Machine Learning with and for Semantic Web Knowledge Graphs

Heiko Paulheim

Data and Web Science Group, University of Mannheim, Germany
heiko@informatik.uni-mannheim.de

Abstract. Large-scale cross-domain knowledge graphs, such as DBpedia or Wikidata, are some of the most popular and widely used datasets of the Semantic Web. In this paper, we introduce some of the most popular knowledge graphs on the Semantic Web. We discuss how machine learning is used to improve those knowledge graphs, and how they can be exploited as background knowledge in popular machine learning tasks, such as recommender systems.

Keywords: Knowledge Graphs, Semantic Web, Machine Learning, Background Knowledge

1 Introduction

The term "Knowledge Graph" was coined by Google when they introduced their knowledge graph as a backbone of a new Web search strategy in 2012, i.e., moving from pure text processing to a more symbolic representation of knowledge, using the slogan "things, not strings"[1].

A similar idea, albeit already introduced in the mid-2000s, underlies the concept of *Linked Data*: in order to organize knowledge, URIs (instead of textual names) are used to identify and distinguish entities [1]. Hence, many datasets of the Linked Open Data cloud [73] could also be considered knowledge graphs.

There is no formal definition of a knowledge graph [12]. In the course of this work, we follow the characteristics sketched in [47], saying that a knowledge graph

1. mainly describes real world entities and their interrelations, organized in a graph.
2. defines classes and properties of entities in a schema.
3. allows for potentially interrelating arbitrary entities with each other.
4. covers various topical domains.

We call a dataset following those characteristics and published using Semantic Web standards a *Semantic Web Knowledge Graph*. As Semantic Web standards, we understand

[1] https://googleblog.blogspot.de/2012/05/introducing-knowledge-graph-things-not.html

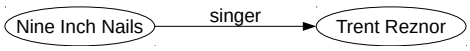


Fig. 1: Example RDF Statement

- the use of *derefencable URIs* [1] for referring to entities, i.e., URIs that point to resources on the Web
- the use of RDF[2] for representing the graph, and
- the use of RDF schema[3] and/or OWL[4] for representing the schema of the graph

RDF organizes knowledge in *statements*, connecting either two entities in a knowledge graph by an edge, or an entity with a literal (i.e., elementary) value (such as a number or a date). Fig. 1 shows an example of such an RDF statement. It can also be written down as a *triple* consisting of a subject, a predicate, and an object, where the subject and the object are the entities, whereas the predicate is the property or edge label:

```
:Nine_Inch_Nails :singer :Trent_Reznor .
```

Hence, an RDF knowledge graph can either be conceived as a directed, labeled graph, or as a set of such triples.

As an alternative to the triple notation, such statements can be expressed in terms of binary predicates, e.g.,

```
singer(Nine_Inch_Nails, Trent_Reznor)
```

In the course of this paper, we will use the following terms to refer to concepts related to knowledge graphs:

entities* or *instances are the nodes in a graph. Typically, they refer to an entity in the real world, such as a person, a city, etc.

literals are elementary data values, such as numbers or dates. They can be used, e.g., for expressing the birth date of a person or the population of a city.

relations are the edges in a graph. They link two entities or an entity and a literal.

Those concepts are typically used in the *A-box*, i.e., the assertional part of the knowledge graph. This is typically the larger part of a knowledge graph. It is complemented by the schema, or *T-box*, which defines the types of entities and relations that can be used in a knowledge graph. Those encompass:

classes* or *types are the categories of entities that exist in a knowledge graph, e.g., *Person*, *City*, etc. They can form a hierarchy, e.g., *City* being a subclass of *Place*.

[2] `https://www.w3.org/RDF/`
[3] `https://www.w3.org/TR/rdf-schema/`
[4] `https://www.w3.org/TR/owl-overview/`

properties are the categories of relations that exist in a knowledge graph, e.g., *birth date*, *birth place*, etc.

While the set of classes is defined in the T-box, the assertion that an individual class is made as a triple in the A-box, e.g.:

```
:Nine_Inch_Nails a :Band .
```

or using a unary predicate:

```
Band(Nine_Inch_Nails)
```

All assertions made in the A-box – relating two entities, relating an entity and a literal, and assigning a type to an entity, are called *facts*.

Often, the schema also defines further constraints, e.g., certain classes may be disjoint, and properties may have a *domain* (i.e., all subjects of the relation have a certain type) and a *range* (i.e., all objects of the relation have a certain type). For example, the relation *birth Place* may have the domain `Person` and the range `City`. Moreover, properties may be defined with other characteristics, such as symmetry, transitivity, etc.

2 Semantic Web Knowledge Graphs

Various public knowledge graphs are available on the Web, including DBpedia [30] and YAGO [33], both of which are created by extracting information from Wikipedia (the latter exploiting WordNet on top), the community edited Wikidata [79], which imports other datasets, e.g., from national libraries[5], as well as from the discontinued Freebase [58], the expert curated OpenCyc [32], and NELL [5], which exploits pattern-based knowledge extraction from a large Web corpus. Lately, new knowledge graphs have been introduced, including DBkWik, which transfers the DBpedia approach to a multitude of Wikis [26], and WebIsALOD, which extracts a knowledge graph of hypernymy relations from the Web [24, 74] using Hearst patterns [20] on a large-scale Web corpus.

2.1 Cyc and OpenCyc

The *Cyc* knowledge graph is one of the oldest knowledge graphs, dating back to the 1980s [32]. Rooted in traditional artificial intelligence research, it is a *curated* knowledge graph, developed and maintained by CyCorp Inc.[6] OpenCyc was a reduced version of Cyc, which used to be publicly available, including a Linked Open Data endpoint with links to DBpedia and other LOD datasets. Despite its wide adoption, OpenCyc was shut down in 2017.[7].

OpenCyc contained roughly 120,000 entities and 2.5 million facts; its schema comprised a class hierarchy of roughly 45,000 classes, and 19,000 properties.[8]

[5] `https://www.wikidata.org/wiki/Wikidata:Data_donation`
[6] `http://www.cyc.com/`
[7] `http://www.cyc.com/opencyc/`
[8] These numbers have been gathered by own inspections of the 2012 of version of OpenCyc.

2.2 Freebase

Curating a universal knowledge graph is an endeavour which is infeasible for most individuals and organizations. To date, more than 900 person years have been invested in the creation of Cyc [72], with gaps still existing. Thus, distributing that effort on as many shoulders as possible through *crowdsourcing* is a way taken by *Freebase*, a public, editable knowledge graph with schema templates for most kinds of possible entities (i.e., persons, cities, movies, etc.). After MetaWeb, the company running Freebase, was acquired by Google, Freebase was shut down on March 31st, 2015.

The last version of Freebase contains roughly 50 million entities and 3 billion facts[9]. Freebase's schema comprises roughly 27,000 classes and 38,000 relation properties.[10]

2.3 Wikidata

Like Freebase, *Wikidata* is a collaboratively edited knowledge graph, operated by the Wikimedia foundation[11] that also hosts the various language editions of Wikipedia. After the shutdown of Freebase, the data contained in Freebase is subsequently moved to Wikidata.[12] A particularity of Wikidata is that for each axiom, provenance metadata can be included – such as the source and date for the population figure of a city [79].

To date, Wikidata contains roughly 16 million entities[13] and 66 million facts[14]. Its schema defines roughly 23,000 classes[15] and 1,600 properties[16].

2.4 DBpedia

DBpedia is a knowledge graph which is extracted from structured data in Wikipedia. The main source for this extraction are the key-value pairs in the Wikipedia infoboxes. In a crowd-sourced process, types of infoboxes are mapped to the DBpedia ontology, and keys used in those infoboxes are mapped to properties in that ontology. Based on those mappings, a knowledge graph can be extracted [30].

The most recent version of the main DBpedia (i.e., DBpedia 2016-10) contains 5.1 million entities and almost 400 million facts.[17] The ontology comprises 754 classes and 2,849 properties.

---

[9] `http://www.freebase.com`
[10] These numbers have been gathered by queries against Freebase's query endpoint.
[11] `http://wikimediafoundation.org/`
[12] `http://plus.google.com/109936836907132434202/posts/3aYFVNf92A1`
[13] `http://www.wikidata.org/wiki/Wikidata:Statistics`
[14] `http://tools.wmflabs.org/wikidata-todo/stats.php`
[15] `http://tools.wmflabs.org/wikidata-exports/miga/?classes\#_cat=Classes`
[16] `http://www.wikidata.org/wiki/Special:ListProperties`
[17] `http://wiki.dbpedia.org/dbpedia-2016-04-statistics`

2.5 YAGO

Like DBpedia, YAGO is also extracted from DBpedia. YAGO builds its classification implicitly from the category system in Wikipedia and the lexical resource *WordNet* [41], with infobox properties manually mapped to a fixed set of attributes. While DBpedia creates different interlinked knowledge graphs for each language edition of Wikipedia [4], YAGO aims at an automatic fusion of knowledge extracted from various Wikipedia language editions, using different heuristics [33].

The latest release of YAGO, i.e., YAGO3, contains 5.1 million entities and 1.5 billion million facts. The schema comprises roughly 488,000 classes and 77 properties [33].

2.6 NELL

While DBpedia and YAGO use semi-structured content as a base, methods for extracting knowledge graphs from unstructured data have been proposed as well. One of the earliest approaches working at web-scale was the *Never Ending Language Learning (NELL)* project [5]. The project works on a large-scale corpus of web sites and exploits a coupled process which learns text patterns corresponding to type and relation assertions, as well as applies them to extract new entities and relations. Reasoning using a light-weight ontology[18] is applied for consistency checking and removing inconsistent axioms. The system is still running today, continuously extending its knowledge base. While not published using Semantic Web standards, it has been shown that the data in NELL can be transformed to RDF and provided as Linked Open Data as well [83].

In its most recent version, NELL contains roughly 2 million entities and 3 million relations between those. The NELL ontology defines 285 classes and 425 properties.

