Detecting Errors in Linked Data Using Ontology Learning and Outlier Detection

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Daniel Fleischhacker
aus Groß-Gerau

Mannheim, 2015

Dekan:	Professor Dr. Heinz Jürgen Müller, Universität Mannheim
Referent:	Professor Dr. Heiner Stuckenschmidt, Universität Mannheim
Korreferent:	Professor Dr. Felix Naumann, Universität Potsdam

Tag der mündlichen Prüfung: 11. März 2016

Abstract

Linked Data is one of the most successful implementations of the Semantic Web idea. This is demonstrated by the large amount of data available in repositories constituting the Linked Open Data cloud and being linked to each other. Many of these datasets are not created manually but are extracted automatically from existing datasets. Thus, extraction errors, which a human would easily recognize, might go unnoticed and could hence considerably diminish the usability of Linked Data. The large amount of data renders manual detection of such errors unrealistic and makes automatic approaches for detecting errors desirable. To tackle this need, this thesis focuses on error detection approaches on the logical level and on the level of numerical data. In addition, the presented methods operate solely on the Linked Data dataset without a requirement for additional external data.

The first two parts of this work deal with the detection of logical errors in Linked Data. It is argued that an upstream formalization of the knowledge, which is required for the error detection, into ontologies and then applying it in a separate step has several advantages over approaches that skip the formalization step. Consequently, the first part introduces inductive approaches for learning highly expressive ontologies from existing instance data as a basis for detecting logical errors. The proposed and evaluated techniques allow to learn class disjointness axioms as well as several property-centric axiom types from instance data.

The second part of this thesis operates on the ontologies learned by the approaches proposed in the previous part. First, their quality is improved by detecting errors possibly introduced by the automatic learning process. For this purpose, a pattern-based approach for finding the root causes of ontology errors that is tailored to the specifics of the learned ontologies is proposed and then used in the context of ontology debugging approaches. To conclude the logical error detection, the usage of learned ontologies for finding erroneous statements in Linked Data is evaluated in the final chapter of the second part. This is done by applying a pattern-based error detection approach that employs the learned ontologies to the DBpedia dataset and then manually evaluating the results which finally shows the adequacy of learned ontologies for logical error detection.

The final part of this thesis complements the previously shown logical error detection with an approach to detect data-level errors in numerical values. The presented method applies outlier detection techniques to the datatype property values to find potentially erroneous ones whereby the result and performance of the detection step is improved by the introduction of additional preprocessing steps. Furthermore, a subsequent cross-checking step is proposed which allows to handle the outlier detection imminent problem of natural outliers.

In summary, this work introduces a number of approaches that allow to detect errors in Linked Data without a requirement for additional, external data. The generated lists of potentially erroneous facts can be a first indication for errors and the intermediate step of learning ontologies makes the full workflow even more suited for being used in a scenario which includes human interaction.

Zusammenfassung

Linked Data ist eine der erfolgreichsten Umsetzungen der Ideen des Semantic Web, was insbesondere an den großen Datenmengen zu erkennen ist, welche im Rahmen der Linked Open Data Cloud verfügbar sind. Viele dieser Datensätze sind jedoch nicht manuell erstellt, sondern mittels automatisierter Ansätze aus bereits vorhandenen Datensätzen extrahiert worden. Hierdurch enthalten sie viele Fehler, welche bei einer manuellen Erstellung der Daten hätten erkannt werden können, nun jedoch die Verwendbarkeit der Daten einschränken. Da eine nachgelagerte manuelle Fehlererkennung aufgrund der großen Datenmenge nicht praktikabel ist, ist ein automatisierter Ansatz zur Erkennung von Datenfehlern wünschenswert. Die vorliegende Arbeit setzt hier an, indem sie Methoden zur Erkennung von Datenfehlern auf der logischen und der numerischen Ebene einführt. Ein Hauptaugenmerk liegt hierbei auf Ansätzen, welche ohne zusätzliche, externe Datenquellen direkt auf dem Linked Data Datensatz angewandt werden können.

Die ersten beiden Teile dieser Arbeit befassen sich mit der Erkennung von Fehlern auf der logischen Ebene. Grundlegend wird hierbei zugunsten der Nutzung von Ontologien zur vorgelagerten Formalisierung des Wissens, welches für die Fehlererkennung genutzt wird, argumentiert. Daher werden im ersten Teil dieser Arbeit induktive Ansätze zum Lernen von expressiven Ontologien präsentiert, welche Ontologie-Axiome für die Disjunktheit von Klassen sowie einer Reihe von Property-spezifischen Axiomen unterstützen.

Der zweite Teil dieser Arbeit baut anschließend auf den dieserart gelernten Ontologien auf. Zur Erkennung von Fehlern in den gelernten Ontologien wird eine muster-basierte Methode zur Bestimmung der Ursachen von Ontologie-Fehlern vorgeschlagen, welche speziell auf die in den Ontologien verwendeten Axiomarten und ihre Verwendung zugeschnitten ist. Diese Methode wird daraufhin im Rahmen von verschiedenen Ansätzen zur Behebung von Fehlern in Ontologien genutzt und die Ergebnisse ausgewertet. Schließlich wird die Erkennung logischer Fehler mittels der gelernten Ontologien anhand von Experimenten auf dem DBpedia Datensatz demonstriert. Die anschließende Auswertung zeigt die Anwendbarkeit gelernter Ontologien zur Erkennung logischer Fehler.

Im dritten Teil wird daraufhin eine Methode zur Erkennung von Fehlern in numerischen Werten erweitert, welche auf Techniken zur Erkennung von Ausreißern basiert. Hierbei verbessert ein Vorverarbeitungsschritt die Genauigkeit des Ansatzes und reduziert gleichzeitig die benötigte Verarbeitungszeit. Ein zusätzlicher Nachbearbeitungsschritt erlaubt die Einbindung von im Linked Data Datensatz verbundenen Werten zur Behandlung von natürlichen Ausreißern.

Zusammenfassend präsentiert diese Arbeit Ansätze, deren Kombination es erlaubt Fehler in Linked Data auf logischer und numerischer Ebene zu erkennen und dabei unabhängig von externen Datenquellen zu sein. Die Listen potentieller Fehler, welche durch diese Ansätze erstellt werden, können anschließend manuell geprüft und wenn notwendig behoben werden. Der Zwischenschritt über Ontologien eröffnet hierbei zusätzliche Möglichkeiten im interaktiven Einsatz.

Contents

List of Algorithms

List of Figures

List of Tables

Acknowledgement

Thanks to all the people without whose support this work would not exist.

Thanks to Heiner for giving me the opportunity to write it although he had to take so many bureaucratic hurdles. Thanks to Michael for many helpful discussions and reassurance. Thanks to Dominique for her last-minute annotation sessions.

Many thanks to Sina for her support and also accepting that I put many hours of additional work in this thesis though we had so much to see on the other side of the world. Thanks to my parents for always supporting me.

Chapter 1

Introduction

The World Wide Web certainly presents the most extensive collection of knowledge ever available to humans. This is not only due to large encyclopedic websites like Wikipedia but rather this accumulation of knowledge and its growth is fostered by the Web's decentralized nature where anyone has the possibility to introduce new websites. By means of links to other websites, these new sites can also be integrated directly into the overall structure without large effort. However, the largest share of the information available on the Internet is made for human consumption since most data is expressed in natural language with arbitrary structure. This severely limits the possibilities of automatically processing the data and thus hinders the realization of many potentially beneficial use cases. Even where data is made available to automatic methods, e.g., by means of providing programmatic access to the data, it often relies on proprietary formats. Hence, applications consuming the data have to be specifically adapted to it on a per-provider base.

As a reply to these shortcomings, the so-called *Semantic Web* has received much attention of researchers during the last years. One of its main ambitions is to not only provide information contained in the form of natural language, which is hard to use by automatic means, but provide data using a common, open format which also allows automatic processing without the need to first handle the ambiguity of language to extract the relevant information. This basic structured representation is extended by allowing to interlink data from different providers. For this purpose, the concept of *Uniform Resource Identifiers* (URI) is employed as in the World Wide Web. These links between datasets help to reduce the number of isolated data repositories and thus can generate additional benefits exceeding the sum of the single datasets' benefits. Thus, the Semantic Web takes the two keystones of the Web's success: decentralized organization and interlinking.

This basic idea is extended further by another fundamental building block of the idea of the Semantic Web. This extension creates the to give the data semantically founded underpinnings by means of *ontologies* that provide more formal specifications of the meaning of the provided data. The latter aspect aims at diminishing the problem of applications that have to be adapted to each data source

specifically even if the data only differs by a small degree. Therefore, there is more chance of interoperability. Ontologies as used in the Semantic Web also exceed a purely declarative nature. They allow to describe concepts in the data as well as to give details on relations between these concepts in a well-defined way to capture relevant domain knowledge including parts of the semantics which can then be used to infer further knowledge. The overall idea of the Semantic Web and possible usages are sketched by Berners-Lee et al. [15] where data availability and properly defined semantics give plenty of new possibilities regarding interactions between different systems. Finally, the realization of the Semantic Web would provide whole new possibilities to utilize data and achieve additional benefits for the users whose extent could be only measured by those brought by the introduction of the World Wide Web itself. This makes the Semantic Web sometimes being referred to as *Web 3.0*.

Even though the overall appeal of the vision is recognizable, the actual spreading of semantic technologies is still very limited. The main cause of this is assumed to lie in the additional effort for content and data providers that is required for providing information not only for human consumption but also in a way that machines can profitably use it. More recently, efforts to make data available which omit the detailed specification of semantics gained more traction. Above all, the schema.org initiative[1] proved to be successful in this direction. It provides a common vocabulary that can be used to annotate websites and simplifies the automatic identification of relevant information. This way of annotating website content is especially pushed into practical use since it is backed by Google, Microsoft (Bing), Yahoo and Yandex where it is used for extracting information relevant in their search engine results. *Linked Data* [14] shares the idea of schema.org regarding the concentration on providing data and putting less focus on formalizing the semantics but, in contrast to just annotating the data, Linked Data takes the idea of interlinking datasets into account. Basically, Linked Data fosters the provision of data by means of open format and defines guidelines which give the possibility of defining links between the dataset. In particular, it encourages the usage of URIs for identifying and accessing data.

Linked Data is not as much targeted on a limited number of main consumers as schema.org. This also means that its quality cannot simply be assessed by testing it with these consumers. Instead there are many works that propose quality criteria for Linked Data. Zaveri et al. [103] assembled an overview about the most important criteria and categorized them into six main groups of dimensions. Moreover, they summarize metrics which can be used to assess the different dimensions. The *accessibility* dimensions, containing criteria like availability of the dataset, licensing issues and interlinking to other datasets, capture how well a consumer is able to retrieve the data and use it on a technical and legal level. The dimensions of *trust* aims to achieve perceived trustworthiness of the data which is influenced by criteria like the reputation of the data source and the verifiability of the data. In ad-

[1] `http://schema.org`

dition, the *dynamicity* covers the time-dependent aspects of data, e.g., how current the data is and how timely it is updated if the base data changes. Criteria whose influence on the usability of the data are more task-dependent are summarized in the *contextual* dimensions. This includes the available amount of data and its completeness which determine whether a given dataset can be applied in a certain use case. Data design-specific criteria are captured by the *representational* dimensions including understandability, i.e., how comprehensible the data is for human data consumers, and the interpretability, describing whether an appropriate notation is used that conforms to the user's ability to process it. Finally, the *intrinsic* dimensions are described. These cover all aspects not depending on the context of the user. Zaveri et al. included three dimensions into this category. First, the *accuracy* of the data describes the data's syntactic correctness and its correctness regarding the real world facts it is representing. Second, the data's *consistency* covers contradictions regarding the formal representation and also the inference mechanisms applied to it. Third, the conciseness of the data represents another quality dimension. For being concise, data should not contain redundant information on the schema level (referred to as intensional conciseness) and the data level (extensional conciseness). Given the fact that the intrinsic features are independent from the user's context, they can be considered as the most far-reaching criteria regarding the data's overall usability.

Considering the currently available Linked Data datasets, many of them are not natively created as Linked Data or as structured data at all but are generated from semi-structured or unstructured data. Although this helps to kick start the availability of Linked Data, also making it more attractive to use it as data source, it poses some major challenges. In particular, methods for extracting structured data from semi-structured data face many of the same problems as when trying to extract information from usual, natural language-based websites like ambiguity of information or the variety with respect to their representation. This leads to the extraction of erroneous information which then gets included into the dataset. When actually using the data, data errors are a problem which has shown to be manageable when working on natural language documents by relying on statistical methods and exploiting the large amount of data available. For structured data like Linked Data however, this poses a much larger problem for several reasons. First, while there are large amounts of unstructured information available in documents which allow to handle data shortcomings or imprecisions, the same does not hold for structured data. This limits the possibilities to handle the problems by merely applying statistical methods. Secondly, the advantage of structured data lies in the fact that it provides a structure for the data and discharges the data consumer from doing additional complex processing of the supplied data. It follows that either errors in the structured data are much more likely to cause problems on the data consumer site because of fewer safety measures in the processing or the data consumer has to process the data himself. The latter obviously means that the data loses many of its potential benefits compared to unstructured data. Hence, proper quality assurance for Linked Data is of great importance, especially for datasets

which are not manually crafted but automatically extracted from other datasets.

This way of creating Linked Data datasets causes them to often contain errors with respect to their conformity with the real-world facts they describe and might cause syntactic deviations that make the data hardly interpretable for data consumers. Consequently, these datasets show weaknesses regarding the intrinsic dimensions of Linked Data quality. Wrongly extracted data leads to flaws in the accuracy of the resulting dataset. This limits the direct usability of the data regardless of the context it is used. An obvious first step into fixing these data flaws is the detection of errors in the dataset. That is why, in this work, we investigate methods for detecting errors in Linked Data so that the errors can be further assessed by humans. Purely manual quality assurance is a cumbersome and laborious task especially for larger datasets. On the other hand, fully automatic approaches are not suitable for reaching a satisfying level of quality. This holds even more for cases where the initial error was introduced by automatic methods. Thus, the preferred way of assuring quality in Linked Data is a compromise of both approaches which includes the accuracy of human judgment but tries to reduce the workload put onto the human annotator. Our work presented here is focused on developing methods that are able to achieve this. We concentrate on two facets of Linked Data correctness: the correctness of the links between entities and the correctness of numerical values.

For detecting the linking errors, we first propose methods that take Linked Data and inductively create a highly expressive ontology[2] describing the data. By further formalizing the domain-specific knowledge, we are also able to bring problems which influence the accuracy dimension into the consistency dimension. This is advantageous since logical problems causing contradictions in the data are easier to detect by using logical methods than general discrepancies from real-world facts. The inductive generation leads to an ontology that is conform with the majority of the data contained in the dataset so that logical errors indicate data which behaves differently. Such deviations can show that the violating data is erroneous. In case the deviation is not caused by a problem in the violating data, the learned ontology is potentially wrong. Due to its foundation in the patterns found in the majority of data, a wrongly learned ontology also provides additional benefit since its errors indicate errors in the majority of data entities.

Moreover, this upstream formalization step allows us to cover the additional aspect of supporting a longer-term correctness of the dataset. Reaching correctness of a dataset at one point in time does not guarantee correctness for a longer period since, like data in the World Wide Web, Linked Data datasets can change rapidly and the increasing creation of data in collaborative manners further boosts the potential change frequency. In particular for the collaborative creation, one promising way of assuring more consistent correctness of data is the provision of adequate documentation regarding the data structure and specifics. Having an ontology specifying a dataset's semantics provides us synergies in this respect. Though their ac-

[2] In this work, we use the notions of *ontology* and *schema* interchangeably.

tual purpose in the Semantic Web differs from the purpose of schemas in the area of relational databases, which provide the general framework of expressing data within the database, well-specified ontologies can also serve as a documentation of a dataset. By summarizing the commonly occurring types of entities in a dataset and typical constructs for expressing relations between those entities, ontologies can provide a single reference point. Thus, if a dataset is accompanied by a corresponding ontology, this reduces the effort to get an overview of the dataset which in turn helps to maintain or extend the data in a way that it stays consistent and at high quality. In this case, the expressivity of ontologies, which is exceeding those of relational schemas by far, proves to be beneficial since it allows to formalize even more complex structures. Furthermore, the generation of expressive ontologies provides additional value on its own since it can be used for performing inference on the data bringing Linked Data one step further towards the overall idea of the Semantic Web.

1.1 Research Questions

As already sketched in the preceding motivation, we are investigating error detection methods for Linked Data, concentrating on errors in the logical structure and links of a dataset as well as on the correctness of numerical values contained in the dataset. For the first aspect we employ ontologies that describe the datasets to detect potential errors by looking for logical contradiction occurring when considering both ontology and data. Although many Linked Data datasets are accompanied by an ontology, these are typically very lightweight. In particular, they most often do not contain those kinds of axioms that are required to cause contradictions like negation. Thus, we cannot directly use these ontologies. Instead, we first have to enrich them with the required axiom types. For this purpose, we first discuss and investigate the following research question.

