DistRDF2ML - Scalable Distributed In-Memory Machine Learning Pipelines for RDF Knowledge Graphs

Carsten Felix Draschner
carsten.draschner@uni-bonn.de
University of Bonn
Bonn, Germany

Claus Stadler
cstadler@informatik.uni-leipzig.de
University of Leipzig
Leipzig, Germany

Farshad Bakhshandegan Moghaddam
farshad.moghaddam@uni-bonn.de
University of Bonn
Bonn, Germany

Jens Lehmann
jens.lehmannn@cs.uni-bonn.de
jens.lehmannn@iais.fraunhofer.de
University of Bonn / Fraunhofer IAIS
Bonn / Dresden, Germany

Hajira Jabeen
hajira.jabeen@uni-koeln.de
University of Cologne
Cologne, Germany

ABSTRACT

This paper presents DistRDF2ML, the generic, scalable, and distributed framework for creating in-memory data preprocessing pipelines for Spark-based machine learning on RDF knowledge graphs. This framework introduces software modules that transform large-scale RDF data into ML-ready fixed-length numeric feature vectors. The developed modules are optimized to the multimodal nature of knowledge graphs. DistRDF2ML provides aligned software design and usage principles as common data science stacks that offer an easy-to-use package for creating machine learning pipelines. The modules used in the pipeline, the hyper-parameters and the results are exported as a semantic structure that can be used to enrich the original knowledge graph. The semantic representation of metadata and machine learning results offers the advantage of increasing the machine learning pipelines' reusability, explainability, and reproducibility. The entire framework of DistRDF2ML is open source, integrated into the holistic SANSA stack, documented in scala-docs, and covered by unit tests. DistRDF2ML demonstrates its scalable design across different processing power configurations and (hyper-)parameter setups within various experiments. The framework brings the three worlds of knowledge graph engineers, distributed computation developers, and data scientists closer together and offers all of them the creation of explainable ML pipelines using a few lines of code.

CCS CONCEPTS

• **Computer systems organization** → *Distributed architectures*; • **Software and its engineering** → Software libraries and repositories; • **Computing methodologies** → **Semantic networks**; *Artificial intelligence*; **Machine learning approaches**; • **Information systems** → *Extraction, transformation and loading*.

KEYWORDS

Machine Learning, Knowledge Graphs, Distributed Computing, Explainable AI, Scalable Semantic Processing, RDF, Preprocessing

ACM Reference Format:
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. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3481999

1 INTRODUCTION

In recent years, an increasing variety of linked open data and RDF knowledge graphs have emerged (wiki data [26], YAGO [24], DBpedia [13] and more generally the LOD cloud[1]). These are exciting data representations for various applications and enable data integration through semantic web[1] standards via IRIs [19]. The data sizes of real-world linked open knowledge graphs are often infeasible to process in main memory of a single computer, as the data volume often exceeds the available main memory resources by far. A single computer cannot be arbitrarily scaled in performance regarding the number of processor cores and the amount of main memory. Additionally, specialized components with higher performance become more expensive as they are not in demand by the common consumer market, resulting in disproportionate costs. Cluster computation can solve this problem in terms of available main memory and required CPU processing power by combining the performance of multiple servers as nodes within a cluster. Many distributed computing frameworks have been developed in recent years and are in productive use, like Apache Spark [8, 15] and Apache Flink [6]. In the area of machine learning, pipelines, frameworks, and libraries that realize both modular pipeline creations and explainable AI have become very popular. These libraries and frameworks are open source and available to interested parties

[1]https://lod-cloud.net

through web-hosted examples. Creating pipelines based on generic transformers (scikit-learn [21], Spark MLlib [15]) opens up the possibility for a wide range of software developers, data scientists, and knowledge engineers to develop machine learning models for their data. While the usual machine learning models implemented in Spark MLlib (BigDL [4], Analytics Zoo[2]) assume fixed length numeric feature vectors, a holistic framework for developing distributed machine learning pipelines is missing for handling large scale RDF knowledge graphs. Such a framework can empower the already existing Apache Spark clusters to operate natively on RDF knowledge graph data. SANSA (Semantic Analytics Stack) [14] uses the scalable processing capability of Apache Spark to allow distributed RDF processing. However, it did not offer a pipeline for ML solutions. Knowledge graphs do not provide a native representation that fits these requirements. Therefore, there are two possibilities: First, latent embeddings can be computed or learned (TransE [2], DistMult [27], RDF2Vec [22]). Alternatively, relevant features can be extracted and digitized. Targeted feature extraction can be implemented in a distributed fashion and has the advantage of being able to generate explainable feature vectors. Explainability means that the individual indices of a feature vector can be assigned to the original features. This possibility is significant for several applications and institutions developing data analytics and machine learning pipelines. These initial explainable ML pipelines also provide an essential comparative reference to evaluate the capabilities of less explainable latent embedding driven ML pipelines. Also, 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 should be avoided in many use cases, especially those that use RDF for data integration and use such features. Especially when a special requirement of a machine learning pipeline is its explainability, disentangled feature vectors should be used.