2.7 DBkWik

DBkWik uses the software that is used to build DBpedia, and applies it to a multitude of Wiki dumps collected from a large Wikifarm, i.e., *Fandom powered by Wikia*.[19]. The most recent version, i.e., DBkWik version 1.1., integrates data extracted from 12,840 Wiki dumps, comprising 14,743,443 articles in total. While DBpedia relies on a manually created ontology and mappings to that, such an ontology does not exist for DBkWik, i.e., the schema for DBkWik needs to be inferred on the fly. Moreover, while for Wikipedia-centric knowledge graphs like DBpedia, there are rarely any duplicate entities (since each entity corresponds to exactly one page in Wikipedia), duplicates exist in DBkWik both on in the A-box and the T-box, and need to be handled in an additional integration step. Figure 2 illustrates the creation process of DBkWik.

[18] The ontology has the complexity *SRF(D)*, it mainly defines a class and a property hierarchy, together with domains, ranges, and disjointness statements.

[19] `http://www.wikia.com/fandom`

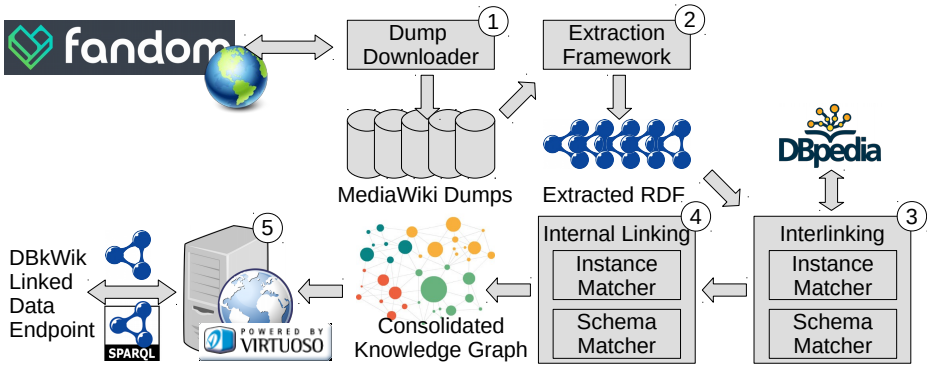


Fig. 2: The DBkWik creation process [26]

As a result, the latest release of DBkWik comprises 11M entities and 96M facts. With 12k classes and 129k relations, the schema is fairly detailed.

2.8 WebIsALOD

While the above knowledge graphs cover a multitude of different relations between entities, the WebIsALOD dataset is focusing solely on building a large-scale hierarchy of concepts, i.e., it builds a lattice of *hypernymy relations*. To that end, it scans the Common Crawl Web corpus[20] for so-called Hearst patterns, e.g. *X such as Y*, to infer relations like `X skos:broader Y`. The graph comes with very detailed provenance data, including the patterns used and the original text snippets, together with their sources [25]. Figure 3 depicts the schema of WebIsALOD, illustrating the richness of its metadata.

In total, WebIsALOD contains more than 212 million entities (however, it does not strictly separate between an *entity* and a *class*), and 400 million hypernymy relations.

Table 1 gives an overview of the knowledge graphs discussed above and their characteristics.[21] The table depicts the number of entities and relations, as well as the average indegree and outdegree of entity (i.e., the average number of ingoing and outgoing relations for each entity), as well as the size of the schema in terms of the number of classes and properties.

2.9 Non-public Knowledge Graphs

Furthermore, company-owned knowledge graphs exist, like the already mentioned Google Knowledge Graph, Google's Knowledge Vault [11], Yahoo's Knowledge Graph [2], Microsoft's Satori, and Facebook's Knowledge Graph. However, those are not publicly available, and hence neither suited to build applications by parties other than the owners, nor can they be analyzed in depth.

[20] `http://commoncrawl.org/`

[21] The numbers are taken from [24] and [61].

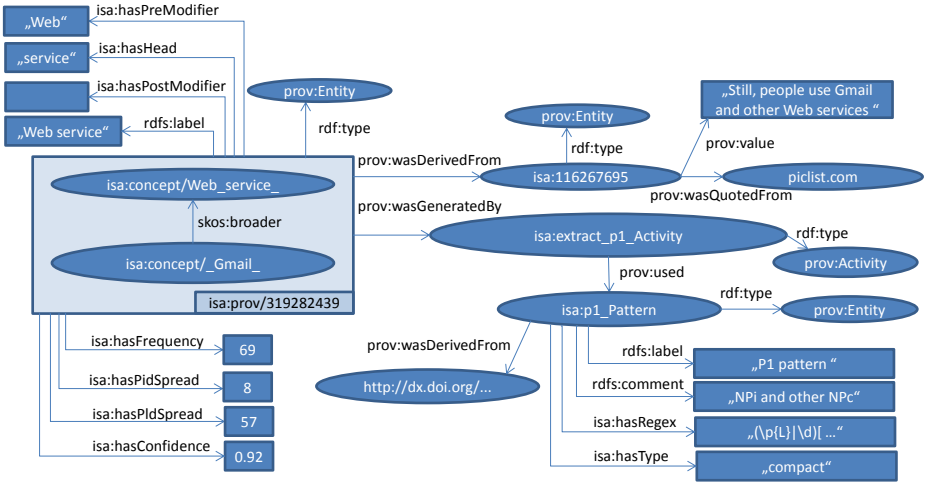


Fig. 3: The schema of the WebIsALOD knowledge graph [24]

Table 1: Public cross-domain knowledge graphs and their size

Knowledge Graph	# Entites	# Facts	Avg. indegree	Avg. outdegree	# Classes	# Properties
OpenCyc	118,499	2,413,894	10.03	9.23	116,822	165
NELL	1,974,297	3,402,971	5.33	1.25	290	1,334
YAGO3	5,130,031	1,435,808,056	9.83	41.25	30,765	11,053
DBpedia	5,109,890	397,831,457	13.52	47.55	754	3,555
DBkWik	11,163,719	91,526,001	0.70	8.17	12,029	128,566
Wikidata	44,077,901	1,633,309,138	9.83	41.25	30,765	11,053
WebIsALOD	212,184,968	400,533,808	3.72	3.31	–	1

3 Using Machine Learning for Building and Refining Knowledge Graphs

As discussed above, large-scale knowledge graph can hardly be created manually [49]. Therefore, heuristics are quite frequently used in the creation of knowledge graphs, i.e., methods that can efficiently create large-scale knowledge graphs, trading off data volume for accuracy.

Machine learning methods can serve as a technique to implement such heuristics. They are widely used both in the creation of a knowledge graph (e.g., for NELL, whose creation is fully machine-learning based), as well as in the subsequent refinement of the generated knowledge graphs [47].

3.1 Type and Relation Prediction

No knowledge graph will ever contain every piece of knowledge that exists in the world. Therefore, all knowledge graphs, no matter whether they are created manually or heuristically, can have only *incomplete* knowledge.

Being grounded in semantic web standards, which adhere to the *open world assumption*, this is not a problem from a logical perspective. However, in many

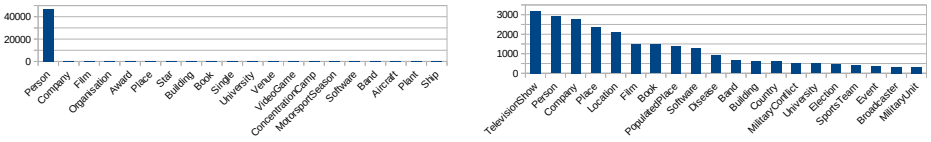


Fig. 4: Distribution of the subject (left) and object (right) types of the `knownFor` relation

application scenarios, the value of a knowledge graph grows with the amount of knowledge encoded therein. Therefore, a lot of work as been devoted on *knowledge graph completion* [47].

As discussed above, semantic web knowledge graphs come with a common schema or ontology, which usually define a – sometimes deeper, sometimes more shallow – hierarchy of classes. Hence, adding missing type information is an important and frequently addressed task in knowledge graph completion.

One of the simplest and hence most scalable approaches is *SDType* [51]. SDType considers each relation in which an entity takes part as an indicator for its type. For example, the statement

```
Germany hasCapital Berlin .
```

involves the two entities `Germany` and `Berlin`, connected by the relation `hasCapital`. Each relation has a specific distribution of types of entities it connects. For example, considering all the pairs of entities connected by the relation `hasCapital` and analyzing their types, there is a high probability that the subject is of type `Country` and the object is of type `City`.

SDType computes a weighted average across all relations an entity (such as `Berlin`) is connected by, using their specific distribution of subject and object types. Here, most of the relations `Berlin` will have a high probability of the entity being a `City`. The weights are assigned by the specificity of the relation w.r.t. types. For example, the relation `hasCapital` is very specific for the subject and the object (it mostly links geographic regions to cities), whereas the relation `knownFor` is only specific for the subject (it is mainly used for subjects of type `Person`), but very unspecific for the object (persons can be known for lots of different things, i.e., the distribution of types for the object of `knownFor` is rather wide). This is illustrated in Fig. 4: when observing a `knownFor` relation between two entities, it is very likely that the subject is of type `Person`, whereas such a conclusion is more difficult for the object.

Due to its simplicity, it is possible to develop well performing implementations of SDType, and the algorithm has meanwhile been integrated in DBpedia and used for building DBpedia releases.

The predictions of SDType are coherent with the existing type hierarchy, given that the type information is fully materialized on the input knowledge graph – i.e., it holds that for each subclass relation $R \sqsubseteq S$, if $R(a)$ is contained in the knowledge graph, then $S(a)$ is also contained. However, this is a characteristic by design, but SDType does not specifically exploit the hierarchy information.

Looking at type prediction from a machine learning perspective, it can be considered a hierarchical multi-label classification problem, i.e., a classification problem where each instance can belong to multiple classes, and those classes form a hierarchy [75]. In [36], we have shown that the type prediction problem can be well solved using a hierarchical classification problem. We used a *local classifier per node*, i.e., we train a classifier for each class in the hierarchy, sampling instances from the class' neighbors in the hierarchy as negative training examples. Since the problem is thereby decomposed into smaller learning problems, the solution becomes scalable and even largely parallelizable.