(RQ1) Which types of constructs can be learned from given instance data?

When learning axioms automatically, the achieved correctness can be assumed to be lower than for manually engineered axioms. Thus, the probability of introducing logical errors into the learned ontology is high and to draw the most advantage of ontologies, such errors should be recognized and fixed. Potentially, the inclusion of humans into this process is also worthwhile since this carries the promise of better results. This is particularly important when using the ontologies for documentation purposes because in this case high quality ontologies are even more desirable to actually reach the goal of guiding collaborators into producing high quality data. Learned ontologies of high expressivity are a special case since they do not follow the same paradigms as manually created ones. Thus, there is further investigation required on how to detect and handle errors in such ontologies properly.

(RQ2) What are suitable ways to debug and repair highly expressive, learned ontologies? Which of these ways qualifies best for including humans into the process?

Up until this point, we concentrated on enriching or learning the ontology and ensuring its quality which is particularly important when using the ontology for documentation purposes. As described above, we also apply the ontology for the actual error detection in the dataset which leads us to the next question.

(RQ3) Are learned ontologies suited for being used to detect errors in Linked Data datasets?

In this context, the influence of the ontologies' coherence, i.e., the satisfiability of all contained classes and properties, on the error detection in Linked Data datasets is highly interesting given our previous investigations on the debugging of the learned ontologies.

(RQ4) How does the coherence of learned ontologies influence the results when using them for detecting errors in Linked Data?

Finally, we also examine another level of correctness of Linked Data. Even though links in Linked Data are a fundamental building block, literal data also carries important information. This data can also suffer from errors, e.g., caused by parsing problems during the extraction from a source dataset. Thus, for making Linked Data more useful, the correctness of literal values is also important. In our work, we concentrate on the correctness of numerical values.

(RQ5) How can we detect errors in numerical value of a Linked Data dataset?

1.2 Reader's Guide

In this work, we roughly follow the structure already outlined in the research questions. After completing this motivation, we first introduce the basic notions required for the remainder of this work in Chapter 2. This includes the definition of Description Logic as foundation of the semantics in the Semantic Web as well as a short description of the Web Ontology Language OWL used for representing expressive ontologies. The introduction of Semantic Web-specific notions is completed by overviews on the Resource Description Framework (RDF), the idea of Linked Data and DBpedia as one of the most central dataset in Linked Data.

This is followed by the first part of this work which deals with inductively learning expressive ontologies. It is preluded by considerations on the possibilities to learn OWL data from instance data in Chapter 3 which also introduces the basics for learning ontologies from instance data: Association rule mining and statistical schema induction. In the following Chapter 4 we describe appropriate related work. Then, in Chapter 5, we examine different inductive approaches for

generating class disjointness from the instances contained in Linked Data datasets. We introduce a correlation-based method and two strategies using the concept of association rule mining and experimentally evaluate these methods. For this evaluation, we manually craft a class disjointness gold standard for the DBpedia 3.7 ontology on whose creation we give further details. Moreover, we compare one of the inductive approaches to a state-of-the-art supervised method for learning class disjointness.

In Chapter 6, we are again picking up association rule mining and propose ways of generating a number of additional axioms types. In particular, we cover the property axioms introduced with the second version of the OWL ontology language. The generated axioms are evaluated two-fold: by ontology engineering experts and by laymen using crowd-sourcing. This also enables us to get some insights into the complexity of working with the newly expressable axiom types.

To tackle the problem of detecting errors in learned ontologies, we rely on explanations for the problems existing in an ontology so that we can fix these. Currently existing approaches have difficulties in providing us explanations for our expressive learned ontologies. Thus, in Chapter 7, we implement the generation of explanations using rules specifically crafted to support the expressivity and characteristics of ontologies created by our learning approaches.

Based on these explanations, we explore different ways of finding sets of axioms whose removal makes the ontology coherent again in Chapter 8. For this purpose, we consider different types of greedy methods as well as approaches based on Markov Logic.

In Chapter 9, we perform the final steps regarding the correctness of links in Linked Data. We perform experiments to detect contradictions arising from the combination of the different enriched ontologies and the actual dataset. Then, the results are manually evaluated regarding whether they point to actual errors in the dataset. This allows us to assess how well the detection of errors works based on learned ontologies.

Afterwards,we shift our focus from logical errors in Linked Data over to erroneous literal values in the dataset and propose an approach for efficiently detecting errors in numerical values. For this purpose, we use outlier detection whose foundations are introduced in Chapter 10. Afterwards, in Chapter 11, we introduce ways of discovering relevant subsets of values to perform outlier detection on as well as a way of avoiding problems otherwise introduced by extraordinary but correct values.

Finally, in Chapter 12, we conclude this work by summarizing the main findings, particularly regarding the research questions and also point out directions for future work.

Chapter 2

Foundations

In this chapter, we give an introduction into the basics required for understanding all approaches and experiments presented later in this work. First, we present the basic notions of description logics, which are the formal underpinnings of knowledge representation and inference in the Semantic Web, and of ontologies which are relying on Description Logics. Afterwards, we give a short introduction into RDF as well as into the principles of Linked Data. For the latter, we especially highlight the DBpedia dataset.

2.1 Description Logics and Ontologies

Description Logics provide the formal background for the most popular knowledge representation languages used in the Semantic Web. Even though they are less expressive than first-order logic, description logics are preferred in these scenarios because there are efficient reasoning procedures which are important for the usage in real-world use cases. In the context of the Semantic Web, description logics of different expressivity are used. One of the most basic description logics is called $\mathcal{ALC}$. In the following, we will introduce $\mathcal{ALC}$ as an example of a description logic and later-on extend this definition towards the additional expressivity provided by, e.g., OWL 2. Our definitions of syntax and semantics are based on those given by Baader [8] and Baader et al. [9].

The fundamental building blocks available in Description Logics are sets of names for atomic *classes*[1] and atomic *properties*.[2] The relevant description logic then provides a set of constructors that are used to define more complex descriptions recursively. For $\mathcal{ALC}$ this leads to the following definition of class descriptions.

Definition 1. Let N_C be a set of class names and N_P a set of property names. Furthermore, let there be the atomic classes $\top$ and $\bot$ called top and bottom class

[1] Classes are commonly also referred to as concepts.

[2] Sometimes also called roles or relations.

respectively. The set of possible class descriptions in $\mathcal{ALC}$ is defined as follows:

- all class names in N_C, $\top$ and $\bot$ are class descriptions (called *atomic classes*),
- if C and D are class descriptions then $C \sqcup D$ (called *disjunction*), $C \sqcap D$ (called *conjunction*) and $\neg C$ (called *complement*) are class descriptions, and
- if C is a class description and P is a property name from N_P then $\exists P.C$ (called *existential restriction*) and $\forall P.C$ (called *universal restriction*) are class descriptions.

We call all class descriptions that are not atomic classes *complex class descriptions*.

By using this definition, we are able to generate all class descriptions possible in $\mathcal{ALC}$. The semantics of these class descriptions are defined using model-theoretic semantics by means of an interpretation.

Definition 2. An *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of the *domain* $\Delta^{\mathcal{I}}$ which is a non-empty set and an *interpretation function* $\cdot^{\mathcal{I}}$ which maps each class name $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each property name $P \in N_P$ to a set $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

For non-atomic class descriptions, we define:

- for class descriptions C and D:
  $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ and $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ and
- for a class description C and a property name $P \in N_P$:
  $(\exists P.C)^{\mathcal{I}} = \{i \in \Delta^{\mathcal{I}} | \exists b \in C^{\mathcal{I}} : (i, b) \in P^{\mathcal{I}}\}$ resp.
  $(\forall P.C)^{\mathcal{I}} = \{i \in \Delta^{\mathcal{I}} | \forall b \in \Delta^{\mathcal{I}} : (i, b) \in P^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\}$

Furthermore, the top class $\top$ is mapped to $\Delta^{\mathcal{I}}$ while the bottom class $\bot$ is equivalent to $\neg\top$.

This basic description logic can already represent a great number of different classes. Some example class descriptions are shown in the following.

Example 1. Let the named classes `Human`, `Dog`, `Cat` and the properties `isPetOf`, `hasPet` and `isMarriedTo` be given.

- $\texttt{Dog} \sqcap \exists \texttt{isPetOf}.\texttt{Human}$: Every dog which is the pet of a human.
- $\texttt{Human} \sqcap \exists \texttt{hasPet}.\texttt{Dog} \sqcap \neg\exists \texttt{isMarriedTo}.\top$: All humans which have a dog as a pet and are not married.

There are other description logics which partly extend the capabilities provided by $\mathcal{ALC}$ and some which only share parts of the expressivity $\mathcal{ALC}$ provides. The

most important ones for our work are based on $\mathcal{ALC}$ and only extend it by providing additional class and property constructors. Some possible extensions are presented in the following.

Qualified number restrictions provide a way to express classes of instances that are used with a specific property at least or at most a given number of times. They are represented by $\geq n\, P.C$ (at-least restriction) and $\leq n\, P.C$ (at-most restriction) where $n \geq 0$ is a non-negative integer, $P \in N_P$ a role name and C a class description. The model-theoretic interpretation of qualified number restrictions is

$$(\geq n\, P.C)^{\mathcal{I}} := \left\{ i \in \Delta^{\mathcal{I}} \mid \#\{j \in P^{\mathcal{I}} | (i,j) \in P^{\mathcal{I}}\} \geq n \right\}$$

and

$$(\leq n\, P.C)^{\mathcal{I}} := \left\{ i \in \Delta^{\mathcal{I}} \mid \#\{j \in P^{\mathcal{I}} | (i,j) \in P^{\mathcal{I}}\} \leq n \right\}$$

respectively, where $\#S$ is the cardinality of the set S. There also are so-called *number restrictions* which are equivalent to the qualified ones but only allow $\top$ to be used at the concept position, often written as $\geq n\, P$ or $\leq n\, P$.

Similar to the constructors provided for classes, there is a number of constructors which allow the definition of more complex property descriptions recursively based on atomic property descriptions. These are not included in the $\mathcal{ALC}$ description logic but only available in more expressive DL variants.

Definition 3. Let N_P be a set of property names. The set of possible property descriptions is defined as follows:

- all property names in N_P (called *atomic properties*) are property descriptions,
- U is a property description called the *universal property*,
- if P is a property description then $\neg P$ (called *property complement*) is a property description,
- if P is a property description, P^{-1} is a property description called *inverse property to* P,
- if P and R are property descriptions, $P \circ R$ is a property description called *property chain*

As for class descriptions, we define the semantics of property descriptions based on set theory by means of an interpretation function.

Definition 4. Let an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ as defined in Definition 2 be given. For non-atomic property descriptions, we extend the interpretation function $\cdot^{\mathcal{I}}$ as follows:

- $(U)^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$,

- for property description P and R:
  $(\neg P)^{\mathcal{I}} = (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus P^{\mathcal{I}}$
- for a property description R:
  $(P^{-1})^{\mathcal{I}} = \{(b, a) \mid (a, b) \in P^{\mathcal{I}}\}$
- for property descriptions P and R:
  $(P \circ R)^{\mathcal{I}} = \{(a, c) \mid a, c \in \Delta^{\mathcal{I}} \wedge \exists b \in \Delta^{\mathcal{I}} : (a, b) \in P^{\mathcal{I}} \wedge (b, c) \in R^{\mathcal{I}}\}$

Complementary to the complex class descriptions, we can use complex property descriptions to cover various real-world relations between entities based on other already existing property descriptions like in the following example.

Example 2. Let property names `hasParent` and `isBrotherOf` be given.

- Starting from the `hasParent` property, we can express that someone is a parent of someone by using the inverse $\texttt{hasParent}^{-1}$ which is the equivalent to a property which could be named `isParentOf`.
- Using property chains, we can express the property of having an uncle as $\texttt{hasParent} \circ \texttt{isBrotherOf}$.

Given the different constructors of Description Logics as described before, it is possible to define the actual notion of ontologies. Ontologies are commonly divided into two parts: TBox and ABox. The TBox contains the terminological knowledge, i.e., it describes the different classes, properties and their relations between each other. The relations between different classes or different properties are defined by means of axioms as introduced in the following definition.

Definition 5. Given class descriptions C and D and property descriptions P and Q, an *axiom* is given by

- $C \equiv D$ and $C \sqsubseteq D$ (*class equivalence axiom* and *class inclusion axiom*) or
- $P \equiv Q$ and $P \sqsubseteq Q$ (*property equivalence axiom* and *property inclusion axiom*)

A finite set of axioms is called *TBox*.

In contrast to the TBox, the ABox as second part of an ontology contains the assertional knowledge about named instances in the ontology.

Definition 6. Let a class description C, a property description P and $a, b \in N_I$ be given. An *individual assertion* is either $C(a)$ or $R(a, b)$. An *ABox* is a finite set of individual assertions.

An interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ maps each instance name $a \in N_I$ to an element of the domain $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

An *ontology* is fully defined by its TBox $\mathcal{T}$ and its ABox $\mathcal{A}$.

Since we now have all notions defined which are required for building an ontology, we need to define the semantics of an ontology based on the contents of its TBox and ABox. This is done on a per axiom level and leads to the definition of *models* of ontologies which are the main notion regarding ontology semantics.

Definition 7. Let C, D be class descriptions, P, Q property descriptions and $a, b \in N_I$ instance names. An interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies* an axiom

- $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$,
- $C \equiv D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$,
- $P \sqsubseteq Q$ iff $P^{\mathcal{I}} \subseteq Q^{\mathcal{I}}$ and
- $P \equiv Q$ iff $P^{\mathcal{I}} = Q^{\mathcal{I}}$.

Furthermore, an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies the individual assertions

- $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and
- $P(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$

An interpretation is called a *model of a TBox* $\mathcal{T}$ if it satisfies each axioms in $\mathcal{T}$. An interpretation is a model of an ABox $\mathcal{A}$ if it satisfies all individual assertions of $\mathcal{A}$. An interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of an ontology $\mathcal{O}$ with TBox $\mathcal{T}$ and ABox $\mathcal{A}$ if it is a model of both $\mathcal{T}$ and $\mathcal{A}$.

It is obviously possible to define ontologies that have no models. For example, an ontology only consisting of the axioms $A \sqsubseteq \neg B$ and $A \sqsubseteq B$ as well as the individual assertion $A(x)$ cannot have a model. This is easily visible from the fact that the only interpretation which satisfies both axioms would assign the empty set to the classes A and B, i.e., $A^{\mathcal{I}} = \emptyset$ and $B^{\mathcal{I}} = \emptyset$. However, this interpretation would not satisfy the individual assertion since there is no element in A to which the interpretation function could map x. Thus, there cannot be a model for this ontology. This is formalized by the following definition.

Definition 8. A TBox $\mathcal{T}$ is *satisfiable* iff there is a model for $\mathcal{T}$.

An ontology $\mathcal{O}$ is *satisfiable* or *consistent* iff there is a model for $\mathcal{O}$ otherwise we call $\mathcal{O}$ *unsatisfiable* or *inconsistent*.

A class C in a TBox $\mathcal{T}$ is called *satisfiable* if there exists a model $\mathcal{I}$ of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. Likewise, a property P in a TBox $\mathcal{T}$ is called *satisfiable* if there exists a model $\mathcal{I}$ of $\mathcal{T}$ with $P^{\mathcal{I}} \neq \emptyset$.

An ontology is *incoherent* if it contains unsatisfiable classes or properties.

A combination of these class and property constructors already allows us to express many characteristics. However, for properties there are some additional characteristics that can be assigned using so-called *property assertions*. We model these statements according to Horrocks et al. [52].

Definition 9. For property names $P, R \in N_P$ with $P, R \neq U$, we call the following assertions *property assertions*. For each property assertion, we provide the restrictions they impose for an interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and for all $x, y, z \in \Delta^{\mathcal{I}}$

- *Property Symmetry*: if $\mathrm{Sym}(P)$ then $(x, y) \in P^{\mathcal{I}}$ implies $(y, x) \in P^{\mathcal{I}}$,
- *Property Asymmetry*: if $\mathrm{Asym}(P)$ then $(x, y) \in P^{\mathcal{I}}$ implies $(y, x) \notin P^{\mathcal{I}}$,
- *Property Transitivity*: if $\mathrm{Tra}(P)$ then $(x, y) \in P^{\mathcal{I}} \wedge (y, z) \in P^{\mathcal{I}}$ implies $(x, z) \in P^{\mathcal{I}}$,
- *Property Reflexivity*: if $\mathrm{Ref}(P)$ then $(x, x) \in P^{\mathcal{I}}$,
- *Property Irreflexivity*: if $\mathrm{Irr}(P)$ then $(x, x) \notin P^{\mathcal{I}}$,
- *Property Disjointness*: if $\mathrm{Dis}(P, R)$ then $(x, y) \in P^{\mathcal{I}}$ implies $(x, y) \notin R^{\mathcal{I}}$.
- *Property Functionality*: if $\mathrm{Fun}(P)$ then $(x, y) \in P^{\mathcal{I}} \wedge (x, z) \in P^{\mathcal{I}}$ implies $y = z$
- *Property Inverse Functionality*: if $\mathrm{InvFun}(P)$ then $(y, x) \in P^{\mathcal{I}} \wedge (z, x) \in P^{\mathcal{I}}$ implies $y = z$

Property symmetry, transitivity and disjointness are also expressible by applying the property constructors introduced before, i.e., $\mathrm{Sym}(P)$ is equivalent to $P \sqsubseteq P^{-1}$, $\mathrm{Tra}(P)$ to $P \circ P \sqsubseteq P$ and $\mathrm{Dis}(P, R)$ can also be expressed by $P \sqsubseteq \neg R$. For these statements, the corresponding property assertions have been introduced to constrain the expressivity to certain patterns. However, in this work, we merely list the property assertion syntax for the matter of completeness.

