TransE [18], SimplE [20], and DistMult [19] are selected (KGEs are trained by OpenKE[14] on NVIDIA GeForce GTX 1080 Ti). For the learning algorithms, we selected Random Forest (RF), Logistic Regression (LR), Multi-Layer Perceptron (MLP), and XGBoost (XG)[28] to

[13] Can not be publicly published due to Intellectual Property concerns
[14] https://github.com/thunlp/OpenKE

Table 1. Dataset statistics (GT=Ground Truth)

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

Table 2. F1-Measure (macro) evaluation results

Approach		**Accident Scenario 1**	**Accident Scenario 2**	**Carcinogenesis**
FeGeLOD[15]	*RF*	0.17 ± 0.01	0.05 ± 0.001	0.59 ± 0.07
	LR	0.20 ± 0.02	0.05 ± 0.003	0.61 ± 0.06
	MLP	0.18 ± 0.01	0.05 ± 0.001	0.58 ± 0.07
	XG	0.18 ± 0.02	0.05 ± 0.004	0.58 ± 0.05
RDF2Vec[17]	*RF*	0.17 ± 0.004	0.05 ± 0.0001	0.41 ± 0.13
	LR	0.25 ± 0.02	0.07 ± 0.006	0.49 ± 0.12
	MLP	0.23 ± 0.02	0.09 ± 0.008	0.51 ± 0.18
	XG	0.23 ± 0.02	0.07 ± 0.006	0.42 ± 0.10
TransE[18]	*RF*	0.16 ± 0.001	0.05 ± 0.0001	0.52 ± 0.09
	LR	0.17 ± 0.002	0.05 ± 0.001	0.56 ± 0.06
	MLP	0.24 ± 0.006	0.08 ± 0.001	0.56 ± 0.06
	XG	0.24 ± 0.005	0.06 ± 0.001	0.51 ± 0.06
DistMult[19]	*RF*	0.22 ± 0.01	0.05 ± 0.0001	0.50 ± 0.06
	LR	0.22 ± 0.005	0.05 ± 0.001	0.53 ± 0.06
	MLP	0.26 ± 0.005	0.09 ± 0.004	0.54 ± 0.04
	XG	0.37 ± 0.04	0.09 ± 0.002	0.58 ± 0.05
SimplE[20]	*RF*	0.22 ± 0.01	0.05 ± 0.0001	0.45 ± 0.09
	LR	0.23 ± 0.003	0.05 ± 0.001	0.48 ± 0.04
	MLP	0.28 ± 0.004	0.09 ± 0.004	0.53 ± 0.04
	XG	0.36 ± 0.03	0.08 ± 0.003	0.50 ± 0.07
Literal2Feature	RF	0.37 ± 0.007	0.10 ± 0.01	0.57 ± 0.07
	LR	0.41 ± 0.002	0.12 ± 0.005	**0.62** ± **0.08**
	MLP	**0.46** ± **0.006**	**0.21** ± **0.005**	0.48 ± 0.07
	XG	0.38 ± 0.18	0.19 ± 0.07	0.53 ± 0.04

cover a range from decision trees to neural networks. As the accident dataset has imbalanced labels (see Figures 5,6), only F1-measure (macro) is reported. The results are summarised in Table 2. These experiments have been conducted on a single node with a Intel Core i5 CPU and 8GB of RAM. For FeGeLOD, its original default configurations have been preserved, and for KGEs the dimension of vectors has been set to 200. Moreover, all algorithms have been trained using 5-fold cross-validation.

The focus of this experiment is to show the quality of the extracted features and compare it with other approaches. As can be seen, in all cases the extracted features from our proposed framework yield a higher F1 score. For example, in Accident Scenario 1, Literal2Feature achieves 0.46 in comparison to 0.37 for DistMult. The same behavior can be observed in Accident Scenario 2. Here, Literal2Feature achieves 0.21, however, the F1 of all other baselines is less than 0.09. In the Carcinogenesis binary classification scenario, still, Literal2Feature outperforms other methods. Although in this case,