In conclusion, in this paper, we propose a framework that enables the intuitive creation of RDF knowledge graph-based machine learning pipelines in multiple areas. Our framework, which is natively executable on Apache Spark clusters, enables many machine learning applications to scale with data size and computation performance.

The contributions of this work are as follows:

- An open source framework for distributed machine learning pipelines on RDF knowledge graphs
- Generic modules for the creation of explainable and context preserving feature vectors
- Various sample machine learning pipelines documented and testable within Databricks[3] notebooks, executable sample classes, and unit tests
- A software architecture which its semantic data machine learning pipelines is aligned with pipelines in established data analytics libraries
- Reproducibility over semantic annotation of pipeline metadata and processed results

[2]https://analytics-zoo.readthedocs.io/en/latest/
[3]https://databricks.com/

The rest of the paper is organised as follows: Section 2 defines technological and conceptual terms used within this paper. Section 3 gives an overview of Related Work. In section 4 the DistRDF2ML architecture is presented as well as the concept of modules. Section 5 presents the framework as a resource and especially targets its Novelty (Sect. 5.1), Availability (Sect. 5.2), Utility (Sect. 5.3) and Predicted Impact (Sect. 5.4). Section 6 and Section 7 evaluated the performance and scalability of the framework among multiple (hyper-)parameter while Section 8 concludes the paper and highlights a few future directions (see Sect. 8.1).

2 PRELIMINARIES

RDF - Resource Description Framework: is a standard to represent metadata and designed as a core technology for building a web of data with the envisioned goal to realize the Semantic Web [1, 19].

Scala: is a functional and object-oriented programming language. Scala uses static types to reduce bugs in complex applications[4]. It is also possible to use Java libraries as Scala is being executed within the Java Virtual Machine (JVM) [18].

Apache Spark: is a framework for cluster computing [8]. It is available under the Apache Open Source License. Apache Spark encapsulates software modules that optimize the execution of big data analytics pipelines. These pipelines can be executed distributed and in-memory on Spark clusters. Apache Spark MLlib [9] adapts the ideas of well-known python library scikit-learn[5] to provide standard interfaced transformers that allow high modular and generic machine learning pipeline construction [15]. 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 operated in distributed processing environments.

Apache Jena: is an open source Apache framework [7] programmed in Java to develop Linked Data, RDF Data, and Semantic Web programs.

Machine Learning: (abbreviation ML) is one approach to implement artificial intelligence. The primary approach is to use existing datasets to build training and test datasets. These datasets are fed to algorithms that abstract the pattern and correlations of the given data to a generalized solution.

ML Preprocessing: describes the process of transforming raw data into the required representation the machine learning models operate on. Many standard machine learning models expect for each sample a fixed size numeric feature vector/tensor.

*SANSA - **S**emantic **An**alytics **Sta**ck:* is an open source framework to process large scale RDF data based on Apache Spark, Apache Flink, and Apache Jena, within various tasks like: semantic data representation, querying, inference, and analytics.