One reason for using description logic as foundation of ontologies is that it provides a way of formally deducing implicit knowledge from a knowledge base. This process is called *reasoning* or *inference* and is formally defined on the level of assertions as follows.

Definition 10. Given an ontology $\mathcal{O}$, an assertion α is entailed by $\mathcal{O}$ (written as $\mathcal{O} \models \alpha$) if every model of $\mathcal{O}$ also satisfies α.

Thus, it can be checked whether a given assertion can be inferred from the ontology. To perform inference on the level of terminological axioms, the query is reduced to the assertional case. For example, to check whether a class C is subsumed by a class D, i.e., $C \sqsubseteq D$, the entailment of $(C \sqsubseteq D)(x)$ is checked for an instance x not yet contained in the ontology's ABox.

To conclude this short introduction into Description Logic, we demonstrate the constructs defined above in the following example ontology.

Example 3. We first introduce the TBox of the ontology as follows.

$$\texttt{Man} \sqsubseteq \texttt{Human} \tag{2.1}$$
$$\texttt{Woman} \sqsubseteq \texttt{Human} \tag{2.2}$$
$$\texttt{Dog} \sqsubseteq \neg\texttt{Human} \tag{2.3}$$
$$\top \sqsubseteq \forall \texttt{hasSister}.\texttt{Woman} \tag{2.4}$$
$$\exists \texttt{knows}.\top \sqsubseteq \texttt{Human} \tag{2.5}$$
$$\texttt{hasSister} \sqsubseteq \texttt{hasSibling} \tag{2.6}$$
$$\texttt{hasChild} \sqsubseteq \neg\texttt{hasParent} \tag{2.7}$$
$$\texttt{hasChild} \equiv \texttt{hasParent}^{-1} \tag{2.8}$$
$$\texttt{hasParent} \circ \texttt{hasDaughter} \sqsubseteq \texttt{hasSister} \tag{2.9}$$
$$\mathrm{Tra}(\texttt{hasAncestor}) \tag{2.10}$$
$$\mathrm{Ref}(\texttt{knows}) \tag{2.11}$$
$$\mathrm{Irr}(\texttt{hasParent}) \tag{2.12}$$

Here, we state that each man as well as each woman is a human (2.1, 2.2) but that dogs are not humans (2.3). The next two axioms define that the relation of having a sister can only hold to women (2.4) and that only humans can know someone (2.5). After these class-related axioms, we introduce a number of property axioms. First, we say that having a sister means having a sibling (2.6) and that children cannot be parents of their parents (2.7). Rather, we say that when someone has a child, this person is the child's parent (2.8). In 2.9, the relation of being a sister is defined to hold to all people which are daughters of the same parents. Finally, we express that the ancestor of a person's ancestor is also his ancestor (2.10), everyone knows himself (2.11) and that nobody can be his own parent (2.12).

All classes provided here are satisfiable as are all properties. Nevertheless, the axiom $\texttt{SuperDog} \sqsubseteq \texttt{Dog} \sqcap \texttt{Human}$ would introduce the unsatisfiable class SuperDog and thus render the ontology incoherent.

We could extend the ABox of this ontology by assertions like $\texttt{Man}(\texttt{Peter})$, $\texttt{knows}(\texttt{Peter})$ or $\texttt{hasSister}(\texttt{Peter}, \texttt{Andrea})$ which would all adhere to the defined axioms. Adding the individual assertion

$$\texttt{hasChild}(\texttt{MartyMcFly}, \texttt{MartyMcFly})$$

which shows a potential problem for time travelers like Marty McFly,[3] however, would lead to an inconsistent ontology since axiom 2.8 would imply that

$$\texttt{hasParent}(\texttt{MartyMcFly}, \texttt{MartyMcFly})$$

also holds and thus 2.7 could not be satisfied.

[3] http://en.wikipedia.org/wiki/Marty_McFly

Description logics are categorized by their expressivity, i.e., by the different axiom types they provide and the restrictions to the syntactic structure they require for specific axioms and constructors. To identify these different logics, there is a naming schema based on letters each of which represents a specific capability of the description logic as shown in Table 2.1. This table has been compiled from Baader et al. [11], Horrocks and Sattler [53] and Horrocks et al. [54]. According to this categorization, the Web Ontology Language OWL DL [70] is based on the Description Logic $\mathcal{SHOIN}(\mathrm{D})$ and on $\mathcal{SROIQ}(\mathrm{D})$ in the more current version OWL 2 [51].

Table 2.1: Naming convention for description logics used to describe logic capabilities. A denotes an atomic class, C, D denote potentially complex class descriptions, P, R denote properties.

Letter	Supported	Examples
$\mathcal{AL}$	atomic concepts, atomic negation, intersection, value restriction, limited existential quantification	A, $\neg A$, $A \sqcap C$, $\forall R.C$, $\exists R.\top$
$\mathcal{S}$	$\mathcal{ALC}$ with transitive roles	$\mathrm{Tra}(R)$
$\mathcal{U}$	Union of concepts	$C \sqcup D$
$\mathcal{E}$	Full existential quantification	$\exists R.C$
$\mathcal{N}$	Number restrictions	$\geq nR$, $\leq nR$
$\mathcal{Q}$	Qualifying number restrictions	$\geq nR.C$, $\leq nR.C$
$\mathcal{C}$	Negation of arbitrary concepts	$\neg C$
$\mathcal{I}$	Inverse property	R^{-1}
$\mathcal{H}$	Property hierarchy	$R \sqsubseteq P$
$\mathcal{R}$	Limited complex (regular) role inclusion, reflexivity, irreflexivity, role disjointness	$P \circ R \sqsubseteq R$, $\mathrm{Ref}(R)$, $\mathrm{Irr}(R)$, $\mathrm{Dis}(P, R)$ resp. $P \sqsubseteq \neg R$
$\mathcal{O}$	Nominals	$\{a, b, c\}$
(D)	Data extensions	$R(a,$ "Name")

2.2 RDF and Linked Data

Given the foundations provided by description logic, the actual base of the Semantic Web is formed by its standards as defined by the *World Wide Web Consortium* (W3C) and others. In this section, we describe the Resource Description Framework, the idea of Linked Data and highlight the latter by presenting the DBpedia dataset.

2.2.1 Resource Description Framework

The *Resource Description Framework* (RDF) [86] is the most regularly used way of representing data in the Semantic Web. In general, it is a framework which provides capabilities to describe information about resources. The notion of resources is not limited to specific objects, instead, RDF is explicitly made for being used on all kind of resources, e.g., documents but also people or abstract objects. Information in RDF is represented by means of a graph structure whose most basic building part are triples of the form $(subject, predicate, object)$ in this context regularly written as

$$< subject > < predicate > < object > \tag{2.13}$$

Such a triple can be used to state a relationship of the type $predicate$ to hold between the resources $subject$ and $object$. This relationship, which is called property in RDF, is defined directionally from $subject$ to $object$.

Hence, each of these statements forms a directed graph consisting of two nodes, formed by subject and object, and a typed directed edge connecting both nodes. By adding more triples that have the same resource as subject, more outgoing edges are added to the subject node which results in the categorization of RDF being a graph-based model. Through extending the set of statements with statements about resources used somewhere else as object a larger, connected graph arises as shown in the upper left part of Figure 2.1. Nevertheless, there is no need for defining fully connected graphs.

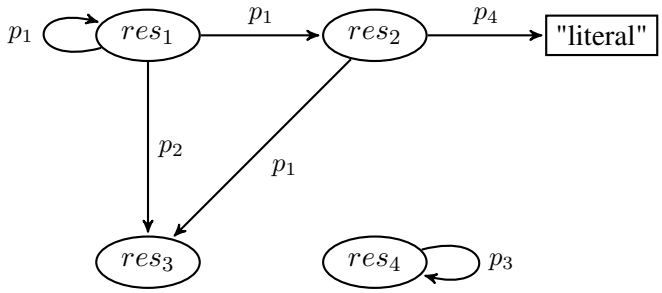


Figure 2.1: Graph structure formed by RDF statements

There are three general data types which can occur in an RDF triple. The first type is an *International Resource Identifier* (IRI) which can be used in all three positions: subject, object and predicate. IRIs are used throughout RDF and the Semantic Web in general to uniquely identify resources. Being a generalization of the Unified Resource Identifier (URI), URIs as `http://example.com/resource1` are valid IRIs as are all other strings satisfying the generic URI syntax like `tel:+491234567`. RDF also allows relative IRIs to be used for identification of resources as long as it is clearly defined on which base IRI the relative one has to be resolved. *Literals* as a second data type are only allowed at the object position of a triple since they present raw data such as numbers or strings that lead to a leaf node in the graph. They can be accompanied by a language tag which allows to define the language they are specified in. The third type of nodes are so-called *blank nodes* which are not identified by global name but merely take the role of variables used to make statements about something not further specified in an RDF graph without explicitly naming it. As the name implies, they can only be used as nodes and thus are allowed at subject and object position.

There are several ways defined regarding how to serialize the graph formed by using RDF into textual formats. This includes serializations into the XML format which is the format most commonly used in practical applications. Since this XML serialization introduces large amounts of textual redundancy, we will not use it in this work. Instead, when having to state RDF graphs textually, we will resort to the Turtle format [13] which uses triples similar to the one shown in 2.13 for textual representation combined with some syntactical extensions for shortening the result. A serialization in Turtle of a graph structure as shown in Figure 2.1 is given in Listing 2.1.

Listing 2.1: Turtle representation of graph structure shown in Figure 2.1

```
@base <http://example.com/data#> .
@prefix preds: <http://example.com/predicates#> .
<#res1> preds:p1 <#res2>.
<#res1> preds:p1 <#res1>.
<#res2> preds:p1 <#res3>.
<#res1> preds:p2 <#res3>.
<#res4> preds:p3 <#res4>.
<#res2> preds:p4 "literal".
```

2.2.2 Linked Data

Based on the general idea of the Semantic Web and the RDF format, Tim Berners-Lee coined the notion of *Linked Data* in 2006 [14]. He envisioned a network of data repositories which are interlinked by means of typed links like RDF predicates referring from resources in one repository to those from another repository. To foster this vision, Berners-Lee outlined four rules, known as *Linked Data Principles*

which qualify data as Linked Data.

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards.
4. Include links to other URIs, so that they can discover more things.

All of these principles aim at having data available in the Semantic Web for automatic consumption very similar to data in the Web available for human consumption. They promise the accessibility and discoverability of the data by means of Web technologies. Consequently, Linked Data is especially positioned in contrast to so-called data silos where the information stored by some data provider is only available by means of (proprietary) Web APIs without any links to data of other providers.

The success of the idea of Linked Data is especially visible when having a look at *Linked Open Data* (LOD) cloud as shown in Figure 2.2.[4] Though containing only datasets released under an open license, it consists of almost 300 datasets interlinked to at least one other datasets in the LOD cloud.

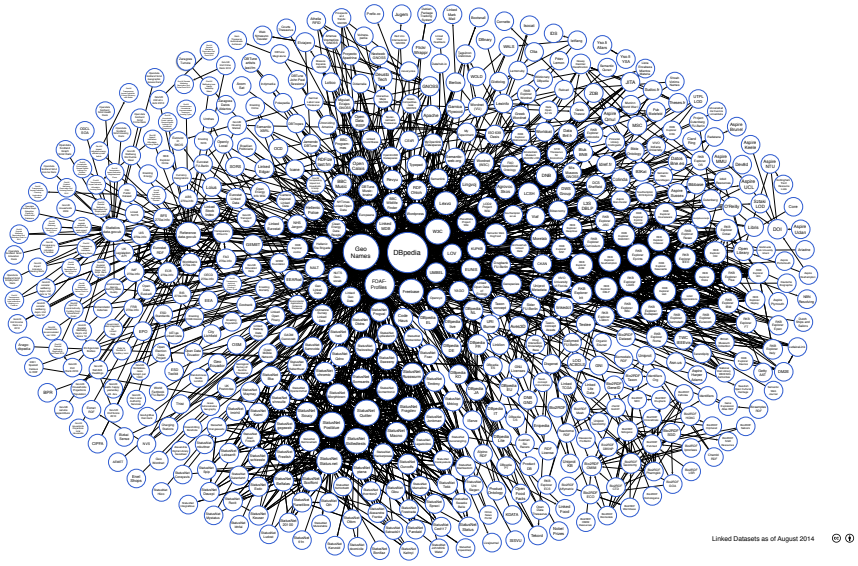


Figure 2.2: Linked Open Data cloud as of August 2014

The Linked Open Data cloud encompasses many different fields reaching from bibliographic and geographical data over medical datasets to census data and other

[4]Linking Open Data cloud diagram 2014, by Max Schmachtenberg, Christian Bizer, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/

datasets provided by governments in the process of increasing transparency. This particularly shows that the idea of Linked Data not only compelled computer scientists but also prople from other areas in which large amounts of data are managed. Thus, the LOD cloud covers a wide range of different domains.

To access these datasets, there are three ways commonly provided by the repositories constituting the LOD cloud. The technically simplest one is the provision as RDF dumps which are files containing the triples of the dataset's serialized RDF graph. A second option for publication is to provide an interface based on *SPARQL* (SPARQL Protocol and RDF Query Language) [48]. This querying language defines constructs allowing to traverse the RDF graph for specifying and retrieving parts of its data similar to what the SQL query language does for relational data models. The third option is to allow RDF crawling of the data, i.e., the repository provides descriptions for all its entities. For these descriptions, there is a central entry point from which the full dataset can be traversed by successively following internal links between entities until all entities have been visited. It is worth to note that for the former two publishing options, RDF dumps and SPARQL endpoint, it is also required to provide dereferencable URIs to actually conform to the Linked Data principles.

2.2.3 DBpedia

In the diagram of the LOD cloud shown in Figure 2.2, the DBpedia [66] dataset stands out for being central and one of the most highly interconnected datasets regarding the number of other datasets linked to it. Like many of the major LOD datasets, DBpedia is not manually created and maintained in its Linked Data variant. Instead, it is extracted from the Wikipedia infoboxes which are the boxes visible on the right side of many Wikipedia articles.

Infoboxes are created from so-called infobox templates that are defined for specific types of entities commonly described in Wikipedia. The templates define a number of attributes which are typically used for the type of entity. Apart from that, the template also determine the information's representation on the final Wikipedia article page to foster a more unified look of Wikipedia articles. For example, the code shown in Listing 2.2 defines some of the most relevant information about Tim Berners-Lee. It uses the infobox template for entities of the type *person* which provides fields to enter information like the name, birth data, parents and occupation but also less often used fields like honorific suffix whose use in an instantiation of a template is optional.

Included in the Wikipedia article's source code, this infobox code is then rendered as shown in Figure 2.3 where, for example, the given image is used to depict the person and the provided birth data is employed to compute Tim Berners-Lee's current age while the information about birth place and the birth name are used to enrich it.

An important aspect of infoboxes is that the templates do not enforce certain data types for their entries thus the authors of articles are free to provide all data

Listing 2.2: Excerpt of the infobox code for the Wikipedia page of Tim Berners-Lee

```
{{Infobox person
| honorific_prefix =
| name        = Sir Tim Berners-Lee
| honorific_suffix =
| image       = Tim Berners-Lee 2012.jpg
| caption     = Berners-Lee in 2012
| birth_name  = Timothy John Berners-Lee
| birth_date  = {{birth date and age|1955|6|8|df=y}}
| birth_place = London, England<br>United Kingdom
| nationality = British
| residence   = United Kingdom and United States
| alma_mater  = [[The Queen's College, Oxford]]
| occupation  = [[Computer scientist]]
| title       = Professor
| partner     = Rosemary Leith
| parents     = {{Plainlist|
* [[Conway Berners-Lee]]
* [[Mary Lee Woods]]
}}
| website     = {{Url|www.w3.org/People/Berners-Lee}}
}}
```

in arbitrary textual formats. Although there is a recommended format for some attributes, this format is only provided as a comment in the infobox template's definition without further (technical) implications arising from this. One explanation for not having strict typing for the infobox attributes is that this would make it harder for the authors of articles to provide certain kind of information if these deviate from the allowed format which would finally lead to a limited degree of adoption. Thus, this missing typing can be seen as one reason that infoboxes are widely deployed in Wikipedia. For example, in August 2013[5] the English Wikipedia contained approximately 4.4 million articles with a total number of 2.4 million infoboxes.[6]

In summary, infobox templates are drafts that provide possible characteristics of certain types of things while the actual infoboxes instantiate these drafts to provide data about a specific thing. This makes the infobox template an approach for defining an untyped semi-structured representation of information.