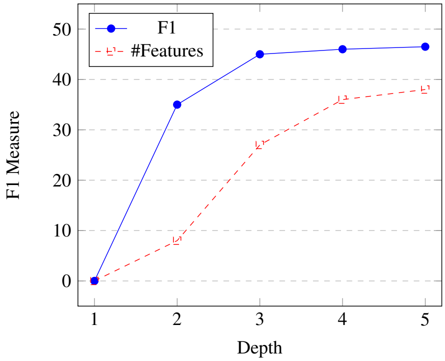


Figure 7. Depth Impact on the Classification Result and Number of Extracted Features

FeGeLOD results are comparable, FeGeLOD achieved 0.61 with 247 extracted features, however, Literal2Feature achieved 0.62 using only 8 extracted features. This reveals that the features extracted by Literal2Feature are more informative and useful.

As a hypothesis, the further we go deeper in the graph, the more likely to find features, but the less trust-worthy and less-relevant the features become. To prove it, we run the classification scenario on the Engie dataset with 4 classes (Accident Scenario 1) with MLP algorithm for the features in different depths. As shown in Figure 7, there is no significant change in F1 after step 3. It indicates that considering the features uptil the depth of 3 is sufficient and there is no need to consider farther features as it can increase the running time of the system without apparent gain.

5.1.2. Clustering

To evaluate the usefulness of the extracted features, we additionally designed a clustering scenario. However, as our datasets have no ground truth for the clustering scenario, we have used Silhouette Coefficient to measure the quality of the clustering results. The Silhouette Coefficient is defined for each sample and is composed of two scores: *a:* The mean distance between a sample and all other points in the same class. *b:* The mean distance between a sample and all other points in the next nearest cluster. The Silhouette Coefficient$s$ for a single sample is then given as:

$$s = \frac{b - a}{max(a, b)} \tag{1}$$

The Silhouette Coefficient for a set of samples is given as the mean of the Silhouette Coefficient for each sample, and a higher Silhouette Coefficient score relates to a model with better defined clusters. We have selected K-Means to demonstrate clustering results and applied elbow-method to determine the optimal number of clusters, which is set to 4. Table 3 shows the result of the clustering.

As the result depicts, Literal2Feature achieved better clustering for the Engie dataset and obtained comparable results to RDF2Vec for the Carcinogenesis dataset, whereas TransE achieved the best clustering score. The reason why TransE achieves good results in clustering is that TransE is a geometric model and is trained using an objective function to keep the similar entities close.

Table 3. Silhouette Coefficient

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

Table 4. Synthetic Dataset Description

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

5.2. *Experiment B: Scalability*

In this experiment we evaluate the scalability of Literal2Feature by using different data sizes and varying cluster computing setups. To be able to have different sizes of datasets, we implemented an RDF data simulator which generates synthetic RDF graph based on the given depth, branching factor, and number of seeds. Table 4 lists the generated datasets and their characteristics. The branching factor is set to 50, depth to 3, and all literals lie at the leaf node to form a complete tree. Worth to mention that the German DBpedia size is 48GB with 336 M triples, however, as the branching factors of nodes are not fixed and equal in real datasets, we decided to do the experiments on synthetic data which requires more CPU and Memory consumption.

5.2.1. *Scalability over number of cores*

To adjust the distributed processing power, the number of available cores was regulated. In this experiment, *DS 4* is selected as a pilot dataset and the number of cores was increased starting from $2^2 = 4$ up to $2^7 = 128$. The experiments were carried out on a small cluster of 4 nodes (1 master, 3 workers): AMD Opteron(tm) CPU Processor 6376 @ 2300MHz (64 Cores), 256 GB RAM. Moreover, the machines were connected via a Gigabit network. All experiments are executed three times and the average value is reported in the results. Figure 8 shows the scalability over different computing cluster setups. It is clear that increasing the computational power horizontally, decreases the execution time. It should be noted that the runtime does not include the time for data ingestion from the Hadoop file system and SPARQL query execution as we used SANSA's internal SPARQL engine for SPARQL execution.

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

5.2.2. *Scalability over dataset size*

To analyze the scalability over different datasets, we fix the computational power to 64 cores and run the experiments for all datasets introduced in Table 4. By comparing the runtime as shown in Figure 9, we note that the execution time does not increase exponentially. Hence, doubling the size of the dataset does not necessarily increase the execution time with the factor of 2. This behaviour is due to the available resources (memory) and partition size. On the other hand, as expected, it can be noted that the random walk consumes much lesser time as compared to full search, which requires comparatively more resources.