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

3 RELATED WORK

Linked open data, especially semantic data in RDF format, is a rich source for baseline datasets[6] [10, 13, 26] to perform data analytic pipelines and machine learning approaches on. Also, in times of big data, frameworks have been developed which can deal with this large-scale data on distributed systems. However, there are no solutions to bring the linked data world together with building distributed explainable machine learning pipelines. On the algorithmic level, diverse approaches transform the knowledge graph entities into fixed-length feature vectors [3, 11, 12, 20]. The fixed-length feature vectors are either retrieved via SPARQL or over latent embedding creation. Latent embeddings can be created by tensor factorization methods such as Distmult [27]. Alternatively, they can be generated over learning feature vectors by optimizing initial random vectors against a loss function (e.g., TransE [2]) and the respective follow-up approaches. Another option for creating latent embeddings is graph walk approaches like RDF2Vec [22]. In general, none of them are available in the Spark world. Within the Apache Spark stack, there are multiple frameworks and libraries which offer standard machine learning models such as Apache Spark MLlib [15] contains models such as Regression[7], Clustering[8], Support Vector Machines, and Multi-Layer Perceptrons. More complex are the approaches of BigDL [4] and Analytics Zoo[9] which offer more complex neural network models but are currently not up to date with the most recent version of Apache Spark 3.x. The only approach that provides native RDF knowledge graph-based machine learning approaches within the Scala-based Apache Spark framework is developed within the SANSA framework. However, there are multiple layers for reading, querying, OWL, and similarity assessment [5]. There are no generic transformers in the respective ML layer to handle multi-modal features retrieved over SPARQL queries. Our approach introduces these models and transformers and also fully integrates them into the SANSA stack. These modules make use of the Literal2Feature [17], an automatic SPARQL generation module, which creates a feature fetching SPARQL queries for the users. Advanced users can also update the designed SPARQL queries to desired purposes.

4 DISTRDF2ML ARCHITECTURE

4.1 Pipeline

The proposed DistRDF2ML framework offers the construction of end-to-end Apache Spark 3.x pipelines (see Figures 1, 2, 3). The principal pipeline contains the following modules:

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

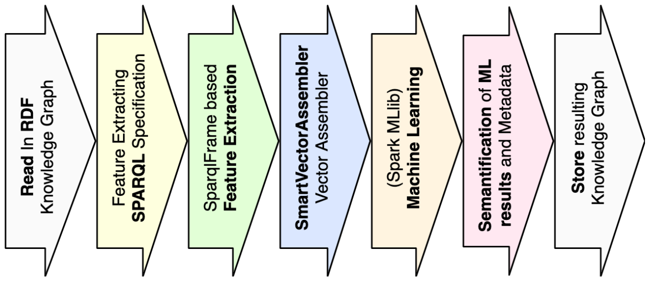
[6]https://lod-cloud.net

[7]http://spark.apache.org/docs/latest/ml-classification-regression.html

[8]http://spark.apache.org/docs/latest/ml-clustering.html

[9]https://analytics-zoo.github.io/master/

Read in knowledge graph: read in of data is performed over the SANSA framework within the RDF layer[10]. This layer supports multiple RDF formats and can read from Hadoop File System (HDFS[11]) to incorporate large-scale RDF data. Besides *n-triples*, SANSA also supports concurrent ingestion of the *turtle* and *trig* RDF formats.



Read in RDF Knowledge Graph
Feature Extracting SPARQL Specification
SparqlFrame based Feature Extraction
SmartVectorAssembler Vector Assembler
(Spark MLlib) Machine Learning
Semantification of ML results and Metadata
Store resulting Knowledge Graph

Figure 1: DistRDF2ML Pipeline Overview

4.2 SparqlFrame

SPARQL creation: In pipelines constructed with DistRDF2ML (see Figure 2), we offer three variants to create a SPARQL query that extracts relevant features. The first option is that a knowledge graph expert manually crafts the desired SPARQL query. The second option, is to let the Literal2Feature [17] 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 which is then 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 1. The projection variable names are automatically generated and offer insights about how the respective feature is reached within the RDF knowledge graph.

```
SELECT
?movie
?movie__down_genre__down_film_genre_name
?movie__down_title
?movie__down_runtime
?movie__down_actor__down_actor_name
WHERE {
?movie <http://www.w3.org/1999/02/22-rdf-syntax-ns#
        type> <http://data.linkedmdb.org/movie/film> .
OPTIONAL { ?movie <http://purl.org/dc/terms/title> ?
        movie__down_title .}
OPTIONAL { ?movie <http://data.linkedmdb.org/movie/
        runtime> ?movie__down_runtime .}
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 .}
```

[10]https://github.com/SANSA-Stack/SANSA-Stack/tree/develop/sansa-rdf

[11]https://hadoop.apache.org/docs/

```
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 1: Sample SPARQL query from hybrid usage of Literal2Feature and manual edit

SPARQL based feature extraction: SparqlFrame is a high-level transformer that allows for easy and intuitive execution of SPARQL queries on RDF knowledge graphs in order to create the Spark DataFrames that are suitable input to conventional ML pipelines. SparqlFrame bundles SANSA query execution engines (Sparqlify[23] and Ontop[12]) together with a novel schema mapper that was created as part of this work. While 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[13] to appropriate Spark 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[14] StringType, DoubleType, and TimeStampType. If a feature has multiple annotated datatypes, a separate column is created for each data type, and this is mapped in the column name. For every unknown RDF literal datatype, a variable is bound to a separate column and will be allocated as usual, however the column type will fall back to StringType, and it will receive the lexical forms of those literals.