Starting from similar considerations, Auer and Lehmann [7] proposed to use the semi-structured content provided by the infoboxes to extract semantically enriched content to allow the querying of this data. This work finally led to DBpedia [66] which is using the infobox contents to create an RDF dataset. We will now shortly describe the approach that is currently used for this.

[5] http://en.wikipedia.org/w/index.php?title=Wikipedia:WikiProject_Infoboxes/Statistics&oldid=569281636

[6] http://stats.wikimedia.org/EN/TablesWikipediaEN.htm

In the first versions, the extraction routines were just applied to the infobox data available as dumps.[7] This basic approach, also called "generic approach" is still available and its result known as the "raw infobox dataset". For each article containing an infobox, a URI is derived from the URL of the article. This URI is used as an identifier for an RDF instance which then gets assigned the values contained in the infobox using RDF properties directly derived from the infobox' attribute names.

This approach is straightforward but suffers from some problems which finally led to the development of a more sophisticated approach as described by Bizer et al. [16]. As explained by them, the main problem of the approach lies in the fact that there is no central management which controls the creation of infoboxes but each subgroup of the Wikipedia community might have its own template to describe the same sort of things and, furthermore, attribute names can differ across different templates though they express the same information. Thus, the generic approach leads to a relatively noisy dataset for which it is hard to formulate queries that consider all relevant data due to the potentially large number of different properties and classes for the same information.

To solve these shortcomings, an OWL ontology called DBpedia ontology was created. This ontology contains a class hierarchy and properties to which the infobox templates and their properties can be mapped. Based on these mappings, an article with an infobox is assigned to the ontology class its infobox is mapped to and the attributes are assigned by means of their mapped ontology properties which also define the object or data type used for a property even though these are not enforced. Since unmapped classes and properties are not extracted by this mapping-based approach, there is a crowd-sourced possibility to extend the ontology by new classes, properties and several other axiom types which allows interested parties to improve the coverage. Furthermore, this possibility for extension allows to make the ontology more expressive so that it gets more useful for tasks apart from the extraction.

[7] The Wikipedia dumps are provided by the Wikimedia Foundation: `http://dumps.wikimedia.org/`

Sir Tim Berners-Lee

Berners-Lee in 2012

Born	Timothy John Berners-Lee 8 June 1955 (age 58)[1] London, England United Kingdom
Residence	United Kingdom and United States[2]
Nationality	British
Alma mater	The Queen's College, Oxford
Occupation	Computer scientist
Employer	World Wide Web Consortium University of Southampton Plessey MIT
Known for	Inventing the World Wide Web Holder of the 3Com Founders Chair at MIT's Computer Science and Artificial Intelligence Laboratory
Title	Professor
Religion	Unitarian (raised as Anglican)[3]
Partner(s)	Rosemary Leith
Parents	Conway Berners-Lee Mary Lee Woods
Awards	OM (2007) KBE (2004) OBE (1997) RDI (2009) FRS (2001) FREng[1]

Website

www.w3.org/People/Berners-Lee

Figure 2.3: Rendered infobox about Tim Berners-Lee

Part I

Learning Expressive Schemas

Chapter 3

Preliminaries

In this first part of our work, we introduce and evaluate methods for learning ontologies from instance data contained in a Linked Data dataset. As we already outlined in the motivation of our work, we do this as an intermediate step for detecting errors. Obviously, this learning step is not required to actually perform error detection later-on. For instance, it would be possible to apply approaches such as outlier detection, which we use with a different focus in Chapter 11, directly on the dataset to detect parts of the data that do not adhere to the typical patterns shown in the dataset. Such an approach could even be considered more efficient due to circumventing the preceding step of generating a corresponding ontology. Nevertheless, we consider this step worthwhile. First, the learned ontology formalizes the typical patterns in the dataset and thus shows which behavior is considered typical for the data. This allows to better explain why certain data was classified as being erroneous and hence gives more opportunities to include humans into the error detection process. Furthermore, given that all these patterns are explicitly stated in the ontology, it is possible to let humans assess the patterns regarding their correctness before using it to identify erroneous instance data.

Secondly, the learned ontology is useful on its own and not only in the context of detecting errors in data. Additional class disjointness or property axioms allow to employ inference techniques to deduce previously unstated information from the dataset and thus give applications more data to work with. For instance, when querying a dataset the inclusion of disjointness axioms in its ontology would allow to more precisely deliver the query answer since the additional disjointness axioms help to separate relevant from irrelevant information. Naturally, the quality of a learned ontology might not be high enough for directly using it in such a scenario. Nonetheless, the ontology can act as a starting point for further manual corrections and extensions which make it more suited for specific inferencing scenarios.

Lastly, another advantage is the additional documentation provided by an ontology. As we already described in Chapter 1, the existence of an ontology helps new contributors to get an overview about the dataset more quickly. For example, an ontology containing a set of classes shows which types of instances the dataset

typically contains. From an ontology enriched with axiom types such as disjointness, contributors can derive additional information about the usage of the classes and the delimitation of the different classes to each other.

In the following, we first detail on some general considerations about learning OWL ontologies from instance data. Afterwards, we introduce the foundations of association rule mining and statistical schema induction.

3.1 Learning from Instance Data

In this section, we have a look on the general possibility and limitations of learning different OWL constructs from given instance data. More specifically, we are going to limit our explanation to schema-level axioms supported in OWL 2. When learning schema-level information from data, the idea is to detect patterns in the data and generalize those to a schema axiom valid for the whole dataset and, to gain a more generally useful ontology, the whole domain of the dataset. This leads to the main prerequisite: The dataset has to contain a sufficiently high number of instances which are also assigned to classes. The lower the number of instances in the dataset and the less extensive their assignment to classes, the higher the probability that its recognizable patterns are misleading and do not generalize well for the whole domain. In contrary, larger numbers of instances with proper class assignments can be expected to show the behavior of the full domain better. Obviously, if learning property axioms is desired, the dataset has to also expose a certain level of property assignments either between two instances (for object property axioms) or between an instance and a data value (for data property axioms). Since we are considering typical patterns of data, a certain amount of erroneous assignments regarding both classes and properties is tolerable without significant influence on the detected patterns.

For our following considerations, we are going to assume that a dataset is given that fulfills these prerequisites. Thus, we deal with the ideal scenario for learning a schema from a dataset. Nevertheless, there are two factors which have to be considered when learning ontological knowledge for OWL from instance data. The first one is the so-called *open-world assumption* (OWA) on which all OWL variants are based. This assumption implies that each statement might be true unless it can be proven otherwise. By implementing the OWA instead of a *closed-world assumption* (CWA) as often used in databases, OWL allows to better handle incomplete data and decentrally extend the data. Since leaving out a certain statement in an OWL dataset does not state anything about its truth value, it is easily possible to provide information about this statement in a different dataset without leading to contradictions between both datasets. In a CWA scenario however, leaving out a certain statement would assign the truth value ”false” to it and thus limit the possibilities to extend the dataset. For the case of learning an ontology from instance data, this especially has impact regarding the learning axioms that are connected with negative information. Though it is possible in OWL to explicitly state that a

certain fact is not true using assertions to complement classes or negative property assertions, this is rarely used in datasets. However, when trying to learn certain axioms from the instance data, it is required to rely on negative information in the dataset. Due to the low number of explicitly given negative statements, the only possibility to get this information is to use the absence of a statement as an evidence for it not being true. Thus, this is similar to applying a CWA to the considered dataset and might lead to learning wrong axioms where the statement would be actually true but is just left out of the dataset. However, in our work presented here, we argue that the overall impact is limited. Since we are looking into approaches to enrich large Linked Data datasets with additional axioms, we assume that the volume of the available data compensates for many errors introduced by applying a limited closed-world view to the actual open-world scenario because a larger dataset makes it more probable that a relevant statement is explicitly stated in the dataset at least once.

The second characteristic that influences the learning from instance data is the fact that OWL and OWL 2 do not implement the *unique name assumption* (UNA). This means that, given two different entity names, you cannot conclude that they point to different entities just based on the name difference. The rationale for not implementing the UNA in OWL is the fact that in a decentralized scenario it cannot be guaranteed that there is one unique name for each real-world entity across all different datasets. However, OWL provides the additional assertion `owl:differentFrom` to explicitly state that two names point to different entities which can be used when such a behavior is desired. For learning schema information from instance data, this missing UNA has some implications. In general it may lead to a generation of wrong axioms or not generating correct axioms. If one entity is addressed by two different names and there are no assertions provided that help to recognize this fact, the assertions made for this entity might be considered for each of the different names individually and thus make a pattern less clear. This could hide patterns relevant for recognizing certain axioms fulfilled by the real-world data that would only be detectable when all assertions for the entity were regarded at once. Naturally, this characteristics of OWL can influence the detection of all patterns. Despite that, we consider this characteristic to be of less influence for the scenario we are going to investigate in the following. As long as we are only relying on patterns from a single Linked Data dataset, it is most likely that the largest share of entities is actually referred to by a unique name because this makes the dataset more usable and manageable. Furthermore, in cases where one entity is referred to by several names in the same dataset, a statement that indicates this fact is more likely to be contained in the dataset. Thus, the largest influence of the missing UNA can be expected when working not only on a single dataset but on several datasets at once which is not the focus of our work on learning axioms described in the following.

Keeping these two problems in mind, all class and object property axioms supported by OWL 2 can be learned from instance data. The same holds for data property axioms, however, since data properties are used with literal values in ob-

ject position, approaches more adapted to the different possible datatypes might be more promising than the methods we are looking at for class and object property axioms in the following. In particular, such methods could work on a much more fine-grained level of values allowed by certain datatypes and also try to apply interpolation based on the given values and the knowledge about the datatype. The key difference between the learnability of the different class and object properties lies in the level of susceptibility from the way the open-world assumption and unique name assumption are handled. Most axioms, like class and property subsumption or property domain and range axioms, are mostly influenced by not having a unique name assumption to rely on and only to a smaller degree by the open-world assumption. The open-world assumption gains more influence for axioms that are expressing kinds of negation. This obviously includes class and property disjointness axioms whose learning has to rely on a closed world perspective of the data. Furthermore, this is also the fact for some more complex negation axioms like property irreflexivity and property asymmetry which state negative characteristics about the corresponding object properties.

Axioms can include more informativeness if they are not only defined for atomic class descriptions but also for more complex descriptions like class intersection, class complement or even cardinality restrictions that represent a set of instances. On a theoretical level, dealing with such descriptions when learning from instance data is easily possible since they do not require a handling considerably different from handling atomic class descriptions. However, the main challenge here lies in the fact that, to recognize patterns, it has to be determined whether certain instances belong to a specific complex class description or not. Particularly for more complex descriptions, this might require the usage of inferencing systems which leads to greatly increased computational demands. Moreover, certain class descriptions, especially cardinality restrictions, are also highly connected to the open-world assumption and might lead to a higher probability of making errors when including them in axioms learned from instance data. The more complex the class expressions that should be learned get, the more instance data has to be available to learn from to increase the chance that there actually are instances adhering to this class description which allows to learn expression at a sufficiently high level of quality.

3.2 Association Rule Mining

Some of our approaches for detecting patterns in instance data presented in the remainder of this work are based on association rule mining. Association rule mining is an approach to discover regularities in large amounts of data. Originally, it has been developed for being applied on the shopping baskets of customers of large super market chains. These regularities could then, e.g., be used to improve the arrangement of products in the markets so that products commonly bought together are placed close to each other. With the rise of online shops, approaches for finding

such regularities got even more traction since they allow to recommend additional goods to customers based on buying patterns of other customers. Association rule mining approaches are well-suited for this scenario since they are performing well for large numbers of shopping carts combined with numerous candidate products. The definitions presented here are modeled after those given by Borgelt [18].

Formally, association rule mining operates on a set $I = \{i_1, \ldots, i_n\}$ of possible items, called *item base*. In the online shopping scenario, the item base would consist of all products that could be purchased from the online store. Furthermore, we have a list $T = (t_1, \ldots, t_n)$ given, called *transaction database*, where each $t_i \subseteq I$ ($i \in \{1, \ldots n\}$) is a set of items called *transaction*. The transaction database models the items bought by a customer in a single shopping cart.

Transaction databases are typically depicted as tables where the possible items are represented by columns and the individual transactions form the rows of the table as shown in Table 3.1. In this table, a 1 in cell (i, j) means that item j is contained in transaction i while a 0 at this position means that transaction i does not contain item j. For example, transaction t_1 contains the items i_1, i_4 and i_5 but not items i_2 and i_3.

Table 3.1: Example of a transaction database represented as table

	i_1	i_2	i_3	i_4	i_5
t_1	1	0	0	1	1
t_2	1	1	1	0	0
t_3	0	0	1	0	0

An *association rule* is an implication pattern $A \rightarrow B$ with $A, B \subseteq I$ and $A \cap B = \emptyset$. We call A antecedent and B consequent. This rule can be interpreted in a way that people who purchase the items in A usually also purchase items in B. Thus, an association rule is similar to the logical implication $A \Rightarrow B$ but it is important to keep in mind that association rules might be violated by data.

Association rules are accompanied by the measures of support and confidence. Support describes the number of transactions a given set of items is contained in and is defined on the level of item sets. The support of $A \subseteq I$ is given by

$$\text{supp}(A) = \frac{\#\{t \in T | A \subseteq t\}}{\#T}$$

It is also possible to define the support in absolute numbers instead of normalizing it by the total number of transactions in the transaction database.

The confidence is defined directly for association rules and for a rule $A \rightarrow B$ computed as

$$\text{conf}(A \rightarrow B) = \frac{\text{supp}(A \cup B)}{\text{supp}(A)}$$

Hence, the confidence value of an association rule is the ratio of occurrences of the antecedent in all transactions to the occurrences of both antecedent and consequent

in the same transactions. This results in a value similar to the conditional probability of B given A and models the validity of the rule over the whole transaction database.

To simplify the representation of association rules, in the following, we will write $i_1 i_2 \rightarrow i_3$ for $\{i_1, i_2\} \rightarrow \{i_3\}$ for items i_1, i_2 and i_3 if the intended meaning remains clear.

The process of mining association rules for a given set of items and a given transaction database T consists of two steps. First, the transaction database is searched for frequent itemsets, i.e., itemsets whose support exceeds a defined threshold. Second, based on the set of frequent item sets, the association rules are generated. In the following, we describe both steps separately.

Computation of Frequent Itemsets

For discovering frequent itemsets in a transaction database, one approach would be to compute the support for all possible subsets of the itembase. However, for an itembase containing n items this would mean computing the support value for a total of 2^n itemsets. This problem was first tackled by Agrawal and Srikant [3] who proposed the so-called Apriori algorithm.

The Apriori algorithm operates in a bottom-up fashion by starting with all itemsets containing only a single item and then moving to larger itemsets. From now on, we will write n-itemset for an itemset which consists of n items from the itembase. Algorithm 1 shows the basic Apriori algorithm. It assumes an itembase I and a transaction database T to be given as well as the minimum support threshold $minsupp > 0$. Starting from the frequent 1-itemsets it uses `apriori-gen` to generate candidate 2-itemsets, then counts and filters these candidates to determine all frequent 2-itemsets. Then it successively generates the candidate n-itemsets from the frequent $(n-1)$-itemsets.

The main reason for being more efficient than exhaustively testing all possible subsets of the itembase regarding their support, however, lies in the `apriori-gen` function of the algorithm. This function generates candidate n-itemsets from a given set of frequent $(n-1)$-itemsets as shown in Algorithm 2.

The efficiency of this algorithm relies on the fact that, for itemsets i and j with $i \subseteq j$, the relation $\mathrm{supp}(i) \geq \mathrm{supp}(j)$ holds. Given that all transactions containing itemset j also contain itemset i this relation is obvious and directly implies that an itemsets can only be frequent if all its subsets are also frequent. This property is called *anti-monotonicity* and is used in the second part of the algorithm to prune the set of candidate itemsets. Furthermore, this property is also used for the termination criterion of Algorithm 1 since there cannot exist any frequent $k+1$-itemsets if there are no k-itemsets.