The scalability of type prediction using hierarchical classification can be even further improved when taking into account that only a small subset of features is required to tell a class from its siblings. For example, for telling a person from an organization, a few relations like `ceo`, `headquarter`, `birthplace`, and `nationality` are sufficient. For telling a movie from a book, relations like `director`, `studio`, `author`, and `publisher` are helpful. Incorporating local feature selection, i.e., determining a small subset of relevant features for each individual classification problem within the hierarchical classification, allows the approach to be scaled to very large knowledge graphs [34].

Type information is often used as a prediction target, but other relations are possible prediction targets as well. Usually, the knowledge graph itself is used as ground truth, i.e., every relation assertion in the knowledge graph is used as a positive training example. Since semantic web knowledge graphs follow the open world assumption, and machine learning classifiers usually expect both positive and negative examples, a common trick to generate negative examples is the so-called *partial completeness assumption* [15] or *local closed world assumption* [11]: it is assumed that if there is set of objects $o_1 ... o_n$ for a given subject s and a relation r, i.e., $r(s, o_1), r(s, o_2), ... r(o_n)$ are contained in the knowledge graph, then this information is complete w.r.t. s, i.e., there is no $o' \notin \{o_1, ..., o_n\}$ so that $r(s, o')$ holds.

Building on this assumption, the approach sketched in [23] uses abstracts in Wikipedia pages are used to train a classifier for each relation. It considers each entity linked within an abstract in a Wikipedia page as a candidate relation. Then, it uses a set of features to learn heuristic rules, such as: The first place to be mentioned in an article about a person is that person's birth place. Using those heuristic rules, about 1M additional statements could be learned for DBpedia. The approach has been shown to work for Wikipedia pages of any language, and even for other Wikis, and is therefore also applicable to DBkWik [22].

3.2 Error Detection

Most heuristics used for knowledge graph construction have to address a trade-off between size and accuracy of the resulting knowledge graph. Decent sized knowledge graphs can only be constructed heuristically by accepting a certain level of noise.

For that reason, a few approaches have been proposed to identify wrong facts in a knowledge graph. Again, a simplistic approach named *SDValidate* follows

the same basic idea of SDType, i.e., using statistical distribution of types in the subject and object positions of a statement [52]. A statement is considered wrong by the approach if the types in the subject and object position differ significantly from the predicate's characteristic distribution.

While SDType is quite stable since it combines a lot of evidence (i.e., many relations in which an instance is involved) and improves with the connectivity of the underlying knowledge graph, SDValidate is less powerful since it usually has only fewer information to work with, i.e., the explicit types set for an entity are typically less than the average degree [61]. Therefore, additional evidence needs to be taken into account to reliably flag wrong relations.

PaTyBRED is a machine-learning based approach that does not only rely on type features, but also on paths in which an entity is involved. It uses relations in the knowledge graph as positive training examples, and creates negative examples by randomly replacing the subject or object with another instance from the knowledge graph.[22]

For training a model, *PaTyBRED* takes into account both types and paths, and therefore, it can also learn that the path $residence(X, Y), country(Y, Z)$ is positive evidence for the relation assertion $nationality(X, Z)$. We have shown that this combination outperforms both approaches based on types and based on paths alone.

An alternative to addressing error detection as a binary classification problem is assigning a confidence score to each entity. This approach is applied to building the WebIsALOD dataset. Here, we use a crowd-sourced gold standard of positive and negative examples (i.e., randomly selected examples from the initial extraction presented to human judges for validation) to train a classifier for telling correct from incorrect relations, using the rich provenance metadata. Instead of discarding the negative examples, we attach the classifier's confidence score as a confidence to each individual statement [24].

Both approaches – identifying and removing errors, as well as using confidence scores – have their advantages and disadvantages. Removing wrong statements leads to a clean and intuitively usable dataset, and also reduces that dataset's size, which can help in the processing. On the other hand, keeping all statements and attaching a confidence score has the advantage of letting the user set an individual threshold, thereby deciding on whether higher recall or higher precision is required for a given task at hand. Furthermore, it opens the opportunity to process the dataset with methods being able to deal with such imprecision, e.g., probabilistic and possibilistic reasoners [55].

3.3 Approximate Local Reasoning

As discussed above, many knowledge graphs come with an ontology, which may be more or less formal. Given that a certain degree of formality is provided,

[22] While more complex strategies for generating meaningful negative training examples exist [29], we have observed no significant qualitative difference in the resulting models' accuracy to creating random negative examples, although those complex strategies are computationally much more expensive.

i.e., the ontology does not only define a class hierarchy and domain and range restrictions for the properties, but also more restrictions, such as disjoint classes, the knowledge graph can be validated against the ontology. For example, if the two classes $City$ and $Team$ are defined as disjoint classes, the range of $playsFor$ is $Team$, and the two axioms $playsFor(Ronaldo, Madrid)$ and $City(Madrid)$ are defined in the knowledge graph, the combination of both axioms can be detected as a violation of the underlying ontology – although deciding *which* of the two axioms is wrong is *not* possible with automatic reasoning.

Some knowledge graphs, such as DBpedia and NELL, validate new axioms against an existing ontology before adding them to the knowledge graph. For DBpedia, there also exists a mapping to the top level ontology *DOLCE* [16], which can be used to detect more violations against the ontology. For example, the statement $award(TimBernersLee, RoyalSociety)$, together with $Organization$ $(RoyalSociety)$ and the range of $award$ being defined as $Award$, is not a violation against the DBpedia ontology, since there is no explicit disjointness between $Award$ and $Organization$. In contrast, the usage of DOLCE defines the corresponding super classes (i.e., $Description$ and $SocialAgent$) as disjoint. Hence, exploiting the additional formalism of the upper level ontology of DOLCE leads to detecting more inconsistencies [54].

An issue with using reasoning, or even local reasoning (i.e., reasoning only on a statement and its related statements) to detect inconsistencies, is computationally expensive: Validating all statements in DBpedia against the DBpedia and DOLCE ontology using a state of the art reasoner like HermiT [17] would take several weeks. In [57], we have introduced an approach that approximates reasoning-based fact checking by exploiting machine learning: we treat the validation problem as a binary classification problem, and let a reasoner validate a small number of statements as consistent or inconsistent. Those examples are then used to train a classifier, whose model approximates the reasoner's behavior. It has been shown that such models can reach an accuracy above 95%, at the same time being some orders of magnitude faster: instead of several weeks, the consistency of all statements in DBpedia can validated in less than two hours.

3.4 Deriving Higher Level Patterns

Once erroneous statements have been identified, there are several ways to proceed. For once, they can be simply removed from the knowledge graph. Second, as for WebIsALOD, they may get lower confidence ratings and defer the decision to a later point.

However, analyzing (potentially) erroneous statements more deeply can also reveal further insights. Clustering the errors may reveal groups of similar errors, which may have a common root [54]. Such common roots may be errors in the ontology, in particular translation statements (e.g., for DBpedia: mappings from a Wikipedia infobox to the common ontology) [60], or the handling of particular kinds of input, such as numerical values [14, 80], dates, links with hashtags, etc. [48].

Once the root cause is identified, it is possible to track back the issue and address it at the respective place, i.e., improving the creation process of the knowledge graph. The advantage is that a group of errors can be addressed with a single fix, and the result is sustainable, i.e., it is also applied for any future version of the same knowledge graph [48]. Moreover, grouping errors can also serve as a sanity check: if an identified error does not belong to any group, i.e., it only occurs as a single, isolated statement, it is often a false negative, i.e., a wrongly identified error [54].

3.5 Evaluating Machine Learning on Knowledge Graphs

Although there is a larger body of work in applying machine learning methods to knowledge graph refinement, there is no common standard evaluation protocol and set of benchmarks. Methods encompass evaluation against (partial) gold standards, the knowledge graph itself (treated as a silver standard), and retrospective evaluations, i.e., evaluating the output of the knowledge graph. With respect to evaluation methods, precision and recall are quite frequently used, but other metrics, e.g., accuracy, area under the precision-recall curve (AUC-PR), area under the ROC curce (AUC-ROC), etc. [6], are also observed. As an additional dimension to result quality, computational performance can also be evaluated.

In [47], we have observed that a majority of all approaches is only evaluated against one knowledge graph, usually DBpedia (see Fig. 5). This often limits the significance of the results, because it is unclear whether the approach overfits and/or consciously or unconsciously exploits of the specific knowledge graph at hand. Therefore, evaluations against multiple knowledge graphs are clearly advised.

3.6 Partial Gold Standard

One common evaluation strategy is to use a partial gold standard. In this methodology, a subset of graph entities or relations are selected and labeled manually. Other evaluations use external knowledge graphs and/or databases as partial gold standards.

For completion tasks, this means that all axioms that *should exist in the knowledge graph* are collected, whereas for correction tasks, a set of axioms in the graph is manually labeled as correct or incorrect. The quality of completion approaches is usually measured in recall, precision, and F-measure, whereas for correction methods, accuracy and/or area under the ROC curve (AUC) are often used alternatively or in addition.

Sourcing partial gold standards from humans can lead to high quality data (given that the knowledge graph and the ontology it uses are not overly complex), but is costly, so that those gold standards are usually small. Exploiting other knowledge graphs based on knowledge graph interlinks (e.g., using Freebase data as a gold standard to evaluate DBpedia) is sometimes proposed to yield larger-scale gold standards, but has two sources of errors: errors in the target knowledge

graph, and errors in the linkage between the two. For example, it has been reported that 20% of the interlinks between DBpedia and Freebase are incorrect [81], and that roughly half of the `owl:sameAs` links between knowledge graphs connect two things which are related, but not *exactly* the same (such as the company *Starbucks* and a particular Starbucks coffee shop) [19].