```
val dataset: Dataset[Triple] = [...]
val sparqlString: String = [...]
val sparqlFrame = new SparqlFrame()
      .setSparqlQuery(sparqlString)
      .setCollapsByKey(true)
      .setCollapsColumnName("movie")
val extractedFeaturesDf = sparqlFrame
      .transform(dataset)
// semantic representation of transformer
sparqlFrame
      .getSemanticTransformerDescription()
```

Listing 2: Sample usage of SparqlFrame transformer

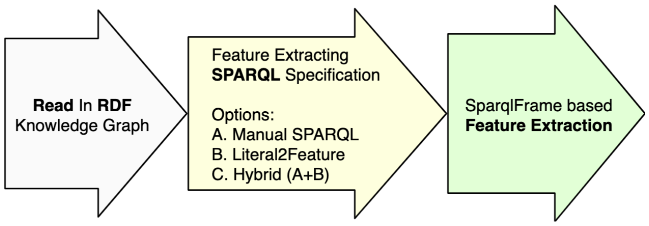

Read In RDF Knowledge Graph
Feature Extracting SPARQL Specification
Options:
A. Manual SPARQL
B. Literal2Feature
C. Hybrid (A+B)
SparqlFrame based Feature Extraction

Figure 2: DistRDF2ML SPARQL specification options

[12]https://ontop-vkg.org
[13]https://www.w3.org/TR/xmlschema11-1/
[14]https://spark.apache.org/docs/latest/sql-ref-datatypes.html

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 [16] 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 that 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.

4.3 SmartVectorAssembler

Digitization and assembling of features: This 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 an input for standard machine learning 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, that 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 by information like *dayOfWeek* offer 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.

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

Listing 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 machine learning 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 *stddev* 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 common machine learning models. However, due to the unique preprocessing pipeline, every single value of the feature vector can be assigned to the original feature (see Listing 3 Line 7). This feature vector indices description enables explainability for machine learning pipelines over knowledge graph embedding-based feature vectors. The resulting Dataframe can be used natively in Apache Spark MLlib [15]. A sample resulting DataFrame from SmartVectorAssembler is shown in Listing 4.

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

Listing 4: Sample output of SmartVectorAssembler

4.4 Machine Learning Models

In the distributed big data processing within Apache Spark, some libraries have been developed, which provide many needed machine learning models which can be executed over Scala and Python and

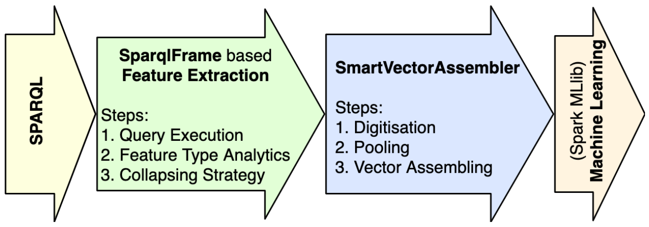


Figure 3: DistRDF2ML feature vector creation pipeline through SparqlFrame and SmartVectorAssembler

executed on an Apache Spark cluster [15]. For the most common downstream machine learning tasks, we can use Apache Spark MLlib[15]. If we need the opportunities to use more complex neural networks than Multi-Layer Perceptrons, the framework BigDL [4] provides further artificial neural network functionalities[16]. Available machine learning models within the Spark MLlib 3.1 are for example: Logistic Regression, Random Forest, Multilayer Perceptron, and Linear Support Vector Machine, which can be used for Classification and Regression tasks[17]. The MLlib machine learning models can directly operate on the output of the DistRDF2ML pipeline (see Examples within [25]). This offers end-to-end machine learning pipelines operating on Apache Spark clusters. Hence, DistRDF2ML serves as an enabler to process large-scale RDF data through downstream ML tasks in a distributed fashion.

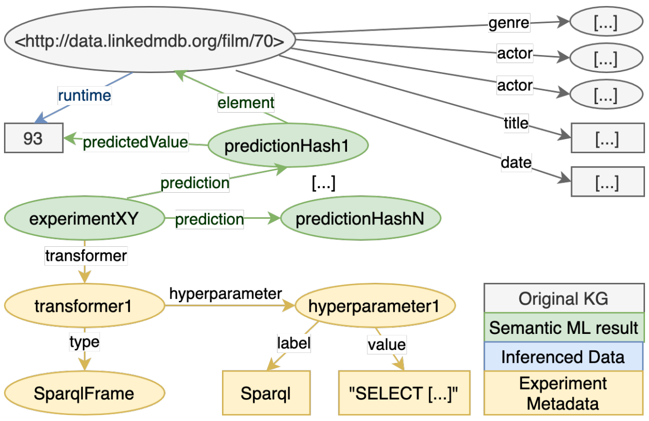


Figure 4: Semantically annotated ML result in original KG

Semantification of Results: The machine learning computations of the DistRDF2ML framework are available in the 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 4) and the corresponding annotation in the prediction field. In the case of similarity estimations, two columns are used for the pair of the URIs and one for the corresponding similarity value. Our ML2Graph module can be used as a transformer. It creates an RDF knowledge graph based on these tabular results

[15]https://spark.apache.org/docs/latest/ml-guide.html
[16]https://bigdl-project.github.io/master/
[17]http://spark.apache.org/docs/latest/ml-classification-regression.html

using *NodeFactory* from *Apache Jena*. The resulting graph can be exported in common RDF data format to complete the initial input knowledge graph with the predictions (see Figure 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.