It remains to show that the set of candidate k-itemsets is complete, i.e., all frequent k-itemsets are contained in C_k. The first line of Algorithm 2 generates the unions of all itemsets in L_{k-1} which share $k-2$ items. Hence, the resulting union of itemsets i_1 and i_2 is equivalent to $C_k = (i_1 \cap i_2) \cup (i_1 \setminus i_2) \cup (i_2 \setminus i_1)$ where

Algorithm 1 The Apriori algorithm

$L_1 \leftarrow \{\}$
for all 1-itemsets i **do** ▷ Determine all frequent 1-itemsets
  if $\text{supp}(i) \geq minsupp$ **then**
    $L_1 \leftarrow L_1 \cup \{i\}$
  end if
end for
for $k = 2$; $L_k \neq \emptyset$; $k = k + 1$ **do**
  $C_k \leftarrow \texttt{apriori-gen}(L_{k-1})$ ▷ Generate candidate itemsets
  for all transactions $t \in T$ **do**
    $C_t = \{c \in C_k \mid c \subseteq t\}$
    for all $c \in C_t$ **do**
      $count[c] = count[c] + 1$ ▷ Count occurrence in transaction
    end for
  end for
  $L_k = \{c \in C_k \mid count[c] \geq minsupp\}$
end for
$res \leftarrow \cup_k L_k$

Algorithm 2 The candidate itemset generation function

Require: L_{k-1} set of all frequent $k-1$ itemsets
$C_k \leftarrow \{i_1 \cup i_2 \mid i_1 \in L_{k-1} \wedge i_2 \in L_{k-1} \wedge \#(i_1 \cap i_2) = k\}$
for all $c \in C_k$ **do**
  for all $(k-1)$-item subsets s of c **do**
    if $s \notin L_{k-1}$ **then**
      $C_k \leftarrow C_k \setminus \{c\}$
    end if
  end for
end for
return C_k

the first intersection contains $k-2$ items while the set differences contain 1 item each and thus the resulting union is a k-itemset. All itemsets not in C_k cannot be frequent. For showing this, we assume to have a frequent itemset $a \notin L_k$. There has to be a subset $s_1 \subset a$ with $\#s_1 = k-1$ and $s_1 \in L_{k-1}$ otherwise a could not be frequent according to our previous observations. Thus, we have a single item x which is in a but not in s_1. Since a is not in L_k, we also know that there cannot be an itemset $s_2 \in L_{k-1}$ with $\#s_2 = k-1$ and $x \in s_2$ otherwise a would be contained in L_k. But because $s_2 \subset a$ and $s_2 \notin L_{k-1}$, a cannot be frequent which contradicts our assumption. Consequently, C_k as constructed in the first line of the algorithm is a superset of the set of all frequent itemsets. Thus, Algorithm 2 generates a set of itemsets which contains all frequent k-itemsets.

This set of candidate itemsets is then pruned further in Algorithm 1 by checking the support against the transaction database. After pruning, L_k exclusively contains all frequent k-itemsets.

3.2.1 Generating Association Rules

The discovered frequent itemsets are used to compute association rules for the considered transaction database. For each frequent itemset i, the basic approach is to compute all non-empty subsets $a \subset i$ and use these to generate rules of the form $a \rightarrow (i \setminus a)$. These generated rules can be evaluated by means of the confidence measure as described above and be included in the set of relevant association rules if they exceed the required minimum confidence threshold.

Similar to the discovery of frequent itemsets, for association rules there is a criterion which provides an upper bound for a rule's confidence value given an association rule derived from the same itemset. Let an association rule $a \rightarrow (i \setminus a)$ for a frequent itemset i and $a \subset i$ be given. Then, for a set $a' \subset a$, we know that the association rule $a' \rightarrow (i \setminus a')$ cannot exceed the confidence of $a \rightarrow (i \setminus a)$ since the support of a' is greater or equal to the one of a and the support of $i \setminus a'$ is less than the support of its counterpart in the first rule.

Also based on this characteristic of confidence, Agrawal and Srikant [3] proposed an efficient way of computing all association rules from given frequent itemsets which is shown in Algorithm 3. This algorithm requires a single frequent k-itemsets and a set of m-item consequents.

In each run, the function AP-GENRULES generates all association rules for the given frequent itemset l_k which have an $(m+1)$-item consequent and exceed the minimum confidence threshold $\mathrm{minconf}$. To prevent the generation of superfluous rules which cannot exceed the threshold according to the rule stated above, it always relies on the set of all m-item consequents H_m. Based on this set and using the APRIORI-GEN function, $(m+1)$-item consequents h_{m+1} are generated and for each the resulting association rule is determined and checked for sufficiently high confidence value. If it fulfills the confidence threshold, the rule is output otherwise h_{m+1} is removed from the set of relevant $(m+1)$-item consequents H_{m+1}. Finally, H_{m+1} is used to recursively call AP-GENRULES for generating all association rules

Algorithm 3 Algorithm for computing association rules from frequent itemsets

function AP-GENRULES(l_k, H_m)
  if k > (m + 1) **then**
    $H_{m+1} \leftarrow$ APRIORI-GEN(H_m)
    for all $h_{m+1} \in H_{m+1}$ **do**
      $conf \leftarrow \text{supp}(l_k)/\text{supp}(l_k \setminus h_{m+1})$
      if $conf \geq \text{minconf}$ **then**
        output $(l_k \setminus h_{m+1}) \rightarrow h_{m+1}$ with confidence = $conf$ and support = $\text{supp}(l_k)$
      else
        $H_{m+1} \leftarrow H_{m+1} \setminus h_{m+1}$
      end if
    end for
    AP-GENRULES(l_k, H_{m+1})
  end if
end function

having $(m + 1)$-item consequents.

To mine all association rules, AP-GENRULES is applied to all frequent k-itemsets with $k \geq 2$. For the first run, the set of 1-item consequents for a frequent itemset l_k is determined from the association rules having 1-item consequents. Those can be generated by using Algorithm 4.

Algorithm 4 Algorithm for computing association rules with 1-item consequents

function GENRULES(l_k)
  $A \leftarrow \{(k-1)\text{-itemsets } a_{k-1} \mid a_{k-1} \subset l_k\}$
  for all $a_{k-1} \in A$ **do**
    $conf \leftarrow \text{supp}(l_k)/\text{supp}(a_{k-1})$
    if $conf \geq \text{minconf}$ **then**
      output $(l_k \setminus a_{k-1}) \rightarrow a_{k-1}$ with confidence = $conf$ and support = $\text{supp}(l_k)$
    end if
  end for
end function

3.2.2 Other Algorithms

As noted by Borgelt [18], the Apriori algorithm for determining frequent itemsets by Agrawal and Srikant as presented above has been mostly superseded by newer algorithms which show advantages regarding their runtime and memory consumption. Instead of computing frequent itemsets in a breadth-first manner many of the newer algorithms, like Eclat [102] or FP-Growth [47], are based on a depth-first

principle and employ an improved candidate generation process. Nevertheless, in the context of this work, we decided to use the Apriori algorithm since it is easier to understand and its performance never showed to be a limiting factor in the approaches presented later-on. Since the discovery of frequent itemsets is a separate step in the process and the actual generation of association rules operates on the results of this step, alternative algorithms might be used as a drop-in replacement without any further modification of other parts of the approach.

3.3 Statistical Schema Induction

In the first part of this work, we extend the approach of *statistical schema induction* which has been introduced by Völker and Niepert [96]. For the sake of self-containedness, we first give an extensive introduction into the work of Völker and Niepert, before later-on describe our contributions which rely on the original approach and extend it further into generating more axiom types.

Statistical schema induction is an approach for inducing a new schema for an RDF repository or enriching an existing but possibly inexpressive schema with new schema information. For this purpose, the actual data contained in the RDF repository is mined following the assumption that the semantics of any RDF resource is revealed by the patterns of its usage in the repository. Stated differently, statistical schema induction or inductive schema learning approaches in general assume that the data contained in the repository follow the rules of a schema which might be explicitly stated but in most cases is only discoverable by means of the patterns shown by the actual data. Thus, the task of inductive approaches is to detect these patterns which indicate certain conditions to hold in the data. This task is similar to the task of finding hidden patterns in the behavior of customers for which association rule mining has been developed. Based on this insight, statistical schema induction applies association rule mining for the pattern discovery. To clarify the idea, we consider the transaction database given in Table 3.2 as it could be derived from an RDF repository. Each row is an instance in the repository while each column is a class occurring in the repository. We put a 1 in a specific cell if the corresponding instance is assigned to the corresponding class. For example, the instance `Michael Jordan` is assigned to the classes `Person` and `Athlete` while `Barack Obama` is assigned to `Person` and `Politician`.

Table 3.2: The idea of statistical schema induction

	`Person`	`Athlete`	`Wrestler`	`Politician`
`Hulk Hogan`	1	1	1	0
`Michael Jordan`	1	1	0	0
`Barack Obama`	1	0	0	1
`Angela Merkel`	1	0	0	1

Considering this example dataset, we see that all instances which are athletes

are persons and that all politicians are also persons. Applying an association rule mining approach, this would lead to association rules like `Athlete` $\rightarrow$ `Person` with an assigned relative support of 0.5 and a confidence score of 1 since all instances in the transaction database show this pattern. The same support and confidence scores hold for the association rule `Person` $\rightarrow$ `Politician` while the rule `Athlete` $\rightarrow$ `Wrestler` only has a support score of 0.25 and a confidence value of 0.5 since the instance `Michael Jordan` does not comply with it. This shows the ability of association rules to point out certain patterns in the data.

To generate schema information from the discovered rules, the corresponding logical axioms in description logic have to be identified. In the case presented here, the discovered association rules show the inclusion of one class into a different one which equals the class subsumption axiom found in description logic. The axiom `Athlete` $\sqsubseteq$ `Person` can be used to cover the relation between athletes and persons while similarly `Politician` $\sqsubseteq$ `Person` does the same for politicians and persons. By enforcing a certain confidence threshold, e.g., of 0.8, we can filter out axioms like `Athlete` $\sqsubseteq$ `Wrestler` which are shown by some but not all instances and thus would introduce possibly wrong schema knowledge.

In general, statistical schema induction as introduced by Völker and Niepert is a three step process that can be applied to RDF repositories for generating or enriching schemas. In Figure 3.1, we show the three steps towards acquiring additional schema knowledge via statistical schema induction and describe them in more detail in the following.

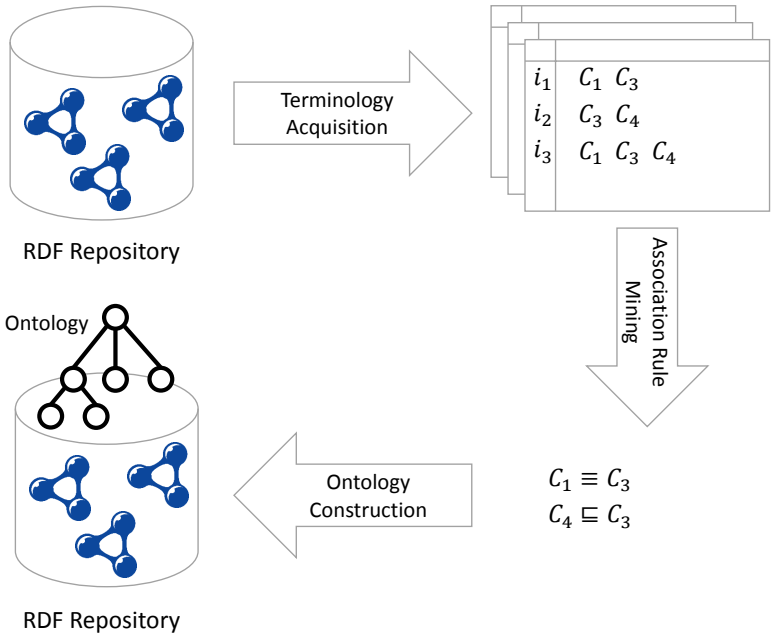


Figure 3.1: Steps of statistical schema induction (based on Völker and Niepert [96])

Terminology Acquisition

In this step, we collect the required information about the data contained in the relevant RDF repository. In principle, all information could be extracted from dumps of the RDF data but since most RDF repositories provide access to their data via a SPARQL endpoint, gathering the relevant data by posing SPARQL queries to this endpoint is the preferred and most convenient way. All data acquired during this step is stored into a relational database which enables rapid retrieval of the data later-on.

First, we gather information about resources which represent *named classes* $C \in N_C$ in the repository. This is done by querying for all resources which are used in the object position of an `rdf:type` axiom:

```
SELECT DISTINCT ?cls
WHERE {
  ?ind rdf:type ?cls
}
```

This has the advantage of including implicitly defined classes which can be expected to often occur in repositories that contain little to no explicit schema knowledge. Since we cannot rely on inferencing abilities being available and active in the repository, querying for all resources of type `rdfs:Class` or `owl:Class` in contrast would only return explicitly defined classes and thus could miss large parts of the class terminology. However, the given query only gives a heuristic for detecting classes in the repository. According to the experimental results presented by Völker and Niepert, it works well for the most common RDF repositories. It is important to note, that only classes which are actually used by instances in the repository are found by this query while classes only defined by explicit `rdfs:Class` or `owl:Class` type statement are not discovered. This does not pose a problem since for the later steps classes that are not used by instances do not have any influence on the results. Each class discovered in the dataset using the heuristic is stored into a relational database system accompanied by a unique numerical identifier. Furthermore, the *instances* which are assigned to classes are determined and stored into the relational database for being used in the generation of the transaction tables.

Similarly, all *named object properties* $P \in N_P$ are extracted from the repository using a heuristic which considers all properties connecting two resources (as opposed to a resource with a data value) as being object properties, assigns them a unique identifier and stores them into the relational database. This is done by posing the following query to the endpoint:

```
SELECT DISTINCT ?pred
WHERE {
  ?r1 ?pred ?r2.
  ?r2 rdf:type [].
}
```

to the SPARQL endpoint. The heuristics for detecting instances used before is again applied by checking the resources for an existing type assignment. In addition to the object properties, all pairs of resources (r_1, r_2) which are connected by an object property are retrieved and stored into the database with a unique identifier.

To limit the schema generation to the current repository, all extracted classes, object properties and instances can be filtered by their URI prefix, e.g., only those having the common URI prefix of the repository are included in the resulting lists. Independently of whether such a filtering step is applied, it is desirable to exclude classes and properties from namespaces with predefined semantics, e.g., the RDF or OWL vocabulary itself, in this step.

Based on the extracted basic terminology, it is now possible to consider more complex class and property expressions. For example, the approach by Völker and Niepert also supports the generation of domain and range restrictions like $\exists r.\top \sqsubseteq C$. To enable the mining of those more complex axioms, it is required to also assign unique identifiers for the class expressions of the types: $\exists r.\top$, $\exists r^{-1}.\top$ and $\exists r.C$ for each $r \in N_P$ and each $C \in N_C$. Furthermore, for each $r \in N_P$ unique identifiers for the expression $r \circ r$ are generated to enable the generation of transitivity axioms.

Association Rule Mining

Before actually mining association rules, we first have to generate *transaction tables* based on the terminology which was extracted from the repository in the previous step. Depending on the types of axioms that should be discovered from a transaction table, the transaction tables are generated by either iterating over all instances or all instance pairs in the repository. For each instance, a transaction is added to the transaction table according to the schema provided in the second column of Table 3.3.

For example, for the generation of each instance a a row containing the identifiers of all classes this instance belongs to is added to the transaction table. For this purpose, the query

```
SELECT DISTINCT ?cls
WHERE {
  <INSTANCE URI> rdf:type ?cls
}
```

is posed to the endpoint after `INSTANCE URI` has been replaced with the actual URI of the instance. Then, the returned classes are resolved to their unique identifiers and added to the transaction table. For the complex class and property expressions, more complex queries have to be considered. For example, for retrieving the identifiers for the complex expressions $\exists r.C$, the query

```
SELECT DISTINCT ?r ?cls
WHERE {
  <INSTANCE URI> ?r ?other.
  ?other rdf:type ?cls.
}
```

can be used which retrieves all pairs of properties and classes of instances assigned using this property.

Furthermore, Table 3.3 shows that some transaction tables can be used to mine different axiom types which means that for generating all axiom types shown in the table, we have to generate six different transaction tables.

After finishing the construction of a transaction table, a frequent itemset and association rule mining approach as described in Section 3.2 is applied to them leading to a set of association rules valid for the RDF repository. Then, these association rules are explored for rules matching the patterns shown in the third column of Table 3.3. For example, an association rule $\{C\} \rightarrow \{D\}$ (after resolving the unique identifiers assigned during the terminology acquisition step) is transformed into an axiom $C \sqsubseteq D$ while an association rule $\{r_i \circ r_i\} \rightarrow \{r_i\}$ results in a property transitivity axiom for the property r. All axioms generated by this approach belong to the OWL 2 EL description logic as defined in the OWL 2 specifications. For each axiom, the confidence and support values of the corresponding association rules are stored for the later process.

Ontology Construction

The last step in the process of statistical schema induction as described by Völker and Niepert is the actual construction of the enriched ontology. We take the confidence values as measures of certainty for the corresponding axiom. For example, only adding axioms with a confidence of at least 1.0 leads to an enriched ontology which exactly fits the actual data in a way that these axioms do not lead to inconsistencies with the assertional data of the repository.