3.7 Knowledge Graph as Silver Standard

Another evaluation strategy is to use the given knowledge graph itself as a test dataset. Since the knowledge graph is not perfect (otherwise, refinement would not be necessary), it cannot be considered as a *gold standard*. However, assuming that the given knowledge graph is already of reasonable quality, we call this method *silver standard* evaluation, as already proposed in other works [18, 27, 45].

The silver standard method is usually applied to measure the performance of knowledge graph *completion* approaches, where it is analyzed how well relations in a knowledge graph can replicated by a knowledge graph completion method. As for gold standard evaluations, the result quality is usually measured in recall, precision, and F-measure. In contrast to using human annotations, large-scale evaluations are easily possible. The silver standard method is only suitable for evaluating knowledge graph completion, not for error detection, since it assumes the knowledge graph to be correct.

There are two variants of silver standard evaluations: in the more common ones, the entire knowledge graph is taken as input to the approach at hand, and the evaluation is then also carried out on the entire knowledge graph. As this may lead to an overfitting effect (in particular for internal methods), some works also foresee the splitting of the graph into a training and a test partition, which, however, is not as straight forward as, e.g., for propositional classification tasks [44], which is why most papers use the former method. Furthermore, split and cross validation do not fully solve the overfitting effect. For example, if a knowledge graph, by construction, has a bias towards certain kinds of information (e.g., relations are more complete for some classes than for others), approaches overadapting to that bias will be rated better than those which do not (and which may actually perform better in the general case).

Since the knowledge graph itself is not perfect, this evaluation method may sometimes underrate the evaluated approach. More precisely, most knowledge graphs follow the *open world assumption*, i.e., an axiom not present in the knowledge graph may or may not hold. Thus, if a completion approach correctly predicts the existence of an axiom missing in the knowledge graph, this would count as a false positive and thus lower precision. Approaches overfitting to the coverage bias of a knowledge graph at hand may thus be overrated.

3.8 Retrospective Evaluation

For retrospective evaluations, the output of a given approach is given to human judges for annotation, who then label suggested completions or identified errors

Table 2: Overview on evaluation methods with their advantages and disadvantages [47]

Methodology	Advantages	Disadvantages
Partial Gold Standard	highly reliable results reusable	costly to produce balancing problems
Knowledge Graph as Silver Standard	large-scale evaluation feasible subjectiveness is minimized	less reliable results prone to overfitting
Retrospective Evaluation	applicable to disbalanced problems allows for more detailed analysis of approaches	not reusable approaches cannot be compared directly

as correct and incorrect. The quality metric is usually accuracy or precision, along with a statement about the total number of completions or errors found with the approach, and ideally also with a statement about the agreement of the human judges.

In many cases, automatic refinement methods lead to a very large number of findings, e.g., lists of tens of thousands of axioms which are potentially erroneous. Thus, retrospective evaluations are often carried out only on samples of the results. For some approaches which produce higher level artifacts – such as error patterns or completion rules – as intermediate results, a feasible alternative is to evaluate those artifacts instead of the actually affected axioms.

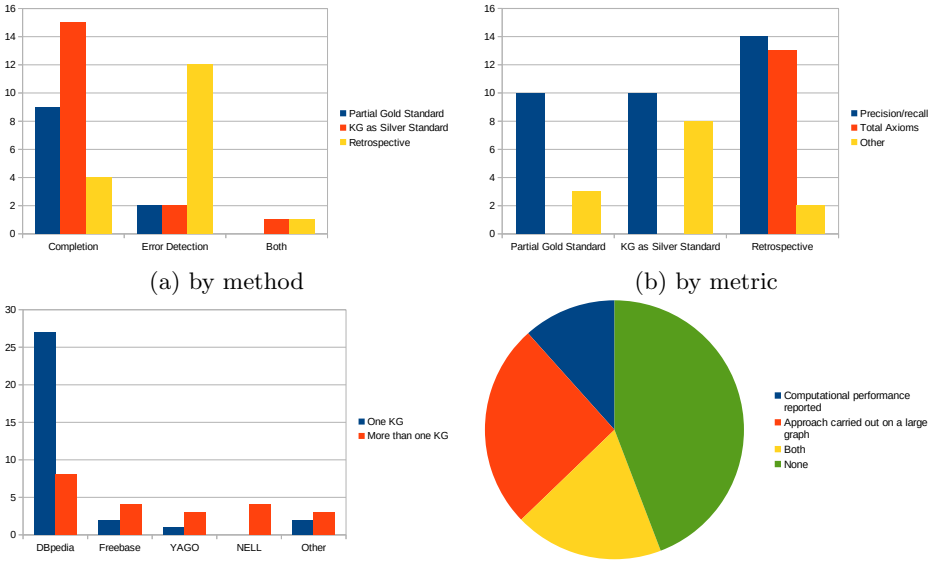
While partial gold standards can be reused for comparing different methods, this is not the case for retrospective evaluations. On the other hand, retrospective evaluations may make sense in cases where the interesting class is rare. For example, when evaluating error detection methods, a sample for a partial gold standard from a high-quality graph is likely not to contain a meaningful number of errors. In those cases, retrospective evaluation methodologies are often preferred over partial gold standards.

Another advantage of retrospective evaluations is that they allow a very detailed analysis of an approach's results. In particular, inspecting the errors made by an approach often reveals valuable findings about the advantages and limitations of a particular approach.

Table 2 sums up the different evaluation methodologies and contrasts their advantages and disadvantages.

3.9 Computational Performance

In addition to the performance w.r.t. correctness and/or completeness of results, computational performance considerations become more important as knowledge graphs become larger. Typical performance measures for this aspect are runtime measurements, as well as memory consumption. Besides explicit measurement of



(a) by method

(b) by metric

(c) by dataset

(d) by performance evaluation

Fig. 5: Breakdown of evaluations observed in [47] by method, metrics, dataset, and computational performance evaluation

computational performance, a "soft" indicator for computational performance is whether an approach has been evaluated (or at least the results have been materialized) on an entire large-scale knowledge graph, or only on a subgraph. The latter is often done when applying evaluations on a partial gold standard, where the retrospective approach is only executed on entities contained in that partial gold standard.

Furthermore, synthetic knowledge graphs, constructed with exactly specified characteristics, can assist in more systematic scalability testing [35].

4 Using Knowledge Graphs for Machine Learning

The common task of data mining is to discover patterns in data, which can be used either to gain a deeper understanding of the data (i.e., descriptive data mining), or for predictions of future developments (i.e., predictive data mining). For solving those tasks, machine learning methods are used.

Fayyad et al. have introduced a prototypical pipeline which leads from data to knowledge, comprising the steps of selection, preprocessing, transformation of data, applying some machine learning or data mining, and interpreting the results [13]. As depicted in Fig. 6 and discussed in detail in [69], Linked Data and, in particular, semantic web knowledge graphs can be used at almost every stage in the process: the first step is to link the data to be analyzed to the corresponding concepts in a knowledge graph. Once the local data is linked to a LOD

dataset, we can explore the existing links in the knowledge graph pointing to related entities in the knowledge graph, as well as follow links to other graphs. In the next step, various techniques for data consolidation, preprocessing, and cleaning are applied, e.g., schema matching, data fusion, value normalization, treatment of missing values and outliers, etc. Next, some transformations on the collected data need to be performed in order to represent the data in a way that it can be processed with any arbitrary data analysis algorithms. Since most algorithms demand a *propositional* form of the input data (i.e., each entity being represented as a set of features), this usually includes a transformation of the knowledge *graph* to a canonical propositional form. After the data transformation is done, a suitable data mining algorithm is selected and applied on the data. In the final step, the results of the data mining process are presented to the user. Here, ease the interpretation and evaluation of the results of the data mining process, where knowledge graphs can be used as well. [69]

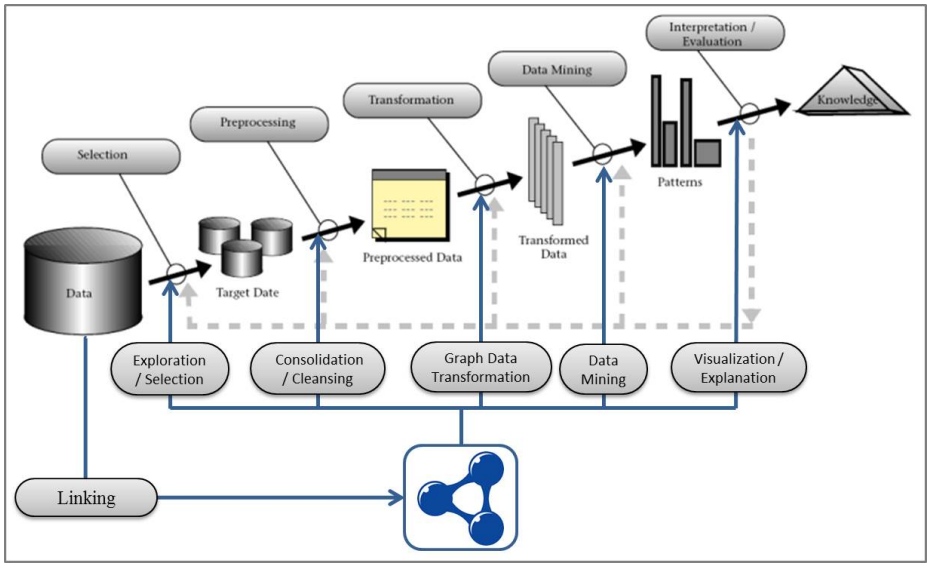


Fig. 6: Enhancing the data mining workflow by Fayyad et al. with Linked Data [69], based on [13]

4.1 Simple Feature Engineering

One of the main issues for exploiting knowledge graphs in machine learning tasks is that most machine learning algorithms are tailored towards propositional data, i.e., data where each instance is represented as a row in a table as a set of attribute values, also called *features*. In contrast, each instance in a knowledge graph is a node in a graph, connected to other nodes via edges. Hence, for being

able to apply a machine learning algorithm to a set of instances in a knowledge graph, those have to be transferred into a propositional form first, a process which we call *propositionalization* [65].