```
val prediction: Dataframe = someMLmodel
    .transform(df)
val ml2Graph = new ML2Graph()
    .setEntityColumn("entityID")
    .setValueColumn("prediction")
val metagraph: RDD[Triple] = ml2Graph
    .transform(prediction)
metagraph
    .saveAsNTriplesFile("/out/put/path")
```

Listing 5: Semantification of machine learning results

5 DISTRDF2ML AS A RESOURCE

The DistRDF2ML framework proposes modules to preprocess RDF knowledge graph 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 [25] 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 various (hyper-)parameter setups to present the advantages of the efficient data processing pipelines in data-intensive computing (see Section 6). 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 [17] and SparqlFrame modules also enable non-native semantic developers to construct explainable feature extraction pipelines. The feature and knowledge extraction based on queries and processed through the SmartVectorAssembler allow multi-modal content preserving representations of feature vectors. Within various partner projects, the framework shows its enabling nature to create with few lines of intuitive code ML pipelines for various domains.

5.1 Novelty

DistRDF2ML brings for the first time a generic preprocessing pipeline of RDF knowledge graphs 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 knowledge graphs. This framework brings multiple complex technology stacks together, such as semantic data processing, distributed scalable computing, generic machine learning, and data science pipelines.

5.2 Availability

All the introduced building blocks of the DistRDF2ML pipeline are integrated into the SANSA framework within the machine learning layer[18]. The framework is also published as a Release[19] 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).

5.3 Utility

The modules are fully documented on the code level[20] but also on how to use level[21]. 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 [25]. This is shown in Listings 2,3. We made multiple sample pipelines available as an example class or as a Databricks Notebook [25] s.t. 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.

5.4 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 to scale them among Big Data requirements. This resource brings together the strengths of multiple domains and offers the interface between those. 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 which are executed for every pull request within GitHub actions s.t. problems or errors become apparent pretty 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 knowledge graphs. Due to the increase of available and the usage of RDF data in the context of knowledge graphs and data integration, and the increase of data set sizes in general, the availability of a scalable framework offering accessible, hands-on opportunities to port ideas directly within the SANSA framework is an important resource contribution.

5.5 Use Cases

Generic scalable distributed preprocessing pipeline for RDF knowledge graphs are needed in almost all common downstream machine learning and data analytics tasks which have to be operated on explainable feature vectors. The EU Horizon 2020 project PLATOON (Platform for Tools in Energy)[22] 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,

[18]https://github.com/SANSA-Stack/SANSA-Stack/tree/develop/sansa-ml
[19]https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1_DistRDF2ML
[20]https://sansa-stack.github.io/SANSA-Stack/
[21]https://github.com/SANSA-Stack/SANSA-Stack/blob/develop/sansa-ml/README.md
[22]https://platoon-project.eu

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[23] 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[24] 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 the further processing of standard ML models.

6 EVALUATION

6.1 Data Description