Table 3.3: Contents of transaction tables for generating specific axiom types by means of association rule mining. Each row in the transaction table either corresponds to one instance $a \in N_I$ or a tuple of instances $(a, b) \in N_I \times N_I$. A concept item C_i is added to the transaction if $a \in C_i^{\mathcal{I}}$ for the corresponding instance a. A property item r_i is added to the transaction if the corresponding property holds between the instance tuple, i.e., $a\ r_i\ b$. r_i^{-1} is added if $b\ r_i\ a$. $r_i \circ r_i$ is contained in a transaction if $a\ r_i\ x$ and $x\ r_i\ b$ for an arbitrary instance $x \in N_I$. $\exists r.C$ is added if the instance a is used as subject of r with an object of class C, $\exists r^{-1}.C$ if it is used as object with a subject of class C.

Axiom Type	Transaction Table Row	Association Rule
$C \sqsubseteq D$	$a \rightarrow C_1, \ldots, C_n$ for $a \in N_I$	$\{C_i\} \rightarrow \{C_j\}$
$C \sqcap D \sqsubseteq E$	$a \rightarrow C_1, \ldots, C_n$ for $a \in N_I$	$\{C_i, C_j\} \rightarrow \{C_k\}$
$D \sqsubseteq \exists r.C$	$a \rightarrow C_1, \ldots, C_n, \exists r_1.C_{11}, \ldots, \exists r_m.C_{mn}$ for $a \in N_I$	$\{C_k\} \rightarrow \{\exists r_j.C_{jk}\}$
$\exists r.C \sqsubseteq D$	$a \rightarrow C_1, \ldots, C_n, \exists r_1.C_{11}, \ldots, \exists r_m.C_{mn}$ for $a \in N_I$	$\{\exists r_j.C_{jk}\} \rightarrow \{C_i\}$
$\exists r.\top \sqsubseteq C$	$a \rightarrow C_1, \ldots, C_n, \exists r_1.\top, \ldots, \exists r_m.\top$ for $a \in N_I$	$\{\exists r_j.\top\} \rightarrow \{C_i\}$
$\exists r^{-1}.\top \sqsubseteq C$	$a \rightarrow C_1, \ldots, C_n, \exists r_1^{-1}.\top, \ldots, \exists r_m^{-1}.\top$ for $a \in N_I$	$\{C_i\} \rightarrow \{C_j\}$
$r \sqsubseteq s$	$(a, b) \rightarrow r_1, \ldots, r_n$ for $a \in N_I \times N_I$	$\{r_i\} \rightarrow \{r_j\}$
$r \circ r \sqsubseteq r$	$(a, b) \rightarrow r_1, \ldots, r_n, r_1 \circ r_1, \ldots, r_n \circ r_n$ for $a \in N_I \times N_I$	$\{r_i \circ r_i\} \rightarrow \{r_i\}$

Chapter 4

Related Work

The approaches proposed in the course of the first part of this work can be categorized as ontology learning methods. In this chapter, we give an overview on the general idea of ontology learning, describe several approaches that have been proposed in this area and give pointers to works that are strongly related to our inductive ontology learning methods presented later-on in the first part.

4.1 Ontology Learning

Ontology learning as coined by Mädche and Staab [69] describes the creation of structured knowledge as represented in ontologies by means of data-mining techniques based on a multitude of different data sources. The primary reason for the development of such approaches is the so-called knowledge acquisition bottleneck which is caused by the fact that manual creation of knowledge bases is a time-consuming and potentially complex task. The introduction of can support knowledge engineers in the creation of knowledge bases by providing proposals for additional axioms. This can considerably lower the time and labor required to create the knowledge base and thus also reduce its overall cost which can be seen as a major step towards a wider application of more expressive ontologies.

Ontology learning approaches can especially be distinguished by the type of data they work on. Many early approaches use textual information which is available in very large quantities also thanks to the Web which provides easy access to a great amount of texts for virtually all fields of knowledge. These approaches are often based on linguistic concepts which are applied to derive logical relations between entities or characteristics of the entities themselves. For example, the so-called Hearst patterns [49], which were originally introduced to find hyponyms in text corpora, have also been adopted in ontology learning. Patterns like “classA is a kind of classB“ are used by the Text2Onto system by Cimiano and Völker [28] or the OntoLearn system by Velardi et al. [93] for detecting subsumption relations between classes. These systems rely on the availability of a mapping between classes in the ontology and terms in the employed texts. Since the task of reliably recog-

nizing which part of a text references a certain class is not trivial, systems like this are much simpler to use for generating ontologies from scratch, where the creation of a class is directly linked to a certain part of the textual information, than for enriching ontologies. In many cases, general data mining techniques have been applied onto the data extracted from the textual information. Especially noteworthy in the context of the work we present later-on is the ontology learning framework Text-To-Onto by Mädche and Staab [68], the predecessor of Text2Onto mentioned above, which employs association rule mining approaches on data retrieved from texts for the purpose of detecting relations between classes. A broader overview on learning ontologies from textual data is given by Cimiano [27]. All these text-based approaches only have a limited applicability with respect to Linked Data. This is due to Linked Data only containing small amounts of texts. In the cases where actual text is available, for example by means of comments, it often only contains short sentences or excerpts which might not be well-suited to apply more advanced linguistic processing. Taking data from the Web for enriching Linked Data schemas is one possibility to work around this limitation, however, finding adequate sources can be challenging and also the alignment of textual information with the corresponding entities in the data repository is a non-trivial task.

Earlier approaches for ontology learning which are not working on texts but are purely logic-based, mostly concentrated on supporting the knowledge engineer in completing the axiomatization of the relevant domain. For instance, Baader et al. [10] used attribute exploration from the field of Formal Concept Analysis to determine which questions have to be posed to a human knowledge engineer to gain the knowledge required to fully express the relevant domain knowledge. Though this approach is promising on smaller knowledge bases, it is very questionable whether it can be gainfully used on datasets of the size typically found in the Linked Data cloud. As Baader et al. themselves state, their approach shares the high computational complexity of Formal Concept Analysis which gives reason to the assumption that it would not be able to handle most Linked Data sources due to their sheer size.

Many of the approaches mentioned above not only support learning schemas, i.e., create axioms for the TBoxes of ontologies, but also provide means of populating ontologies with instances and for generating additional information about the instances and their relations. However, since we target our work on Linked Data sources which already contain high amounts of instances, we do not further detail on such ontology population approaches.

One problem shared by the greater part of the works given here is their limited support for more expressive ontologies. Many of them only support the basic description logic $\mathcal{ALC}$ and thus are far away from exploiting the full potential of OWL. Motivated by this lack of expressiveness, recent work started to support more expressive description logics as used in OWL and OWL 2.

Völker et al. [95] proposed a method for enriching inexpressive schemas with expressive axioms and implemented it in a tool named LExO. Instead of general textual data, the approach requires definitory descriptions to be available for the

classes whose logical descriptions should be enriched with additional axioms. This requirement also simplifies the aforementioned problem of aligning the entities in the ontology with their textual representations since textual definitions are written so that they specify a single concept further. Given a class to enrich and a corresponding definition, Völker et al. employ linguistic techniques including the analysis of the definitory sentences' structure. After applying different processing steps to the structure, pre-defined transformation rules are used to convert the relevant parts of the structure into their logical representation. They propose a number of transformation rules which reach from patterns for disjunctions up to more complex constructs such as existential quantification. Due to the possibility of patterns recursively containing other patterns, this leads to the support of complex axioms. In their critical discussion, Völker et al. highlight a number of problems that they encountered during the evaluation of their approach many of which have their roots in the complexity and dynamics of natural language. Based on the work done in this direction, Völker and Rudolph [97] also proposed a method that combines the LExO approach with Relational Exploration which is used for posing decisive questions to a human ontology engineer.

4.2 Inductive Ontology Learning

An alternative to relying on external data for enriching ontologies with additional axioms is the use of instance data to discover new schema knowledge. One possibility to learn highly expressive axioms from instance data is the application of logics-based ontology learning approaches. The most comprehensive collection of algorithms for this purpose is provided in the DL-Learner system by Lehmann [64]. In particular, DL-Learner applies supervised machine learning techniques to description logics and also uses Inductive Logic Programming (ILP) to learn class expressions for given positive and negative examples in a way that the resulting expression covers as many positive examples while only applying to as few as possible negative examples. For this purpose, Lehmann defines refinement operators which are used to explore the search space of possible class expressions in a structured way in this case by always starting at the top-level class ($\top$) and then exploring the search space in a tree-like manner where child nodes are representing class expressions that are more specific than the class expression of their parent nodes. Thus, by always producing more constraining class descriptions through adding additional constraints, the search space is explored in a structured way and the currently achieved performance can be computed based on the examples. The main challenge in doing this is to control the direction of further exploration to reduce the number of steps required until finding the final result. This is achieved by using heuristics which guide the search In addition, these heuristics are also responsible for introducing a preference to shorter class descriptions which is particularly important in a general ontology engineering scenario where users of the ontology would get quickly overwhelmed by too complex and long class descriptions. The

main bottleneck of the ILP-based approach proposed by Lehmann is its reliance on reasoning techniques. This dependency leads to the fact that the raw approach is hardly applicable to large knowledge bases as they get more and more common in the Linked Data field. Hellmann et al. [50] tackle this problem by extracting fragments from larger knowledge bases on which the actual class expression learning can then be applied. Nevertheless, the reasoning requirement is a major bottleneck in the ILP-based approach.

There are some differences of the approaches which we present in the following two chapters of this work to the DL-Learner approach. First, our approaches do not rely on reasoning tasks to be performed on the instances of a knowledge base but employ data mining approaches for discovering hidden information. Instead, we only retrieve the information from the knowledge base as is and work directly on it. This allows us to include all available instance data into our considerations. Nevertheless, though it is not necessary to use reasoning, our approach is flexible enough to be transparently extended to also gain advantage from inferable knowledge. The second major difference is that the ILP-based approaches presented by Lehmann are supervised which means that they need a set of examples to produce the class expressions for. The positive examples are in most cases retrieved based on their affiliation to the class for which an alternative class descriptions is desired or by simply providing a set of arbitrary instances which should be contained in the class. This allows a much larger amount of control into which expressions are produced but makes the application to a whole dataset at once impractical. In contrast, our methods are based on unsupervised methods and therefore can be applied to a knowledge base without the need of determining positive or negative examples. Furthermore, since our approaches are not primarily developed for producing class expressions, they are easily extendable to produce axioms having not only single classes as a left-hand side as for class definitions but can be used to find more general and complex subsumption relations.

4.3 Learning Disjointness Axioms

When producing expressive schemas, class disjointness axioms are an important part. Nevertheless, they are only rarely used in current real-world ontologies. Thus, in Chapter 5, we are introducing inductive methods for learning class disjointness axioms. There are several works also including or specifically focusing on learning this specific type of axioms. Völker et al. [98] proposed the usage of supervised learning methods to generate class disjointness axioms which they implement in a tool named LeDA. LeDA employs supervised classification algorithms which are first trained on an ontology that has been manually annotated with class disjointness and afterwards are used to classify class pairs as disjoint or not disjoint. For both involved ontologies, all possible class pairs are generated and for each class pair a number of different features is computed. These features can be roughly divided into three groups: lexical, logical and corpus features. The lexical features

are a nummber of different similarity and relatedness measures used on the class labels assigned in the ontology. Some of these measures just operate on the character level like the Levenshtein distance [67] which is used to compute the edit distance between two class labels. Other measures, like the one proposed by Wu and Palmer [101], try to capture the notion of semantic relatedness of words. For this purpose, they resort to the lexical database WordNet [72] and use the lexical relations encoded in this database for determining a measure of relatedness. To find a WordNet sense for a given class label, LeDA resorts to the most common sense for the given word. The logical features work on characteristics of the given ontology. This includes the number of common subclasses of the two considered concepts, the similarity regarding object properties and also a semantic distance measure which is based on the minimum distance path between both concepts that only consists of subclass relations. Finally, the corpus-based features rely on additional external data. This data consists of a text corpus generated from Wikipedia articles corresponding to the ontology concepts and a so-called background ontology created from the text corpus using the Text2Onto [28] tool. The background ontology is used to apply the logical features also applied to the original ontology. On the text corpus, the cosine similarity between the articles corresponding to the classes is computed as a separate feature. Völker et al. evaluate the results of this classification process against a manually created gold standard for the PROTON[1] ontology whose creation is discussed in greater detail especially highlighting commonly encountered human disagreements. The automated class disjointness learning achieves a very good performance with accuracy values up to 90% for a cross-validation scenario.

Since LeDA can be considered the state-of-the-art approach for supervised learning of class disjointness, we later-on compare its results to the results of our different inductive methods. However, there also are clear distinctions between both approaches. The most striking difference is the fact that LeDA works in a supervised manner and thus needs training data for which the disjointness between classes is already known. Since there are many different ways of modeling ontologies the results might be highly dependent on the specific combination of training dataset and ontology to finally classify, thus it is hard to apply this approach in an unsupervised way after a training phase because to reach the best results carefully choosing the dataset to train on is required. Another difference is that LeDA needs supplemental data to compute many of its features. For example, the WordNet database is required to compute similarity measures which leads to limitations in the general applicability of the approach since an analogous database only exists for a very small number of languages and can often be regarded as less reliable and comprehensive than their English counterpart. Therefore, developing unsupervised approaches which do not rely on great amounts of external data can still be beneficial even though they cannot generally be expected to outperform well-tuned supervised approaches such as LeDA accompanied by well chosen training data.

[1] `http://www.ontotext.com/proton-ontology`

Bühmann and Lehmann [22] also proposed ways to generate additional schema axioms solely based on the instance data available in the dataset. For this purpose, they first collect the basic schema knowledge, like classes and properties contained in the dataset, by means of SPARQL queries. Afterwards, the evidence needed to determine which additional schema axioms could hold according to the instance data is retrieved by a number of predefined SPARQL query templates. Bühmann and Lehmann present such templates for different axiom types including class disjointness and class subsumption as well as more complex axiom types like property symmetry and reflexivity. These SPARQL queries return the number of occurrences of the specific patterns which is then interrelated with the overall number of possibilities of the pattern which delivers the confidence for the validity of the specific axiom. This approach always has to be applied to single classes and properties in the ontology. Thus, for enriching the whole ontology, it is required to iterate over all classes in the ontology and retrieve the result using the instantiated templates for the currently desired axiom type.

The main difference to our approach lies in the fact that Bühmann and Lehmann use the counting facilities of the SPARQL endpoint providing the dataset. Our approaches in contrast retrieve the raw instance assignments by means of very simple SPARQL queries from the endpoint and all required counting operations are done implicitly and locally. In cases where additional axiomatizations are only desired for a very limited number of entities in the dataset, the more targeted application scenario of Bühmann and Lehmann provides advantages with respect to runtime and load on the dataset server. However, when there is no such precise need for additional schema axioms, performing the counting operations on the server might prove disadvantageous since it could lead to computing similar results multiple times. Furthermore, the SPARQL pattern-based approach does not scale well when more complex axioms like $A \sqsubseteq B \sqcap C$ are desired. In this case, each additional conjunction element leads to the need for posing another query to the server while especially the association rule mining-based approaches already have all required information available locally so that it only takes small adjustments regarding the parameters of the mining algorithm to also detect patterns representing such axioms. By relying on the well-researched foundations of association rule mining, we can also expect achieve good performance even for more complex axiom types.

Bühmann and Lehmann [23] also extended this initial approach. In this extension, they mainly remove the need of manually defining SPARQL patterns. Rather, they introduce an initialization step in which a set of ontologies is inspected and contained axioms are retrieved. These axioms are generalized by replacing the usage of specific classes with placeholder variables which leads to general axiom patterns that are independent from the actual ontology. After this step, the further processing is done on the dataset to enrich. Bühmann and Lehmann detail this for the example pattern $A \sqsubseteq B \sqcap \exists p.(C)$. By formulating SPARQL count queries, the number of the pattern's occurrences is determined for the left (A) and right hand side ($B \sqcap \exists p.(C)$) as well as for the intersection of both sides of the axiom pattern ($A \sqcap B \sqcap \exists p.(C)$) occurring together. These occurrence counts are then used

to determine precision and recall of the given patterns which exactly resembles the notion of confidence as defined in association rule mining. Precision and recall values are used in the further process to compute the F-measure for a pattern which is then used to score it and decide about the inclusion of the instantiated axiom pattern into the knowledge base. Since the counting process induces high load onto the SPARQL endpoint server, Bühmann and Lehmann also propose to determine fragments from the remote endpoint which can then be processed locally.