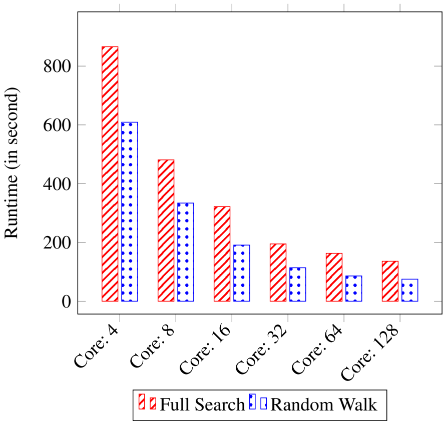


Figure 8. Processing Power Scalability on DS 4 Dataset

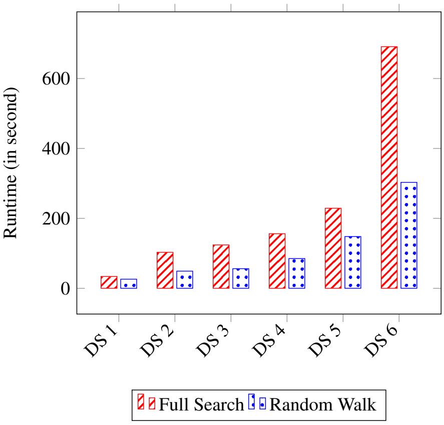


Figure 9. Sizeup performance evaluation over 64 Cores

6. Conclusion

In this paper, we introduce Literal2Feature, a generic distributed framework for transforming RDF data into a feature matrix, which can be used in many machine learning algorithms. By providing full control over different hyper-parameters, users will have a substantial level of flexibility in using the framework. Our experiments also showed that the framework can be used to analyze RDF data with existing statistical machine learning pipelines. Moreover, our experiments show that Literal2Feature can be successfully scaled over a cluster of nodes for large amount of data. In future, we plan to exploit more features such as graph topological structure, e.g. number of neighbors, type-related features, etc.. For multiple features with the same property path, we aim to test the application of aggregator functions like average or max, min, as discussed in Section 3.1.5.

Acknowledgment

This work was partly supported by the EU Horizon 2020 project PLATOON (Grant agreement ID: 872592)

References

[1] T. Berners-Lee, A roadmap to the Semantic Web, 1998.

[2] M. Schmachtenberg, C. Bizer and H. Paulheim, Adoption of the Linked Data Best Practices in Different Topical Domains, in: *The Semantic Web – ISWC 2014*, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz and C. Goble, eds, Springer International Publishing, Cham, 2014, pp. 245–260. ISBN ISBN 978-3-319-11964-9.

[3] C. Bizer, M.-E. Vidal and H. Skaf-Molli, *Linked Open Data*, in: *Encyclopedia of Database Systems*, Springer New York, New York, NY, 2018, pp. 2096–2101. ISBN ISBN 978-1-4614-8265-9.

[4] W.Y. Wang, K. Mazaitis and W.W. Cohen, Structure Learning via Parameter Learning., in: *CIKM*, ACM, 2014, pp. 1199–1208. ISBN ISBN 978-1-4503-2598-1.

[5] L. Galárraga, C. Teflioudi, K. Hose and F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE+., *VLDB J.* **24**(6) (2015), 707–730.

[6] M. Nickel, K. Murphy, V. Tresp and E. Gabrilovich, A Review of Relational Machine Learning for Knowledge Graphs, *Proceedings of the IEEE* **104**(1) (2016), 11–33. doi:10.1109/JPROC.2015.2483592.

[7] L. Bühmann, J. Lehmann and P. Westphal, DL-Learner - A framework for inductive learning on the Semantic Web., *J. Web Semant.* **39** (2016), 15–24.

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

[9] J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, I. Ermilov, S. Bin, N. Chakraborty, M. Saleem, A.-C.N. Ngonga and H. Jabeen, Distributed Semantic Analytics using the SANSA Stack, in: *Proceedings of 16th International Semantic Web Conference - Resources Track (ISWC'2017)*, Springer, 2017, pp. 147–155.