Generally, there are two families of approaches for feature engineering from semantic web knowledge graphs: supervised and unsupervised [53].

For supervised approaches, knowledge about the knowledge graph at hand and its vocabulary is required. This can be either hard coded in the application, e.g., by adding features for genre and actors to a movie [7], or by letting the user specify queries to the knowledge graph, assuming that the user knows the graph's schema in depth [28].

Unsupervised approaches, on the other hand, do not make any assumptions about the dataset at hand. In contrast, they rather create features dynamically for all entities they encounter, e.g., by creating a numerical feature for each numerical literal, a set of binary features for each entity's classes, etc. These approaches have been shown to create valuable data mining features for a lot of machine learning problems, however, they may also lead to a very large set of features, many of which are actually not very valuable, e.g., because of sparsity or uniformity. For example, for a dataset of movies, the information on which novels they are based may be very *sparse* (since most movies are not based on novels). On the other hand, almost[23] of all them will be identified as being of type *Movie*. Therefore, this feature is not very informative, since it contains the same information for (almost) all entities.

To address the potentially large number of non-informative features and pick the subset of those which are valuable, feature subset selection [9, 42] has to be applied. Once the data is in propositional form, standard feature selection algorithms can be applied [62]. However, since knowledge graphs also have information about semantics, it is valuable to incorporate that semantics into the feature subset selection process. For example, for features that form a hierarchy (such as types and categories), we have shown that this hierarchy information bears valuable information for the feature subset selection, since it allows for heuristically discarding features that are either too abstract or too generic [66]. This is illustrated in Figure 7: for a dataset of persons in general, it may be specific enough to distinguish them by profession, whereas for a dataset of athletes, a finer grained set of features may be more useful.

4.2 Feature Vector Generation using RDF2vec

As discussed above, simple feature creation techniques lead to a large number of features, which, at the same time, are often rather sparse, i.e., each feature in itself carries only little information. This lead us to the development of RDF2vec, a method for creating universal, re-usable dense feature vectors from knowledge graphs. We have shown that even with moderately sized feature vectors (200 to 500 features), it is possible to outperform simple feature generation techniques on many tasks, at the same time allowing for re-using the same set of features

[23] Due to the open world assumption, it is likely that this does not hold for all movies.

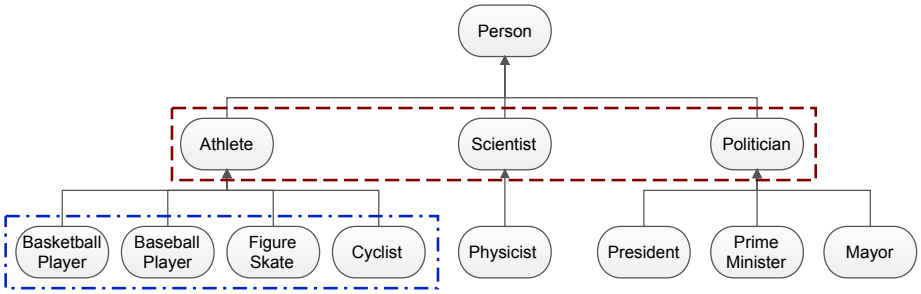


Fig. 7: A sample hierarchy of type features [66]

across tasks, which minimizes the effort of feature generating for a new task. [68, 70].

In a nutshell, RDF2vec picks up the idea of *word2vec* [39], which creates feature vector representations for word. Given a text corpus, word2vec trains a neural network for either predicting the surroundings of a word (context bag of words or CBOW variant), or a word, given its surroundings (skip-gram or SG variant). For example, for the sentence

Trent Reznor founded the band Nine Inch Nails in 1988

CBOW takes a single word (such as *Reznor*) as input and try to predict a set of probably surrounding words (such as *Trent*, *band*, *Nine*, *Inch*, etc.), while SG takes a set of words (e.g., *Trent, founded, the, band, ...*) and tries to predict a word that "'misses"' in that set (e.g., *Reznor*).

The CBOW model predicts target words from context words within a given window. The model architecture is shown in Fig. 8a. The input layer is comprised from all the surrounding words for which the input vectors are retrieved from the input weight matrix, averaged, and projected in the projection layer. Then, using the weights from the output weight matrix, a score for each word in the vocabulary is computed, which is the probability of the word being a target word. Formally, given a sequence of training words $w_1, w_2, w_3, ..., w_T$, and a context window c, the objective of the CBOW model is to maximize the average log probability:

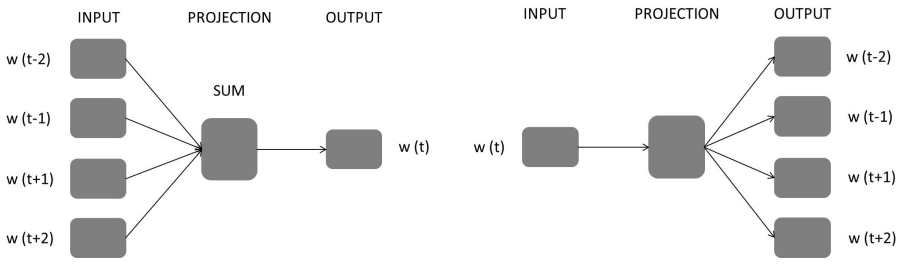
$$\frac{1}{T}\sum_{t=1}^{T} logp(w_t|w_{t-c}...w_{t+c}), \tag{1}$$

where the probability $p(w_t|w_{t-c}...w_{t+c})$ is calculated using the softmax function:

$$p(w_t|w_{t-c}...w_{t+c}) = \frac{exp(\bar{v}^T v'_{w_t})}{\sum_{w=1}^{V} exp(\bar{v}^T v'_w)}, \tag{2}$$

where v'_w is the output vector of the word w, V is the complete vocabulary of words, and $\bar{v}$ is the averaged input vector of all the context words:

$$\bar{v} = \frac{1}{2c}\sum_{-c\leq j\leq c, j\neq 0} v_{w_{t+j}} \tag{3}$$



(a) CBOW architecture (b) Skip-gram architecture

Fig. 8: Architecture of the CBOW and Skip-gram model [68]

The skip-gram model does the inverse of the CBOW model and tries to predict the context words from the target words (Fig. 8b). More formally, given a sequence of training words $w_1, w_2, w_3, ..., w_T$, and a context window c, the objective of the skip-gram model is to maximize the following average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t), \tag{4}$$

where the probability $p(w_{t+j}|w_t)$ is calculated using the softmax function:

$$p(w_o|w_i) = \frac{exp(v'^T_{wo} v_{wi})}{\sum_{w=1}^{V} exp(v'^T_w v_{wi})}, \tag{5}$$

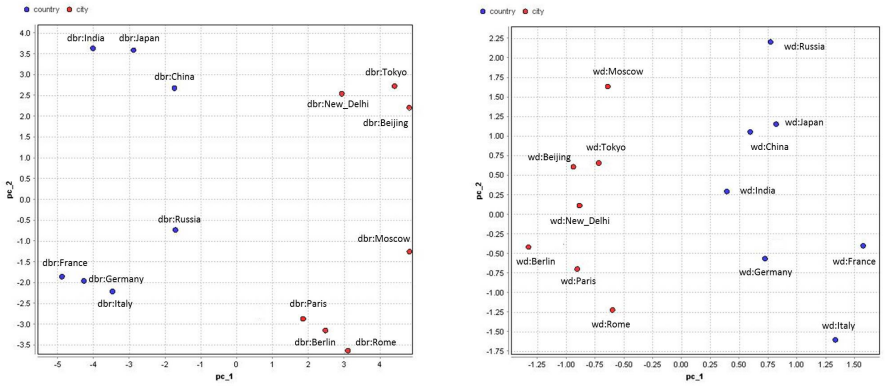
where v_w and v'_w are the input and the output vector of the word w, and V is the complete vocabulary of words.

In both cases, calculating the softmax function is computationally inefficient, as the cost for computing is proportional to the size of the vocabulary. Therefore, two optimization techniques have been proposed, i.e., hierarchical softmax and negative sampling [40]. Empirical studies haven shown that in most cases negative sampling leads to a better performance than hierarchical softmax, which depends on the selected negative samples, but it has higher runtime.

Since knowledge graphs are not a text corpus, pseudo sentences are generated by performing random walks on the knowledge graph, and feeding the resulting sequences into the word2vec model.

Once the training is finished, all words (or, in our case, entities) are projected into a lower-dimensional feature space, and semantically similar words (or entities) are positioned close to each other.

The word2vec vector space representations for words have been shown to have two properties, among others: (1) semantically similar words are close in the resulting vector space, and (2) the direction of both grammatical as well as semantic relations between words remains stable for different pairs of words. Besides being able to use RDF2vec as a versatile and well performing feature vector generator, we have observed the same properties for RDF2vec as well, as depicted in figure 9.



(a) DBpedia vectors

(b) Wikidata vectors

Fig. 9: Two-dimensional PCA projection of the 500-dimensional skip-gram vectors of countries and their capital cities [68]

To visualize the semantics of the vector representations, we employ Principal Component Analysis (PCA) to project the entities' feature vectors into a two dimensional feature space. We selected seven countries and their capital cities, and visualized their vectors as shown in Figure 9. Figure 9a shows the corresponding DBpedia vectors, and Figure 9b shows the corresponding Wikidata vectors. The figure illustrates the ability of the model to automatically organize entities of different types, and preserve the relationship between different entities. For example, we can see that there is a clear separation between the countries and the cities, and the relation *capital* between each pair of country and the corresponding capital city is preserved. Furthermore, we can observe that more similar entities are positioned closer to each other, e.g., we can see that the countries that are part of the EU are closer to each other, and the same applies for the Asian countries.

Using random walks for creating the sequences to be fed into the RDF2vec model is a straight forward and efficient approach, but has its pitfalls as well. Since not all entities and relations in a knowledge graph are equally important, putting more emphasis on the more important paths could lead to an improved embedding. However, telling an important from a less important relation, especially when attempting to create a task agnostic feature representation, is a hard problem.