For the evaluation and reproducibility of DistRDF2ML usage, we use the openly available Linked Movie Database[10] which introduces a movie knowledge graph 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 knowledge graph like the NLP of the title, categorical lists of String representing the movie genres. Alternatively, even the countries population, the movie's runtime as digit features can be incorporated into data analytics and ML pipelines. Figure 5 shows a small extract from the knowledge graph as a plot (extract from N-Triple file are available in the release data set section [25]).

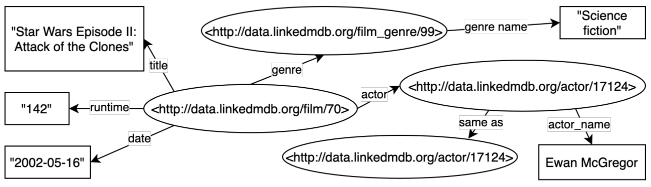

"Star Wars Episode II: Attack of the Clones"
<http://data.linkedmdb.org/film_genre/99>
genre name
"Science fiction"
title
genre
actor
<http://data.linkedmdb.org/actor/17124>
"142"
runtime
<http://data.linkedmdb.org/film/70>
same as
actor_name
date
"2002-05-16"
<http://data.linkedmdb.org/actor/17124>
Ewan McGregor

Figure 5: Sample graph snippet from Linked Movie Database

6.2 Description Evaluation Dimensions

A major aspect of this contribution is the usage of Apache Spark to allow distributed execution of machine learning pipelines starting from semantic data. This offers scalability in memory and processing power, which would not be available in single system implementations. Also, processing power, 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, 7, 8, 9 present the processing times of the most complex software modules as stacked bar charts. The dark green lower part of each full bar corresponds to the Sparql-Frame transformer, and the light blue upper part corresponds to the SmartVectorAssembler. The height of the bars always corresponds to the execution time in seconds.

[23]https://simple-ml.de
[24]https://www.opertusmundi.eu

6.3 Processing Power vs Processing Time

The following paragraph shows how the available computational resources majorly influence the processing time. The substantial decrease of processing time over an increase of processing power allows high scalability among higher-performing clusters. For use cases where the processing time is critical, it is even possible to reduce the processing time by increasing the number of executing cores, showing the high effective parallelism and distributed approach of DistRDF2ML.

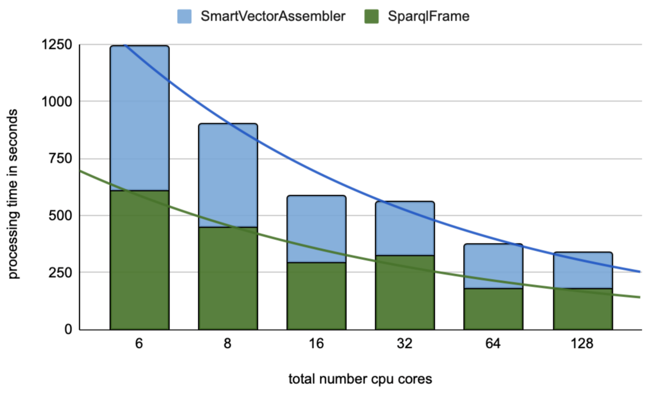


Figure 6: Processing power vs processing time

6.4 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 respective feature level and the type of used feature. In our example of linked movie database where a movie has assigned an arbitrarily long feature list of *actors' names*, but also a single digit feature of the movie's *runtime*, the feature processing steps for a variable length string feature like the actors is more memory-intensive than the single digits representation of the *runtime*. We also compared the local execution of this pipeline compared 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 there available fully open source to allow a higher reproducibility of the experiments and find respective recommendations for other use cases [25].

6.5 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

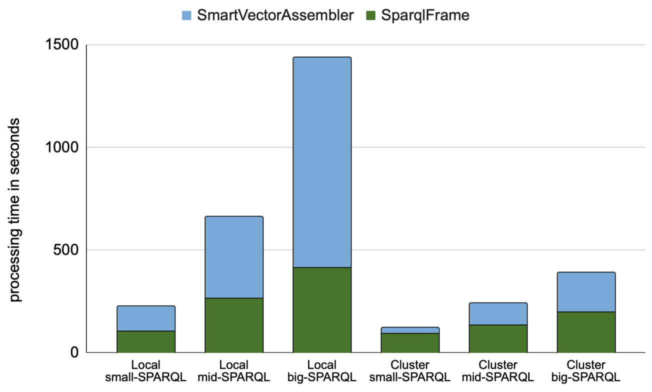


Figure 7: SPARQL complexity vs processing time

achieved with consistent data and feature density. The evaluation shows that DistRDF2ML scales even with exponentially increasing data sizes. Also, it can be seen that for small data up to 10^3 movies, a local execution is faster than a distributed execution. This behaviour is because the data size does not justify the overhead of network communication and distributed execution management. However, it can be seen that 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 prevent out of memory problems (see figure 8, see section 1). 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 in full 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.