All in all, this approach seems to be closely related to the association rule mining based approaches as introduced by Völker and Niepert and to the extensions we are presenting in the course of this work. Bühmann and Lehmann introduce a pattern learning step prior to actually looking at the data to enrich to reduce the number of queries they have to pose to the SPARQL endpoint. This has the advantage of only learning new axioms which are similar to those frequently used in the training ontologies and thus the learned axioms might easier to understand for humans. However, this similarity to common usage patterns also forms a weakness of the approach since it loses the ability to actually enrich datasets with axioms which are not yet commonly used, e.g., the patterns shown in their evaluation do not contain disjointness at all supposedly because it is not commonly contained in the ontologies the patterns were extracted from. Using F-measure as a combination of precision and recall as primary means of judging the applicability of the patterns might introduce problems when trying to learn subsumption patterns. In cases where the occurrence frequencies of both sides of the axiom are not balanced at all (e.g., $|A|$ much smaller than $|B \sqcap \exists p.(C)|$), the precision value will be low and thus influence the F-measure to a large degree. This case is similar to the considerations of symmetry and non-symmetry for association rules as described in Section 5.2 of the next chapter and could finally lead to the exclusion of totally valid subsumption axioms. These two works by Bühmann and Lehmann [22, 23] described here are also the only other approaches that are considering property axioms in the context of ontology learning as we are doing in Chapter 6.

Töpper et al. [92] also learn class disjointness from Linked Data datasets. For this purpose, they employ an approach based on a vector space model as commonly used in the area of Information Retrieval. They represent classes by vectors whose elements are based on the usage of properties for instances of the corresponding class. By computing the distance between two vector representations of classes, they determine the similarity of the classes regarding their property usage. Töpper et al. consider classes with a similarity below a certain threshold as disjoint. According to their evaluation at least 97.4% of the disjointness axioms learned by this approach are correct. Combined with learned domain and range axioms, the disjointness axioms are employed for detecting wrong links during the DBpedia extraction process.

4.4 Profiling Linked Data Datasets

Instead of generating schema information from discovered patterns in the dataset, it is also possible to use them for profiling a dataset and thus gaining insights into the characteristics of the dataset. An early approach into this direction is the work by Böhm et al. [17]. Being concerned with providing profiling techniques for Linked Data sources which enable to more quickly discover and understand the datasets, they propose to examine the data in several steps. First, they cluster the instances based on their similarity regarding property usage and determine a so-called mean schema from the most frequently used properties in the resulting clusters. This clustering step is mainly done to limit the number of entities which have to be processed in further steps. On each cluster of instances, different methods are used for finding patterns which help to understand the actual usage of the data. For example, Böhm et al. use association rule mining and correlations to find properties which are commonly used in combination (e.g., the usage of a property `isbn` implies that the instance also has assigned an `author` property) but also to detect equivalences between properties which are detected by finding negative correlations and association rules between two properties' usages. From the correlations between usages of a property p connecting instances x and y with a different property r connecting the instances in inverse order, they also gain knowledge about properties which show characteristics of inverse properties. These schema-level insights are complemented by further data type and pattern statistics.

Though the approach shows a number of similarities to our approaches, the main difference is the actual usage of the discovered patterns. While Böhm et al. use the pattern primarily for understanding the usage of vocabulary schema better, we specifically use patterns recognized by means of association rule mining and correlations between class and property usages for detecting specific structural knowledge and use it for learning ontology schemas. This especially means that we more carefully define the connections between discovered patterns and the corresponding schema representation. Furthermore, since we are not purely interested in the behavior of the data but also in its implications on schema-level, we perform a manual evaluation of the learned ontology elements. By combining the different items which we use to find additional schema knowledge, we can also provide information about the actual data usage for a number of usage patterns which cannot be as easily detected by the ProLOD system developed by Böhm et al.

Further in the direction of the initial work by Böhm et al. [17], Abedjan and Naumann [1] apply association rule mining to Linked Open Data datasets with the goal of supporting the improvement of the datasets. In particular, they define the structure of their transaction tables by means of so-called configurations of which they propose six. Based on the configurations, the application of association rule mining leads to different discoveries for the dataset including schema discovery, range discovery and schema matching. For example, Abedjan and Naumann propose these configurations to find common co-occurrences of properties or classes for many entities. They especially target at predicate suggestions and

auto-amendment of triples but also on proposals for changing the ontology which might be desired if the ontology and the corresponding data develop in parallel for some time and thus start to diverge. This work differs from ours since its main use case is supporting the knowledge engineer in giving suggestions for modifications in the dataset which lead to improvements or better suit the actual usage in the data. Furthermore, it helps the engineers to discover and understand the usage of the data.

Paulheim [75] also uses association rule mining to detect patterns of types that commonly co-occur and to propose additional types for given instances. The follow-up work by Paulheim and Bizer [77] focuses on completing the types of instances. However, instead of relying on the already existing class assertions of the instances, the properties used with an instance are employed for finding possible types. This is done by considering the typical classes of entities used with a certain property and then aggregating this information for all properties used with the currently considered instance for finally getting its most probable type. The authors argue that this approach could complement or even replace the traditional inferencing which points in the same directions as the arguments given by d'Amato [31] in favor of inductive methods. The advantage of these approaches compared to fully schema-inferencing based approaches is that they do not suffer from problems in the schema or from deviations between schema and the actual schema usage in the data. On the other side, just using statistics-based approaches without ever materializing the generated rules so that an ontology engineer can check them, might also rapidly lead to blowing up noise contained in the data. Therefore, we favor the formalization of rules in a human-readable format so that the rules are available for further checking. By providing confidence annotations for learned axioms, we also circumvent the problem of usual schemas only containing hard-and-fast rules. An ontology engineer can assess the formalized rules also considering the confidence values and thus might remove wrongly learned rules which point to errors in the data.

Chapter 5

Inductive Learning of Class Disjointness Axioms

One central aspect of more expressive ontologies is negation. Nevertheless, it is not widely used in real-world ontologies. Even class disjointness axioms, being the simplest type of negation supported by many ontology languages, are only used scarcely. According to the LODStats project,[1] which crawls datasets in the LOD cloud and provides statistics about the characteristics of the datasets, only 6 out of 365 properly crawled datasets (1.7%) contain class disjointness statements. Glimm et al. [42] performed an analysis of the Billion Triple Challenge 2011 dataset. Though class disjointness belonged to the top-20 OWL primitives employed in the dataset, its total number of usages was low.

This is especially a hindrance when working in the direction of data quality since only negation and other more sophisticated axiom types provide knowledge usable to detect inconsistencies in the dataset. Thus, these axioms are required to identify problems in the data by means of most logics-based automatic approaches. Furthermore, to serve as a documentation of the dataset, an ontology increases its informative value when it not only provides non-binding definitions but also states that certain situations and modeling cases are not possible. Regarding the logical detection of problems, consider the following knowledge base described by means of the DL syntax.

$$\top \sqsubseteq \texttt{navigates.Ship}$$
$$\top \sqsubseteq \texttt{hasHometown.City}$$
$$\texttt{Person}(\texttt{Tom})$$
$$\texttt{navigates}(\texttt{Tom}, \texttt{Berlin})$$
$$\texttt{hasHometown}(\texttt{Tom}, \texttt{Berlin})$$

The TBox of this knowledge base describes the properties `navigates` and `hasHometown` for both of which a domain restriction is given. While the domain of `navigates`

[1] `http://stats.lod2.eu`

is given as Ship, a City is defined to be the domain of hasHometown. In addition, the ABox describes Tom, an instance of the class Person who navigates the instance Berlin which is also his hometown. For a human, it is self-evident that there is some error in the data. Based on the domain restrictions, Berlin would have to be both a city and a ship but common knowledge says that nothing can be both at the same time. Probably, there has to be a ship named Berlin and the city Berlin as separate instances thus the ABox contains errors in using the same instance for the ship and the city. If automatic approaches were applied on this knowledge base, they would not spot any errors since the relevant fact that cities and ships are distinct is not expressed here. Thus, these approaches would not be able to help finding the problems. By enriching the knowledge base with the fact that ships and cities have no common instances, automatic methods for finding errors would become applicable. This statement can be expressed by a class disjointness axiom

$$\texttt{Ship} \sqsubseteq \neg \texttt{City}$$

or equivalently $\texttt{Ship} \sqcap \texttt{City} \sqsubseteq \bot$ which specifies that the classes Ship and City do not share any instances. This axiom would then render the ontology inconsistent since the instance Tom could not be assigned to both classes in any model for the ontology.

Given the low deployment of disjointness axioms in real-world data, a first step into using them for detecting errors in Linked Data is to gather the disjointness holding between the different classes which are contained in the data. In this chapter, we propose approaches which generate class disjointness inductively, i.e., which use instance data from the dataset itself for finding valid class disjointness axioms. The choice was made for several reasons. First, this helps us to limit the requirement for external data. For example, many extensional approaches require text corpora which have some relation to the considered dataset. Though in many cases such data might be easy to find, e.g., from the Internet, for more specialized datasets the additional information might be sparse. As an alternative, textual information assigned to the entities of the dataset could replace this dependency, however there is no guarantee for such textual information in the datasets. In either case, for the most promising results, manual intervention regarding the choice of additional data is most certainly needed. The second reason why we consider the intensional approach here, is that learning disjointness axioms from the knowledge base's data leads to more compact statements regarding its contents which can be helpful not only for being used in automatic approaches but also in scenarios where humans are involved. For example, if a certain disjointness axiom would be expected by a human and it is not learned by the intensional approach, a more detailed inspection could be fruitful since the data probably does not support this axiom and thus either the human expectation or the data might be wrong. In both cases, the manual inspection either leads to the discovery of a structural error in the data or additional insights into it. Disregarding this, for both methods, it is worth to note that manual inspection of their results is always beneficial

and helps to increase the data quality considerably. Given our focus on learning additional axioms for Linked Data, another requirement is important for the proposed approaches. Since Linked Data datasets easily contain many instances, the approaches have to be scalable to these amounts of data. In particular, this makes us disregard methods that rely on logical reasoning because reasoning systems can hardly handle large numbers of instances.

This chapter is based on the work initially presented by Fleischhacker and Völker [39] and Völker et al. [94]. It is structured as follows. First, we report on the creation of a gold standard for class disjointness in the DBpedia ontology in Section 5.1. Afterwards, Section 5.2 focuses on the three approaches for learning class disjointness, one based on correlation and two using association rule mining. In Section 5.3, we report on the results of the different approaches on the DBpedia dataset and also compare the results of a state-of-the-art supervised approach to one of the inductive approaches. Finally, we conclude this section by pointing out the major findings of our experiments in Section 5.4.

5.1 Class Disjointness Gold Standard

For evaluating the results of learning class disjointness axioms, we manually created a gold standard for the relevant classes of the DBpedia ontology version 3.7. To give a better understanding of this gold standard, we provide a thorough description of the creation process in the first part of this section and afterwards we give insights into the findings gained during this process.

5.1.1 Methodology

For making the results of the annotation more comparable between all annotators, we defined a common methodology to apply while annotating the ontology classes with disjointness which we describe in the following along with the overall setting.

We let three annotators independently work on the enrichment of the ontology. The goal was a completely axiomatized ontology containing a disjointness axiom between two classes if and only if the classes are considered disjoint by the annotator. Each annotator was introduced into the methodology presented in Figure 5.1.

We provided all annotators with a cleaned up version of the original DBpedia ontology. During the cleaning step, we removed all classes from the ontology which did not belong to the DBpedia ontology namespace,[2] the RDF or RDF Schema namespace or to the OWL namespace. This was done to limit the gold standard creation to the actual DBpedia ontology and to not include other referenced ontologies linked to from DBpedia. Additionally, we removed all non-class axioms from the ontology.

[2] `http://dbpedia.org/ontology`

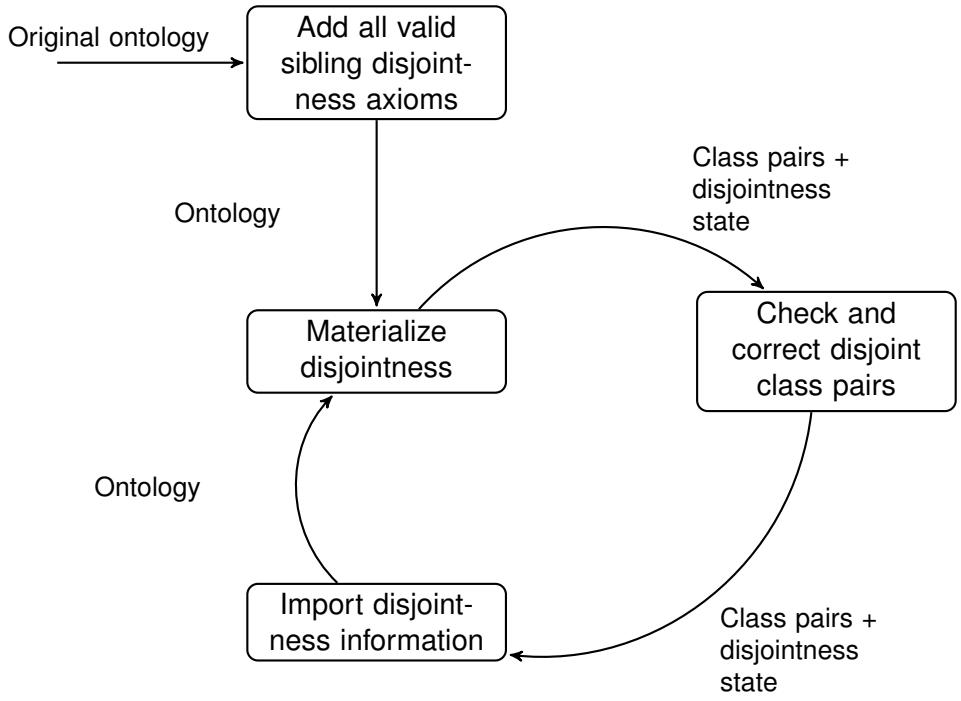


Figure 5.1: Gold standard creation methodology

As a first step, the annotators started adding disjointness axioms using the Protege ontology editor[3] in a top-down manner, first assessing the disjointness of the top-most classes to their siblings and continuing the same on the next level of concepts. This is similar to the disjoint siblings assumption used by Schlobach [84] with the exception, that only manually assessed disjointness axioms are added to the ontology. For this and all following assessments, the annotators had several ways of gathering information on the actual meaning of a class. The most obvious was the class label and the comment possibly assigned to the class but also the external sources like Wikipedia articles were valid resources for finding information relevant for the decision. Furthermore, an inspection of the related classes, especially subclasses, and a small number of instances from the actual DBpedia dataset often helped in understanding a class' semantics.

The next step in the process consisted of materializing all disjointness axioms inferable from the ontology created in the previous step and determine for all pairs of classes in the ontology whether they are disjoint or not (their *disjointness state*). This materialization step helped the engineers to see the actual implications of the disjointness axioms added in the previous step. In the generated list, they were able to directly specify whether a disjointness axiom should be added between both classes or not. Based on this list, additional disjointness axioms were then added to the ontology. Finally, the whole process started again by materializing the

[3] `http://protege.stanford.edu/`.

resulting disjointness axioms to allow the engineers to check the results and correct class pairs erroneously set to be disjoint. The whole process was continued until the engineer was satisfied with the result.

All in all, we consider this workflow to be superior to approaches like the one previously used by Völker et al. [98]. First, by also including the taxonomy, the engineers were able to consider the subclass relations when determining whether to set a class pair as disjoint which proved helpful in some cases for recognizing the actual meaning of classes. In addition, the direct access to the ontology itself allowed the engineers to also access the comments and additional labels contained in the ontology which might clarify the meaning of specific classes further. Moreover, by including a materialization step in the workflow, inferenced disjointness axioms are made apparent to the engineers. We think these aspects considerably outweigh the problem that we had to rely on the subsumption hierarchy potentially being faulty itself also because such problems could be discovered when wrong inferred disjointness axioms are showing up.

5.1.2 Analysis

The resulting gold standard not only gives us the possibility to evaluate disjointness learning approaches against it but also provides us with the possibility to better understand the complexity of disjointness in the DBpedia ontology and the problems arising from this complexity. Starting the analysis of the gold standard, we computed basic statistics about the distribution of votes on axioms which are given in Table 5.1. In this table, the number of concept pairs is given which got a certain number of votes for being disjoint. Obviously, since the gold standard was created by three engineers, this ranges from zero votes for disjointness, i.e., all engineers consider the two concepts not to be disjoint, up to three votes for disjointness, i.e., all engineers consider the given pair to be disjoint. The distribution of votes has been computed for three different sets of class pairs. The set P_{all} contains all possible pairs of classes in the ontology (e.g. $\{A, B\}$). P_{direct} contains all pairs $\{A, B\}$ where A and B are direct siblings, i.e., have a common direct superclass. The third set of pairs contains all pairs $\{A, B\}$ for which neither $A \sqsubseteq B$ nor $B \sqsubseteq A$ holds. This set is called $P_{indirect}$ in the following. This three-fold evaluation was done to particularly examine the annotators' performance on sibling classes where we expected the main complexity in the decision whether two classes are disjoint.