In [8], we have explored a total of 12 different heuristics to guide the RDF graph walks towards more important paths, based on the frequency of properties, entities, and combinations of properties and entities, as well as the PageRank [3] of entities [76]. In that work, we could show that different heuristics, especially those based on PageRank, can improve the results in many cases, however, the interaction effects between the dataset or task at hand, the machine learning algorithm used, and the heuristic used to generate the paths are still underex-

plored, so that it is difficult to provide a decision guideline on which strategy works best for a given task.

4.3 Example Application 1: Recommender Systems

The purpose of recommender systems is to suggest items to users that they might be interested in, given the past interactions of users and items. Interactions can be implicit (e.g., users looking at items in an online store) or explicit (e.g., users buying items or rating them). [59]

Generally, there are two directions of recommender systems:

Collaborative filtering recommender systems, which rely on the interactions of users and items, and

Content based recommender systems, which rely on item similarity and suggest similar items.

Hybrid approaches, combining collaborative and content based mechanisms, exist as well. For collaborative filtering, there are two variants, i.e., user based and item based.

Content based approaches can benefit massively from knowledge graphs. Given that a dataset of items (e.g., movies, books, music titles) is linked to a knowledge graph, background information about those items can be retrieved from the knowledge graph, both in the form of direct interlinks between items (e.g., movies by the same director), as well as in the form of similar attributes (e.g., movies with a similar budget, runtime, etc.).

For the 2014 Linked Open Data-enabled recommender systems challenge [10], book recommendations had to be computed, with all the books in the dataset linked to DBpedia. There were two tasks: task 1 was the prediction of a user's rating given a user and a product, whereas task 2 was an actual recommendation task (i.e., proposing items to users). We made an experiment [63] using the software RapidMiner[24], together with two extensions: the Linked Open Data extension, which implements most of the algorithms discussed above [62], and the recommender system extension [38].

The features for content-based recommendation were extracted from DBpedia using the RapidMiner Linked Open Data extension. We use the following feature sets for describing a book:

- All *direct types* of a book[25]
- All *categories of a book*
- All *categories of a book's author(s)*
- All *categories of a book including broader categories*[26]

[24] `http://www.rapidminer.com/`

[25] This includes types in the YAGO ontology, which can be quite specific (e.g., *American Thriller Novels*)

[26] The reason for not including broader categories by default is that the category graph is not a cycle-free tree, with some subsumptions being rather questionable.

– All *categories of a book's author(s) and of all other books* by the book's authors
– All *genres of a book* and of all other books by the book's authors
– All *authors that influenced or were influenced* by the book's authors
– A bag of words created from the *abstract* of the book in DBpedia. That bag of words is preprocessed by tokenization, stemming, removing tokens with less than three characters, and removing all tokens less frequent than 3% or more frequent than 80%.

Furthermore, we created a *combined book's feature set*, comprising direct types, qualified relations, genres and categories of the book itself, its previous and subsequent work and the author's notable work, the language and publisher, and the bag of words from the abstract.

Besides DBpedia, we made an effort to retrieve additional features from two additional LOD sources: British Library Bibliography (BLB) and DBTropes[27]. Using the RapidMiner LOD extension, we were able to link more than 90% of the books to BLB entities, but only 15% to DBTropes entities. However, the generated features from BLB were redundant with the features retrieved from DBpedia, and the coverage of DBTropes was too low to derive meaningful features. Hence, we did not pursue those sources further.

In addition to extracting content-based features, we used different generic recommenders in our approach. First, the RDF Book Mashup dataset[28] provides the average score assigned to a book on Amazon. Furthermore, DBpedia provides the number of ingoing links to the Wikipedia article corresponding to a DBpedia instance, and the number of links to other datasets (e.g., other language editions of DBpedia), which we also use as global popularity measures. Finally, SubjectiveEye3D delivers a subjective importance score computed from Wikipedia usage information[29].

To combine all feature sets into a content-based recommender engine, we trained recommender systems on each of the feature sets individually. In order to create a more sophisticated combination of the different base and generic recommenders, we trained a *stacking* model as described in [77]: We trained the base recommenders in 10 rounds in a cross validation like setting, collected their predictions, and learned a stacking model on the predictions. As stacking models, we use linear regression and random decision trees [82], a variant of random forests, and for rank aggregation, we also use Borda rank aggregation. Table 3 shows the results, showing both the RMSE and F1 score on the two tasks of the challenge, as well as the weight β computed for each feature by the linear regression meta learner.

The main learnings from the experiment were:

– Knowledge graphs can provide additional background knowledge, and content based recommender systems trained using that background knowledge outperform collaborative filtering approaches.

[27] `http://bnb.data.bl.uk/` and `http://skipforward.opendfki.de/wiki/DBTropes`
[28] `http://wifo5-03.informatik.uni-mannheim.de/bizer/bookmashup/`
[29] `https://github.com/paulhoule/telepath/wiki/SubjectiveEye3D`

Table 3: Performances of the base and generic recommenders, the number of features used for each base recommender, and the performance of the combined recommenders

		Task 1		Task 2
Recommender	**#Features**	**RMSE**	**LR β**	**F-Score**
Item-based collaborative filtering	–	0.8843	+0.269	0.5621
User-based collaborative filtering	–	0.9475	+0.145	0.5483
Book's direct types	534	0.8895	-0.230	0.5583
Author's categories	2,270	0.9183	+0.098	0.5576
Book's (and author's author books') genres	582	0.9198	+0.082	0.5567
Combined book's properties	4,372	0.9421	+0.0196	0.5557
Author and influenced/influencedBy authors	1,878	0.9294	+0.122	0.5534
Books' categories and broader categories	1,987	0.939	+0.012	0.5509
Abstract bag of words	227	0.8893	+0.124	0.5609
RDT recommender on combined book's properties	4,372	0.9223	+0.128	0.5119
Amazon rating	–	1.037	+0.155	0.5442
Ingoing Wikipedia links	–	3.9629	+0.001	0.5377
SubjectiveEye3D score	–	3.7088	+0.001	0.5369
Links to other datasets	–	3.3211	+0.001	0.5321
Average of all individual recommenders	14	0.8824	–	–
Stacking with linear regression	14	0.8636	–	0.4645
Stacking with RDT	14	**0.8632**	–	0.4966
Borda rank aggregation	14	–	–	**0.5715**

- Combining predictions with a reasonably sophisticated stacking method outperforms simple averaging.
- Generic recommenders (i.e., global item popularity ranks) that are not taking into account the user's preferences are a surprisingly strong baseline.

In [71], we have explored the use of RDF2vec vector models as a means for feature generation for recommender systems. We have shown that content-based methods based on RDF2vec do not only outperform other feature generation techniques, but also state of the art collaborative filtering recommender systems.

Accordingly to Figure 9a, Figure 10 depicts a 2-dimensional PCA projection of a set of movies in the recommendation dataset. We can observe that related movies (Disney movies, Star Trek movies, etc.) have a low distance in the vector space, which allows for the use of content-based recommender systems based on item similarity in the vector space.

4.4 Example Application 2: Explaining Statistics

While recommender systems are an example for predicting machine learning, i.e., training a model for predicting future behavior of users, semantic web knowledge graphs can also be utilized in descriptive machine learning, where the task is to understand a dataset. One example for this is the interpretation of a given dataset, e.g., a collection of statistical observations.

One of the first prototypes for explaining statistical data with semantic web knowledge graphs was *Explain-a-LOD* [46]. The tool takes as input a statistics file, consisting of entities (e.g., regions) and a target variable (e.g., unemployment). It then uses a knowledge graph like DBpedia to identify factors that can be used to explain the target variable, e.g., by measuring correlation.

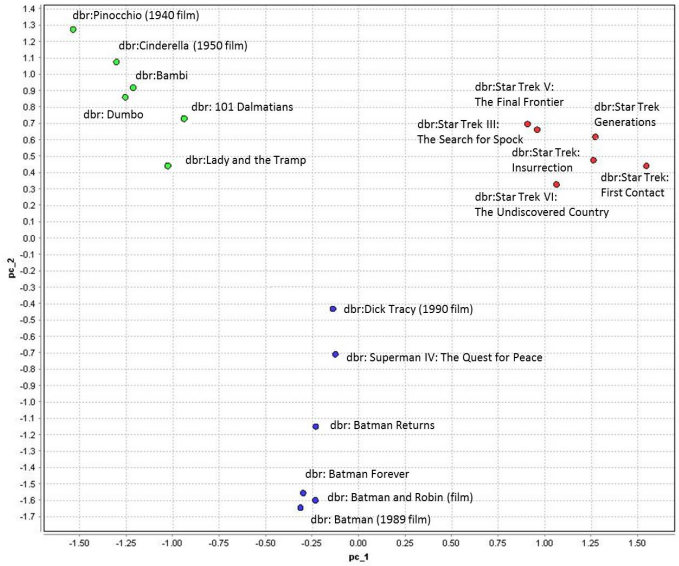


Fig. 10: PCA projection of example movies

As an example, we used a dataset of unemployment in different regions in France. Among others, Explain-a-LOD found the following (positively and negatively) correlating factors, using DBpedia as a starting point, and following links to other knowledge graphs and linked open datasets [64]:

- gross domestic product (GDP) (negative)
- available household income (negative)
- R&D spendings (negative)
- energy consumption (negative)
- population growth (positive)
- casualties in road accidents (negative)
- number of fast food restaurants (positive)
- number of police stations (positive)

In order to further inspect the findings and allowing the user to more intuitively interact with such interpretations, it is also possible to graphically visualize the correlations, as shown in Fig. 12 [67].

While embedding based models have been shown to work well on predictive tasks, their usage in descriptive scenarios is rather limited in contrast. Even if we could find that certain attributes generated by an embedding model explain a statistical variable well, this would not be very descriptive, since the embedding dimensions do not come with a semantic annotation, and hence are not interpretable by humans.