[10] Literal2Feature ScalaDoc. https://sansa-stack.github.io/SANSA-Stack/scaladocs/0.8.0/net/sansa_stack/ml/spark/utils/FeatureExtractingSparqlGenerator$.html.

[11] Literal2Feature Tutorial. https://github.com/SANSA-Stack/SANSA-Stack/blob/develop/sansa-ml/README.md.

[12] V.N.P. Kappara, R. Ichise and O.P. Vyas, LiDDM: A Data Mining System for Linked Data, in: *WWW2011 Workshop on Linked Data on the Web, Hyderabad, India, March 29, 2011*, CEUR Workshop Proceedings, Vol. 813, CEUR-WS.org, 2011. http://ceur-ws.org/Vol-813/ldow2011-paper07.pdf.

[13] W. Cheng, G. Kasneci, T. Graepel, D.H. Stern and R. Herbrich, Automated feature generation from structured knowledge, in: *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, ACM, 2011, pp. 1395–1404. doi:10.1145/2063576.2063779.

[14] M.A. Khan, G.A. Grimnes and A. Dengel, Two pre-processing operators for improved learning from semanticweb data, in: *First RapidMiner Community Meeting And Conference (RCOMM 2010)*, 2010.

[15] H. Paulheim and J. Fürnkranz, Unsupervised generation of data mining features from linked open data, in: *2nd International Conference on Web Intelligence, Mining and Semantics, WIMS '12, Craiova, Romania, June 6-8, 2012*, ACM, 2012, pp. 31:1–31:12. doi:10.1145/2254129.2254168.

[16] P. Ristoski, C. Bizer and H. Paulheim, Mining the Web of Linked Data with RapidMiner, *J. Web Semant.* **35** (2015), 142–151. doi:10.1016/j.websem.2015.06.004.

[17] P. Ristoski and H. Paulheim, RDF2Vec: RDF Graph Embeddings for Data Mining, in: *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, Lecture Notes in Computer Science, Vol. 9981, 2016, pp. 498–514. doi:10.1007/978-3-319-46523-4_30.

[18] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston and O. Yakhnenko, Translating Embeddings for Modeling Multi-Relational Data, in: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, Curran Associates Inc., Red Hook, NY, USA, 2013, pp. 2787–2795–.

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

[20] S.M. Kazemi and D. Poole, SimplE Embedding for Link Prediction in Knowledge Graphs, 2018.

[21] W.W. Cohen, TensorLog: A Differentiable Deductive Database, *CoRR* **abs/1605.06523** (2016).

[22] L. Galárraga, C. Teflioudi, K. Hose and F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE+, *VLDB J.* **24**(6) (2015), 707–730. doi:10.1007/s00778-015-0394-1.

[23] F.Z. Smaili, X. Gao and R. Hoehndorf, Onto2Vec: joint vector-based representation of biological entities and their ontology-based annotations, *Bioinformatics* **34**(13) (2018), i52–i60. doi:10.1093/bioinformatics/bty259.

[24] C. Stadler, G. Sejdiu, D. Graux and J. Lehmann, Sparklify: A Scalable Software Component for Efficient Evaluation of SPARQL Queries over Distributed RDF Datasets, in: *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II*, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I.F. Cruz, A. Hogan, J. Song, M. Lefrançois and F. Gandon, eds, Lecture Notes in Computer Science, Vol. 11779, Springer, 2019, pp. 293–308. doi:10.1007/978-3-030-30796-7_19.

[25] E.F. Moore, *The Shortest Path Through a Maze*, Bell Telephone System. Technical publications. monograph, Bell Telephone System., 1959.

[26] 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**(2) (2020), 95–107. doi:10.1109/TETCI.2019.2952908.

[27] P. Westphal, L. Bühmann, S. Bin, H. Jabeen and J. Lehmann, SML-Bench - A benchmarking framework for structured machine learning, *Semantic Web* **10**(2) (2019), 231–245.

[28] T. Chen and C. Guestrin, XGBoost: A Scalable Tree Boosting System, in: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2016, pp. 785–794.