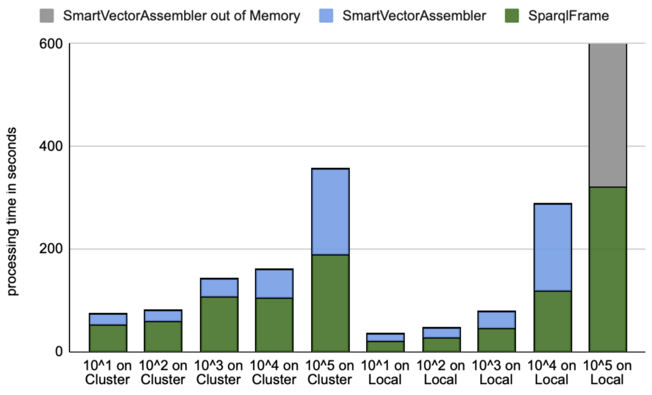


Figure 8: Dataset (-size) and number movies on cluster and local execution vs processing time

6.6 Spark Setup vs Processing time

In the Spark setup, we can configure over the spark-submit a distribution of executors and their assigned executor cores and executor memories. In a cluster of nodes, each having 64 cores and 256GB RAM, we can decide to 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 9 depicts the optimization of execution time by balancing between fat and micro executors (best configuration in 4th bar: 15 executors each having 12 executor cores).

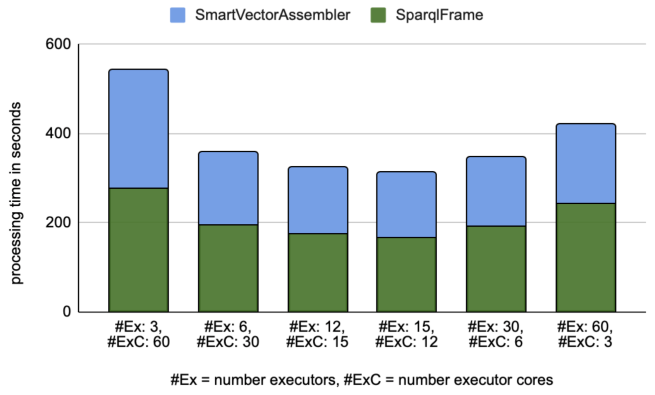


Figure 9: Spark setup vs Processing time

6.7 Sample DistRDF2ML pipelines

For demonstration purposes, we developed machine learning pipelines on RDF knowledge graphs and made those available within the SANSA stack GitHub repository and over Databricks notebooks[25]. Within the documentation, one can find:

- 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

7 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 out of the available resources. In Figure 7 we have shown the first-hand-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 CPU-processing wise the cluster scales far better than local execution. Also very important, the cluster can easily be configured with more RAM and can more easily 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 behaviour among various (hyper-)parameters (see section 6).

Generality of Approach: Within paragraph 6.7, DistRDF2ML shows its various application opportunities as enabler and glue between large scale RDF knowledge graphs and existing numeric feature-based Apache Spark MLlib machine learning 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 cause of its fully open source nature.

8 CONCLUSION

DistRDF2ML provides the first end-to-end Apache Spark framework of generic modules to create explainable machine learning pipelines for knowledge graphs. 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 machine learning models. By incorporating modules of the SANSA stack, Apache Spark and Apache Jena, DistRDF2ML operates natively on large-scale RDF data sets. 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, it can be observed that by increasing resources (number of CPU cores, see Figure 6), the processing time is reduced. Also, a distributed environment with the increased allocation of memory can be perceived as capable of significantly processing more extensive data sets (see Figure 8). The variety of documentation from Scala docs over example pipelines to the mapped functionalities in the unit tests allows a quick start in using the framework [25]. The alignment to the standard interfaces of Spark MLlib and transformer (see Listings: 2, 3) also makes working with large-scale RDF data possible for a large number of data scientists who have less experience with working on semantic, linked open data, or Apache Spark. The availability of open source also makes necessary customization by users possible.

8.1 Future Work

We will keep track of the DistRDF2ML and SANSA stack user needs and will update desired features corresponding to upcoming requirements. DistRDF2ML will become part of the next regular SANSA release. Also, we plan to extend the pipeline to include the possibility of knowledge graph embeddings. We also plan to make graph native machine learning and linked data analytics approaches part of the framework. We will facilitate the opportunities of DistRDF2ML to develop applied projects which solve prediction and data analytic problems.

ACKNOWLEDGEMENT

This work was partly supported by the EU Horizon 2020 project PLATOON (Grant agreement ID: 872592). We would also like to thank the SANSA development team for their helpful support.

REFERENCES