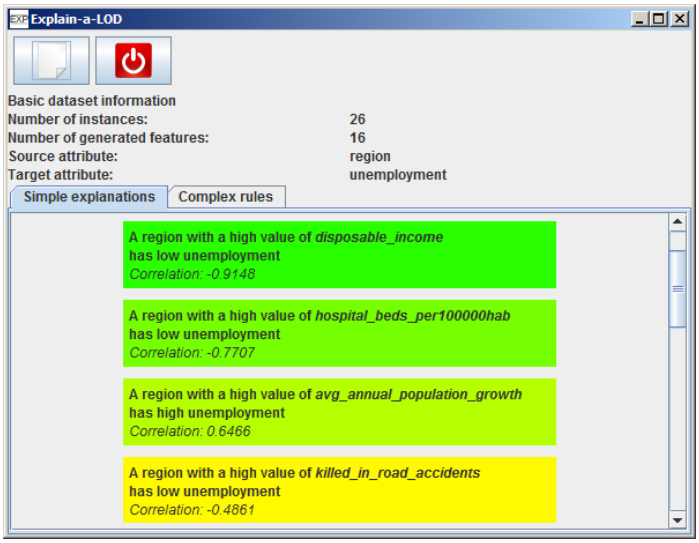


Fig. 11: Screenshot of the original Explain-a-LOD prototype

5 Conclusion and Future Directions

In this chapter, we have looked at a possible symbiosis of machine learning and semantic web knowledge graphs from two angles: (1) using machine learning for knowledge graphs, and (2) using knowledge graphs in machine learning. In both areas, a larger body of works exist, and both are active and vital areas of research.

Our recent approaches to creating knowledge graphs which are complementary to those based on Wikipedia, i.e., WebIsALOD and DBkWik, have shown that there is an interesting potential of generating such knowledge graphs. Whenever creating and extending a knowledge graph or mapping a knowledge graph to existing ones, machine learning techniques can be used, either by having a training set manually curated, or by using knowledge already present in the knowledge graph for training models that add new knowledge or validate the existing information.

With respect to knowledge graph creation, there are still valuable sources on the Web that can be used. The magnitude of Wikis, as utilized by DBkWik, is just one direction, while there are other sources, like structured annotations on Web pages [37] or web tables [31], which can be utilized [78]. Also for Wiki-based extractions, not all information is used to date, e.g., with the potential of tables, lists, and enumerations still being underexplored [56, 43].

While knowledge graphs are often developed in isolation, an interesting approach would be to use them as training data to improve each other, allowing cross fertilization of knowledge graphs. For example, WebIsALOD requires training data for telling instances and categories from each other, which could

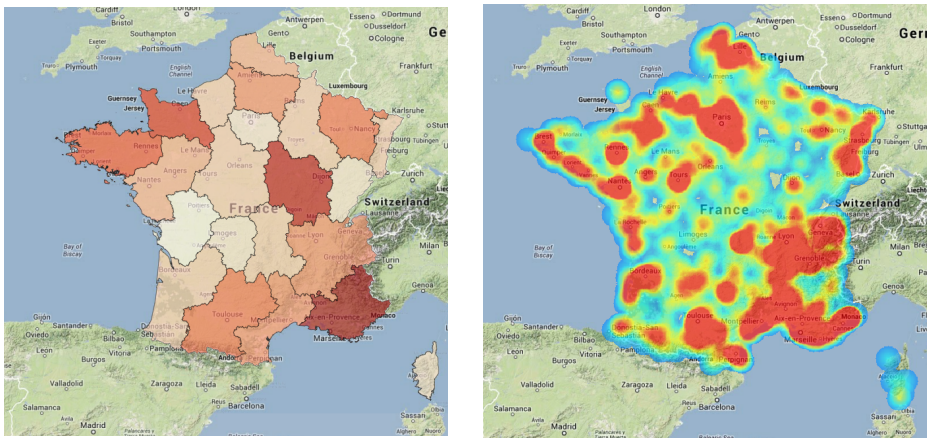


Fig. 12: Visualization of unemployment (left) and density of police stations (right)

be gathered from DBpedia or YAGO. On the other hand, the latter are often incomplete with type information, which could be mined using features from WebIsALOD.

As discussed above, embedding methods are currently not usable for descriptive machine learning. Closing the gap between embeddings (which produce highly valuable features) and simple, but semantics preserving feature generation methods would help developing a new breed of descriptive machine learning methods [50]. To that end, embeddings either need to be semantically enriched with a posteriori developments, or trained in a fashion (e.g., by using pattern or rule learners, such as [21]) that allow for creating embedding spaces which are semantically meaningful.

Acknowledgements

I would like to thank (in alphabetical order) Aldo Gangemi, André Melo, Christian Bizer, Daniel Ringler, Eneldo Loza Mencía, Heiner Stuckenschmidt, Jessica Rosati, Johanna Völker, Julian Seitner, Kai Eckert, Michael Cochez, Nicolas Heist, Petar Ristoski, Renato De Leone, Robert Meusel, Simone Paolo Ponzetto, Stefano Faralli, Sven Hertling, and Tommaso Di Noia for their valuable input to this paper.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. International journal on semantic web and information systems 5(3), 1–22 (2009)
2. Blanco, R., Cambazoglu, B.B., Mika, P., Torzec, N.: Entity Recommendations in Web Search. In: The Semantic Web–ISWC 2013. LNCS, vol. 8219, pp. 33–48 (2013)

3. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems 30(1), 107 – 117 (1998)
4. Bryl, V., Bizer, C.: Learning conflict resolution strategies for cross-language wikipedia data fusion. In: Proceedings of the 23rd International Conference on World Wide Web. pp. 1129–1134. ACM (2014)
5. Carlson, A., Betteridge, J., Wang, R.C., Hruschka Jr, E.R., Mitchell, T.M.: Coupled semi-supervised learning for information extraction. In: Proceedings of the third ACM international conference on Web search and data mining. pp. 101–110 (2010)
6. Caruana, R., Niculescu-Mizil, A.: Data mining in metric space: an empirical analysis of supervised learning performance criteria. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 69–78. ACM (2004)
7. Cheng, W., Kasneci, G., Graepel, T., Stern, D., Herbrich, R.: Automated feature generation from structured knowledge. In: 20th ACM Conference on Information and Knowledge Management (CIKM 2011) (2011)
8. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Biased graph walks for rdf graph embeddings. In: Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics. p. 21. ACM (2017)
9. Dash, M., Liu, H.: Feature selection for classification. Intelligent data analysis 1(3), 131–156 (1997)
10. Di Noia, T., Cantador, I., Ostuni, V.C.: Linked open data-enabled recommender systems: Eswc 2014 challenge on book recommendation. In: Semantic Web Evaluation Challenge. pp. 129–143. Springer (2014)
11. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 601–610. ACM (2014)
12. Ehrlinger, L., Wöß, W.: Towards a definition of knowledge graphs. In: SEMANTiCS (2016)
13. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P.: Advances in knowledge disc overy and data mining. pp. 1–34. American Association for Artificial Intelligence, Menlo Park, CA, USA (1996), http://dl.acm.org/citation.cfm?id=257938.257942
14. Fleischhacker, D., Paulheim, H., Bryl, V., Völker, J., Bizer, C.: Detecting errors in numerical linked data using cross-checked outlier detection. In: International Semantic Web Conference. pp. 357–372. Springer (2014)
15. Galárraga, L.A., Teflioudi, C., Hose, K., Suchanek, F.: AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In: 22nd international conference on World Wide Web. pp. 413–422 (2013)
16. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L.: Sweetening Ontologies with DOLCE. In: 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW2002). Springer (2002)
17. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: an owl 2 reasoner. Journal of Automated Reasoning 53(3), 245–269 (2014)
18. Groza, T., Oellrich, A., Collier, N.: Using silver and semi-gold standard corpora to compare open named entity recognisers. In: IEEE International Conference on Bioinformatics and Biomedicine (BIBM). pp. 481–485. IEEE, Piscataway, New Jersey (2013), http://dx.doi.org/10.1109/BIBM.2013.6732541
19. Halpin, H., Hayes, P.J., McCusker, J.P., McGuinness, D.L., Thompson, H.S.: When owl:sameAs Isn't the Same: An Analysis of Identity in Linked Data. In: The Seman-

tic Web – ISWC 2010, LNCS, vol. 6496, pp. 305–320. Springer, Berlin Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-17746-0_20
20. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th conference on Computational linguistics-Volume 2. pp. 539–545. Association for Computational Linguistics (1992)
21. Hees, J., Bauer, R., Folz, J., Borth, D., Dengel, A.: An evolutionary algorithm to learn SPARQL queries for source-target-pairs: Finding patterns for human associations in dbpedia. CoRR abs/1607.07249 (2016), http://arxiv.org/abs/1607.07249
22. Heist, N., Hertling, S., Paulheim, H.: Language-agnostic relation extraction from abstracts in wikis. Information 9(4), 75 (2018)
23. Heist, N., Paulheim, H.: Language-agnostic relation extraction from wikipedia abstracts. In: International Semantic Web Conference. pp. 383–399. Springer (2017)
24. Hertling, S., Paulheim, H.: Webisalod: providing hypernymy relations extracted from the web as linked open data. In: International Semantic Web Conference. pp. 111–119. Springer (2017)
25. Hertling, S., Paulheim, H.: Provisioning and usage of provenance data in the webisalod knowledge graph. In: First International Workshop on Contextualized Knowledge Graphs (2018)
26. Hofmann, A., Perchani, S., Portisch, J., Hertling, S., Paulheim, H.: Dbkwik: towards knowledge graph creation from thousands of wikis. In: International Semantic Web Conference (Posters and Demos) (2017)
27. Kang, N., van Mulligen, E.M., Kors, J.A.: Training text chunkers on a silver standard corpus: can silver replace gold? BMC bioinformatics 13(1), 17 (2012), http://dx.doi.org/10.1186/1471-2105-13-17
28. Kappara, V.N.P., Ichise, R., Vyas, O.: Liddm: A data mining system for linked data. In: Workshop on Linked Data on the Web (LDOW2011) (2011)
29. Lao, N., Cohen, W.W.: Relational retrieval using a combination of path-constrained random walks. Machine learning 81(1), 53–67 (2010)
30. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web Journal 6(2) (2013)
31. Lehmberg, O., Ritze, D., Meusel, R., Bizer, C.: A large public corpus of web tables containing time and context metadata. In: Proceedings of the 25th International Conference Companion on World Wide Web. pp. 75–76. International World Wide Web Conferences Steering Committee (2016)
32. Lenat, D.B.: CYC: A large-scale investment in knowledge infrastructure. Communications of the ACM 38(11), 33–38 (1995)
33. Mahdisoltani, F., Biega, J., Suchanek, F.M.: Yago3: A knowledge base from multilingual wikipedias. In: CIDR (2013)
34. Melo, A., Paulheim, H.: Local and global feature selection for multilabel classification with binary relevance. Artificial intelligence review pp. 1–28 (2017)
35. Melo, A., Paulheim, H.: Synthesizing knowledge graphs for link and type prediction benchmarking. In: European Semantic Web Conference. pp. 136–151. Springer (2017)
36. Melo, A., Paulheim, H., Völker, J.: Type prediction in rdf knowledge bases using hierarchical multilabel classification. In: Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics. p. 14. ACM (2016)

37. Meusel, R., Petrovski, P., Bizer, C.: The webdatacommons microdata, rdfa and microformat dataset series. In: International Semantic Web Conference. pp. 277–292. Springer (2014)
38. Mihelčić, M., Antulov-Fantulin, N., Bošnjak, M., Šmuc, T.: Extending rapidminer with recommender systems algorithms. In: RapidMiner Community Meeting and Conference (RCOMM 2012) (2012)
39. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
40. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
41. Miller, G.A.: WordNet: a lexical database for English. Communications of the ACM 38(11), 39–41 (1995), http://dx.doi.org/10.1145/219717.219748
42. Molina, L.C., Belanche, L., Nebot, À.: Feature selection algorithms: A survey and experimental evaluation. In: International Conference on Data Mining (ICDM). pp. 306–313. IEEE (2002)
43. Muñoz, E., Hogan, A., Mileo, A.: Using linked data to mine rdf from wikipedia's tables. In: Proceedings of the 7th ACM international conference on Web search and data mining. pp. 533–542. ACM (2014)
44. Neville, J., Jensen, D.: Iterative classification in relational data. In: Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data. pp. 13–20. AAAI, Palo Alto, CA (2000), http://www.aaai.org/Library/Workshops/2000/ws00-06-007.php
45. Nothman, J., Ringland, N., Radford, W., Murphy, T., Curran, J.R.: Learning multilingual named entity recognition from Wikipedia. Artificial Intelligence 194, 151–175 (2013), http://dx.doi.org/10.1016/j.artint.2012.03.006
46. Paulheim, H.: Generating possible interpretations for statistics from linked open data. In: Extended Semantic Web Conference. pp. 560–574. Springer (2012)
47. Paulheim, H.: Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. Semantic Web (2016)
48. Paulheim, H.: Data-driven joint debugging of the dbpedia mappings and ontology. In: European Semantic Web Conference. pp. 404–418. Springer (2017)
49. Paulheim, H.: How much is a triple? – estimating the cost of knowledge graph creation. In: ISWC Blue Sky Ideas (2018), to appear
50. Paulheim, H.: Make embeddings semantic again! In: ISWC Blue Sky Ideas (2018), to appear
51. Paulheim, H., Bizer, C.: Type inference on noisy rdf data. In: International Semantic Web Conference. pp. 510–525. Springer (2013)
52. Paulheim, H., Bizer, C.: Improving the quality of linked data using statistical distributions. International Journal on Semantic Web and Information Systems (IJSWIS) 10(2), 63–86 (2014)
53. Paulheim, H., Fürnkranz, J.: Unsupervised Generation of Data Mining Features from Linked Open Data. In: International Conference on Web Intelligence, Mining, and Semantics (WIMS'12) (2012)
54. Paulheim, H., Gangemi, A.: Serving DBpedia with DOLCE–More than Just Adding a Cherry on Top. In: International Semantic Web Conference. LNCS, vol. 9366. Springer (2015)
55. Paulheim, H., Pan, J.Z.: Why the semantic web should become more imprecise (2012)
56. Paulheim, H., Ponzetto, S.P.: Extending dbpedia with wikipedia list pages. NLP-DBPEDIA ISWC 13 (2013)

57. Paulheim, H., Stuckenschmidt, H.: Fast approximate a-box consistency checking using machine learning. In: International Semantic Web Conference. pp. 135–150. Springer (2016)
58. Pellissier Tanon, T., Vrandečić, D., Schaffert, S., Steiner, T., Pintscher, L.: From freebase to wikidata: The great migration. In: Proceedings of the 25th International Conference on World Wide Web. pp. 1419–1428 (2016)
59. Ricci, F., Rokach, L., Shapira, B.: Introduction to recommender systems handbook. In: Recommender systems handbook, pp. 1–35. Springer (2011)
60. Rico, M., Mihindukulasooriya, N., Kontokostas, D., Paulheim, H., Hellmann, S., Gómez-Pérez, A.: Predicting incorrect mappings: A data-driven approach applied to dbpedia (2018)
61. Ringler, D., Paulheim, H.: One knowledge graph to rule them all? analyzing the differences between dbpedia, yago, wikidata & co. In: Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz). pp. 366–372. Springer (2017)
62. Ristoski, P., Bizer, C., Paulheim, H.: Mining the web of linked data with rapidminer. Web Semantics: Science, Services and Agents on the World Wide Web 35, 142–151 (2015)
63. Ristoski, P., Mencía, E.L., Paulheim, H.: A hybrid multi-strategy recommender system using linked open data. In: Semantic Web Evaluation Challenge. pp. 150–156. Springer (2014)
64. Ristoski, P., Paulheim, H.: Analyzing statistics with background knowledge from linked open data. In: Workshop on Semantic Statistics (2013)
65. Ristoski, P., Paulheim, H.: A comparison of propositionalization strategies for creating features from linked open data. Linked Data for Knowledge Discovery 6 (2014)
66. Ristoski, P., Paulheim, H.: Feature selection in hierarchical feature spaces. In: Discovery Science (2014)
67. Ristoski, P., Paulheim, H.: Visual analysis of statistical data on maps using linked open data. In: International Semantic Web Conference. pp. 138–143. Springer (2015)
68. Ristoski, P., Paulheim, H.: Rdf2vec: Rdf graph embeddings for data mining. In: International Semantic Web Conference. pp. 498–514. Springer (2016)
69. Ristoski, P., Paulheim, H.: Semantic web in data mining and knowledge discovery: A comprehensive survey. Web semantics: science, services and agents on the World Wide Web 36, 1–22 (2016)
70. Ristoski, P., Rosati, J., Noia, T.D., Leone, R.D., Paulheim, H.: Rdf2vec: Rdf graph embeddings and their applications. Semantic Web (2018)
71. Rosati, J., Ristoski, P., Di Noia, T., Leone, R.d., Paulheim, H.: Rdf graph embeddings for content-based recommender systems. In: CEUR workshop proceedings. vol. 1673, pp. 23–30. RWTH (2016)
72. Sarjant, S., Legg, C., Robinson, M., Medelyan, O.: "All you can eat" ontology-building: Feeding Wikipedia to Cyc. In: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 01. pp. 341–348. IEEE Computer Society, Piscataway, NJ (2009), http://dx.doi.org/10.1109/WI-IAT.2009.60
73. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: International Semantic Web Conference. pp. 245–260. Springer (2014)

74. Seitner, J., Bizer, C., Eckert, K., Faralli, S., Meusel, R., Paulheim, H., Ponzetto, S.P.: A large database of hypernymy relations extracted from the web. In: LREC (2016)
75. Silla, Jr., C.N., Freitas, A.A.: A survey of hierarchical classification across different application domains. Data Min. Knowl. Discov. 22(1-2), 31–72 (Jan 2011)
76. Thalhammer, A., Rettinger, A.: PageRank on Wikipedia: Towards General Importance Scores for Entities. In: The Semantic Web: ESWC 2016 Satellite Events, pp. 227–240. Springer International Publishing, Crete, Greece (2016)
77. Ting, K.M., Witten, I.H.: Issues in stacked generalization. Artificial Intelligence Research 10(1) (1999)
78. Tonon, A., Felder, V., Difallah, D.E., Cudré-Mauroux, P.: Voldemortkg: Mapping schema. org and web entities to linked open data. In: International Semantic Web Conference. pp. 220–228. Springer (2016)
79. Vrandečić, D., Krötzsch, M.: Wikidata: a Free Collaborative Knowledge Base. Communications of the ACM 57(10), 78–85 (2014)
80. Wienand, D., Paulheim, H.: Detecting incorrect numerical data in dbpedia. In: European Semantic Web Conference. pp. 504–518. Springer (2014)
81. Zaveri, A., Kontokostas, D., Sherif, M.A., Bühmann, L., Morsey, M., Auer, S., Lehmann, J.: User-driven Quality Evaluation of DBpedia. In: 9th International Conference on Semantic Systems (I-SEMANTICS '13). pp. 97–104. ACM, New York (2013), http://dx.doi.org/10.1145/2506182.2506195
82. Zhang, X., Yuan, Q., Zhao, S., Fan, W., Zheng, W., Wang, Z.: Multi-label classification without the multi-label cost. In: Proceedings of the Tenth SIAM International Conference on Data Mining (2010)
83. Zimmermann, A., Gravier, C., Subercaze, J., Cruzille, Q.: Nell2RDF: Read the Web, and Turn it into RDF. In: Knowledge Discovery and Data Mining meets Linked Open Data. CEUR Workshop Proceedings, vol. 992, pp. 2–8 (2013), http://ceur-ws.org/Vol-992/