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The semantic web. *Scientific american* 284, 5 (2001), 34–43.
[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*. NIPS, South Lake Tahoe, 1–9.
[3] Weiwei Cheng, Gjergji Kasneci, Thore Graepel, David H. Stern, and Ralf Herbrich. 2011. 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, 1395–1404. https://doi.org/10.1145/2063576.2063779
[4] Jason Jinquan Dai, Yiheng Wang, Xin Qiu, Ding Ding, Yao Zhang, Yanzhang Wang, Xianyan Jia, Cherry Li Zhang, Yan Wan, Zhichao Li, et al. 2019. Bigdl: A distributed deep learning framework for big data. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, Santa Cruz CA USA, 50–60.
[5] Carsten Felix Draschner, Jens Lehmann, and Hajira Jabeen. 2021. DistSim-Scalable Distributed in-Memory Semantic Similarity Estimation for RDF Knowledge Graphs. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*. IEEE, Laguna Hills, California, 333–336.
[6] Apache Flink Foundation. 2021. Apache Flink. https://flink.apache.org.
[7] Apache Jena Foundation. 2021. Apache Jena. https://jena.apache.org/index.html.
[8] Apache Spark Foundation. 2021. Apache Spark. https://spark.apache.org.
[9] Apache Spark Foundation. 2021. Apache Spark MLlib. https://spark.apache.org/mllib/.
[10] Oktie Hassanzadeh and Mariano P Consens. 2009. *Linked Movie Data Base*. openreview.net, no adress.
[11] Venkata Narasimha Pavan Kappara, Ryutaro Ichise, and O. P. Vyas. 2011. 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. http://ceur-ws.org/Vol-813/ldow2011-paper07.pdf
[12] M.A. Khan, G.A Grimnes, and A. Dengel. 2010. Two pre-processing operators for improved learning from semanticweb data. In *First RapidMiner Community Meeting And Conference (RCOMM 2010)*.
[13] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. 2015. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web* 6, 2 (2015), 167–195.
[14] Jens Lehmann, Gezim Sejdiu, Lorenz Bühmann, Patrick Westphal, Claus Stadler, Ivan Ermilov, Simon Bin, Nilesh Chakraborty, Muhammad Saleem, Axel Cyrille Ngonga Ngomo, and Hajira Jabeen. 2017. 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, iii (2017), 147–155. https://doi.org/10.1007/978-3-319-68204-4_15
[15] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.
[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[17] Farshad Bakhshandegan Moghaddam, Carsten Felix Draschner, Jens Lehmann, and Hajira Jabeen. 2021. Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor. In *Proceedings of the 17th International Conference on Semantic Systems, SEMANTICS 2021, Amsterdam, The Netherlands, September 6-9, 2021*.
[18] Martin Odersky, Lex Spoon, and Bill Venners. 2008. *Programming in scala*. Artima Inc.
[19] Jeff Z Pan. 2009. Resource description framework. In *Handbook on ontologies*. Springer, 71–90.
[20] Heiko Paulheim and Johannes Fürnkranz. 2012. 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, 31:1–31:12. https://doi.org/10.1145/2254129.2254168

[21] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[22] Petar Ristoski and Heiko Paulheim. 2016. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*. Springer, Kobe, Japan, 498–514.

[23] Claus Stadler, Gezim Sejdiu, Damien Graux, and Jens Lehmann. 2019. Sparklify: A Scalable Software Component for Efficient Evaluation of SPARQL Queries over Distributed RDF Datasets. In *The Semantic Web – ISWC 2019*, Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtěch Svátek, Isabel Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon (Eds.). Springer International Publishing, Cham, 293–308.

[24] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*. 697–706.

[25] SANSA team. 2021. DistRDF2ML Release. https://github.com/SANSA-Stack/SANSA-Stack/releases/tag/v0.8.1_DistRDF2ML.

[26] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[27] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* arXiv:1412.6575 (2014), 1–12.