As we can see from these statistics, all annotators agreed on the disjointness and non-disjointness for the majority of pairs. Furthermore, the agreement is considerably lower for the direct siblings case compared to the all pairs case.

To more formally quantify the agreement between annotators, so-called inter-annotator agreement measures are commonly used. The *inter-annotator agreement*, also called inter-rater or inter-coder agreement, is a way of quantifying the agreement between annotators on the same task. There are many different inter-annotator agreement measures proposed which differ, e.g. regarding the number of annotators they can compute agreement between, the number of annotation cat-

Table 5.1: Basic statistics about the gold standard

	$\mathbf{P_{all}}$		$\mathbf{P_{direct}}$		$\mathbf{P_{indirect}}$	
	#	**%**	**#**	**%**	**#**	**%**
0 votes	2,412	6.09	487	28.92	1,948	4.98
1 vote	286	0.72	72	4.28	269	0.69
2 votes	432	1.09	74	4.39	432	1.10
3 votes	36,491	92.10	1,051	62.41	36,486	93.23

egories and whether they take agreement by chance into account. An introduction and overview is provided by Artstein and Poesio [6] whose article also provides the basis for the following definitions. For our scenario, we have to consider the annotations done by three annotators on a binary scale. We assess the agreement by means of the observed agreement without any chance correction and additionally by means of Fleiss' κ, which is referred to as Fleiss' multi-π by Artstein and Poesio. The most basic definition regarding inter-annotator agreement is the observed agreement for two annotators which is defined as:

$$A_O = \frac{1}{\sharp I} \sum_{i \in I} \mathrm{agr}_i \tag{5.1}$$

with I being the set of all annotated items and agr_i an indicator whether the annotators agreed or not, given by

$$\mathrm{agr}_i = \begin{cases} 1 & \text{if both annotators assign item } i \text{ to the same category} \\ 0 & \text{if both annotators assign item } i \text{ to different categories} \end{cases} \tag{5.2}$$

Based on this two-annotator case, the generalization for a set of annotators C who annotate items from a set I with categories a and b is given in Equation 5.3. The variables n_{ai} and n_{bi} give the numbers of annotators who assigned category a or b to item i.

$$A_O = \frac{1}{\sharp I \sharp C(1 - \sharp C)} \sum_{i \in I} n_{ai}(n_{ai} - 1) + n_{bi}(n_{bi} - 1) \tag{5.3}$$

The problem of the simple observed agreement is that the number of annotators and, in the more general case, the number of categories directly influence the result of the agreement measure. As a result, adding superfluous and thus never chosen categories could lead to a raise of the observed agreement. Because of this, the so-called chance-corrected measures have been proposed which use the observed agreement but only consider it in relation to the agreement which would occur if all annotators would just randomly pick the category for each item. Chance correction is included in several measures which amongst others differ regarding the random distribution lying behind this random choice. Fleiss' κ, which we will use in this

work, assumes an equal distribution for all annotators. This distribution is derived from the actual distribution of the category assignments which leads to an expected agreement A_E as shown in Equation 5.4 and finally to the actual measure provided in Equation 5.5.

$$A_E = \frac{1}{(\sharp I \sharp C)^2} \sum_{k \in K} n_k^2 \tag{5.4}$$

$$\kappa = \frac{A_O - A_E}{1 - A_E} \tag{5.5}$$

The resulting measure is in the range of -1 to 1, where 1 means perfect agreement while 0 means that only the agreement was achieved that would have to be expected by chance. Values of less than 0 hence mean that the agreement was even lower than to be expected by chance. One possibility to qualify the level of agreement is proposed by Landis and Koch [63] as shown in Table 5.2.

Table 5.2: Levels of agreement according to Landis and Koch [63]

Kappa Values	Strength of Agreement
< 0.00	Poor
0.00 - 0.20	Slight
0.21 - 0.40	Fair
0.41 - 0.60	Moderate
0.61 - 0.80	Substantial
0.81 - 1.00	Almost Perfect

Table 5.3 provides observed agreement and Fleiss' κ values for the annotation made for creating the gold standard based on the categories of a class pair being disjoint or non-disjoint. As we see, the intuitive observation of a good agreement is confirmed by these values which all denote almost perfect agreement according to the classification of Landis and Koch.

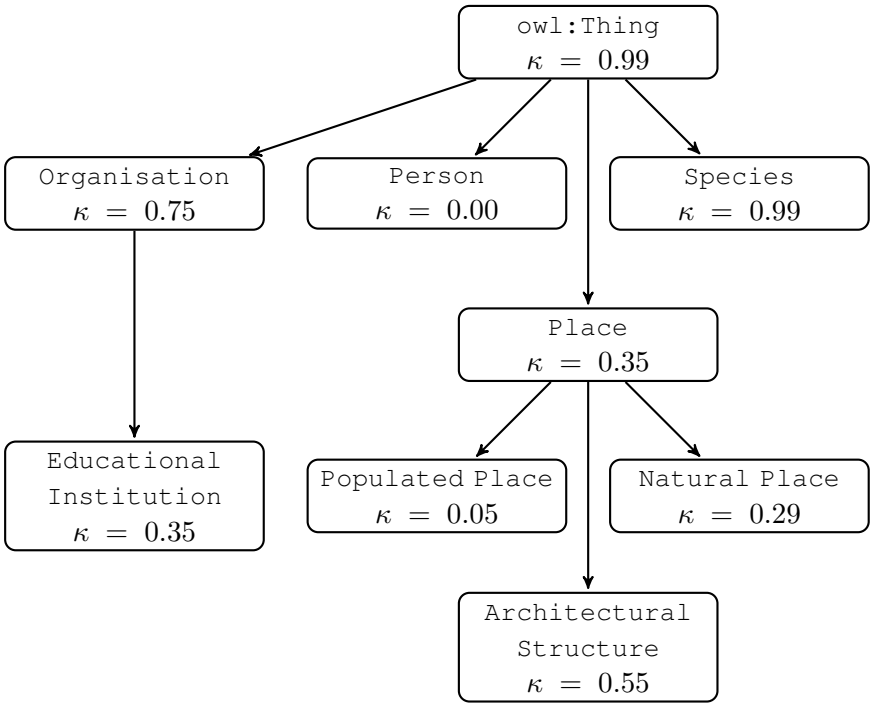
Table 5.3: Gold standard inter-annotator agreement

	$\mathbf{P_{all}}$	$\mathbf{P_{direct}}$	$\mathbf{P_{indirect}}$
Observed	0.988	0.942	0.988
Fleiss' κ	0.989	0.870	0.989

Since these values are computed over all pairs of the ontology they do not provide us with information whether some class pairs are harder to assess than others when annotating them. We especially assume the complexity of the annotation to differ for the types the classes represent, thus, we performed the inter-annotator agreement evaluation not only on the full ontology but also on class pairs where both classes belong to the same subtree of the ontology. Since the class tree is

spanned by subclass and hence is-a relations, the class at the root node of each subtree identifies the super type of all classes in this subtree. Some results of this analysis are depicted in Figure 5.2. In this figure, the given κ-value shows the agreement reached for class pairs where both classes are in the subtree spanned by the concept shown in the diagram.

Figure 5.2: Inter-annotator agreement for subtrees of selected classes



As we can see in this diagram, the inter-annotator agreements for different subtrees differ widely. High agreements are mostly achieved for subtrees containing classes which can be separated accurately like the species subtree consisting of biological classifications. Since biological taxonomies are disjoint, the disjointness of the classes representing the biological classes is clearly defined. Most strikingly, the agreement on the subtree of the class `Person` is 0.00 which would mean no agreement above chance. However, the observed agreement for this subtree is higher than 0.99 which means that the annotators almost all the time agreed in their judgment. Considering the actual assessments given by the annotators, it turned out that nearly all pairs where marked as non-disjoint and that the annotators had only very few disagreements. This led to a bias in the distribution towards non-disjointness and thus raised the expected agreement very high so that the few disagreements had a great influence on the final agreement measure. The fact that there is such a high agreement for non-disjointness in the `Person` subtree hints that the annotators considered the fact that the membership to many of these classes can change in the course of time and thus (time-independent) class disjointness might

not be the right way of representation here.

We also created a list of all cases of disagreements between the annotators, i.e., all pairs for which we only had one or two votes for disjointness. This list contained 928 class pairs. To find out more about the reasons for these disagreements, we passed the resulting list back to the annotators and let them discuss the individual cases. From the notes taken during the annotators' discussion, we tried to categorize the disagreements by the facet which caused them. This categorization is similar to the categorization done by Völker et al. [98] for their gold standard. In the following, we describe the most common categories in more detail.

Many disagreements were caused by problems *finding the right interpretation* of the classes in the ontology. Most of the time, the contents of the ontology itself led to confusion like, e.g., when the `RailwayLine` class was subsumed by the class `ArchitecturalStructure` but at the same time the comment of the former stated that it should "not be mistaken for a railway track" which was obviously the interpretation leading to the subsumption. Another similar case was the class `Infrastructure` which should have been named `InfrastructureAsset` since it only described parts of the infrastructure and not the (rather abstract) infrastructure itself. Similarly, for spatial concepts like in `AdministrativeRegion` or `ProtectedArea`, the annotators disagreed in their judgments because of the missing additional documentation further describing the intended interpretation.

Another common problem which we discovered, was similar to the common problem about properties changing in the course of time mentioned above. It occurred for classes which were assigned to an instance because it had a specific property but where the *instance might lose this property in the course of time*. During the creation of the gold standard, one example for this was the class `Theater` which was a subclass of `Building`. Some annotators remarked that this class could hardly be disjoint to `Museum` which is also subsumed by `Building` since a theater that is no longer in use could be used as a museum. This problem seems to be close to the well-known aspect of rigidity as also discussed by the OntoClean approach [45] where the modeling of food classes is discussed which also posed some problems to our annotators.

Further disagreements were caused by concepts where the *intentions were disjoint while the extensions were equal*. An example we found in the ontology for this case were the concepts `Band` and `MilitaryUnit`. Some annotators argued towards disjointness while others pointed out that there are military bands being established as military units for themselves.

One problem which even led to incoherence in the gold standard was posed by the class `Library` which was subsumed by both `Organisation` and `Building`. Since those two superclasses were labeled as disjoint by all annotators, the class `Library` became unsatisfiable. This problem is not solvable without modifying the original ontology so that either one of the subsumption axioms is removed from it or the `Library` class is split into one class for the organization of a library and one for the building the library is located in.

All in all, from the discussions during this phase, we were able to conclude

that the main problem in many cases could lie in the fact that the ontology was created in a crowd-sourced scenario. Thus, there is no dedicated group of people who maintains the ontology but different people develop the ontology without any supervising authority. Also the fact that there is hardly any documentation describing the background of modeling ontologies and that probably very few contributors have experience in modeling ontologies, seems to have led to a mixture of different modeling approaches which maybe even contradict each other to a certain degree. This in turn made it very hard even for ontology engineers to create a coherent disjointness axiomatization based on the ontology. In addition, we also discovered a number of problems which were already known from other ontology modeling contexts not focused on the modeling of disjointness axioms.

5.2 Approaches

All three approaches presented here are working on the instance data of a data source to deduce disjointness axioms for this data. The relevant parts for this purpose are the instances contained in the knowledge base and the classes these instances are assigned to. For all approaches, we consider transaction tables as already used by Völker and Niepert [96] where each row represents an instance of the dataset and each column represents a class to which at least one instance is assigned. This leads to the basic representation as shown in Table 5.4.

Table 5.4: Example representation of instance and class data. 1 means that this instance is assigned to the corresponding class, 0 means that no such assignment exists in the dataset.

IRI	`Place`	`City`	`Person`	`OfficeHolder`
`Berlin`	1	1	0	0
`Charles_Darwin`	0	0	1	0
`Eiffel_Tower`	1	0	0	0
`John_F._Kennedy`	0	0	1	1
`Golden_Gate_Bridge`	1	0	0	0

For the following descriptions of the different approaches, we assume that such an instance-class table already given.

5.2.1 Correlation-Based Approach

As a first approach for finding disjointness axioms from the instance to class assignments, we consider only pairs of classes and determine how often instances are assigned to both classes at the same time by means of the `rdf:type` statement. The basic idea is that for classes which are disjoint there should be no (or

only few, in case of data errors) instances that are assigned to both classes. Thus, if an instance is assigned to one class, it is not assigned to the other one and vice versa while for non-disjoint classes assignment of the same instance to both classes can be expected more regularly.

For capturing the common occurrence of classes in this first approach, we use the *Pearson product-moment correlation coefficient*, often also referred to as *Pearson's r*, which is a widely known and commonly used measure for the strength of linear relation of two sequences of data. Pearson's r has also been used in a similar fashion by Antonie and Zaïane [5] in combination with association rules for filtering. Its general form for sample data is defined as follows:

$$r = \frac{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{\sqrt{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)^2} \sqrt{\sum_{i=1}^{n} \left(Y_i - \bar{Y}\right)^2}} \tag{5.6}$$

For the application on the class disjointness problem, we consider the instance-class data mentioned above for a pair of classes C and D and compute the correlation between the sequences of 0 and 1 given by the columns corresponding to the classes. Based on these sequences, we compute the number of occurrences of the four possible combinations of classes as depicted in the following table.

	C	$\neg C$	
D	n_{11}	n_{01}	n_{*1}
$\neg D$	n_{10}	n_{00}	n_{*0}
	n_{1*}	n_{0*}	

For this specific variant, which only contains binary data, the Pearson correlation coefficient can be reduced to the so-called ϕ-coefficient:

$$\phi = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{n_{1*}n_{0*}n_{*0}n_{*1}}} \tag{5.7}$$

Given the correlation coefficients for a pair of classes, we can assess the strength of their relation. The possible correlation coefficients are in the interval $[-1.0, 1.0]$ where 1.0 shows a total positive correlation and -1.0 a total negative correlation. The former would be reached for two perfectly equivalent classes while the latter is the result of two classes which are mutually exclusive and collectively exhaustive, i.e., for classes which are a disjoint union of the whole set of instances.

For a more fine-grained subdivision based on the values of the correlation coefficients, we use the coarse categorization provided by Cohen [29]. According to this work, a strong correlation is indicated by absolute values greater than 0.5, medium correlation by absolute values in the range from 0.3 to 0.5 and small correlation by absolute values in the range from 0.1 to 0.3. Absolute values of less than 0.1 are considered as inexpressive.

Based on these considerations, we can conclude that high negative correlation coefficient values give the most evidence for a disjointness holding between both classes. In the final approach, we consider class pairs with negative correlation coefficients as disjointness candidates. We use the absolute value of the correlation coefficient as confidence value for the disjointness.

For the transaction database shown in Table 5.4 and the classes `Place` and `OfficeHolder`, we would get $\phi = \frac{-3}{\sqrt{24}} = -0.61$. From this strong negative correlation, the correlation-based algorithm would propose a disjointness axiom between both classes using the absolute correlation value as confidence.

5.2.2 Association Rule Mining-Based Approach

Both the second and the third approach which we present here for generating disjointness axioms from the data in Table 5.4 are based on association rule mining as introduced in Section 3.2. These methods develop the idea of Völker and Niepert regarding subsumption mining further for finally generating disjointness between classes. Analogical to the association rule pattern $A \rightarrow B$ for classes A and B, which leads to a class subsumption $A \sqsubseteq B$, the idea is to discover association rule patterns like $A \rightarrow \neg B$ for finding disjointness axioms like $A \sqsubseteq \neg B$. These types of associations rules are frequently referred to as *negative association rules*. More precisely, association rules are called negative association rules if one or both of antecedent and consequent are negated, i.e., are sets not contained in a transaction (cf. [104]). In case of negative association rules, the task of association rule mining is not to find a set of items whose occurrence makes the presence of another item set more probable. Instead, the task is, e.g., finding sets of items whose presence makes the absence of another set more likely.

First, we describe the method most similar to the standard (positive) association rule mining approaches. It exploits the fact that for a class A not only A itself but also the class complement $\neg A$ represents a valid class description. Thus, a possible approach for generating association rules like $A \rightarrow \neg B$ is to enrich the data shown above by the assignment of instances to complement classes. As described in the beginning of this chapter, we usually do not have the knowledge available which instance belongs to which complement class. Thus, we assign those instances to the class complement that are not asserted to belong to the corresponding class. This allows us for circumventing the datasets' limitation although it contradicts the open-world assumption semantics employed by OWL and RDF as explained in Chapter 3. Applying this approach extends the transaction data represented in Table 5.4 as shown in Table 5.5.

This leads to some changes in the characteristics of the transaction tables for the data. First, we double the number of items in the item base. Second, the size of each transaction changes substantially. Typically, there are many more items *not* contained in a transaction than items contained in a transaction. But by adding the complement classes to each transaction, this is no longer valid. Since for each class either the positive class or its complement is added to the transaction, we

Table 5.5: Transaction database containing materialized class complements

IRI	Place	City	Person	OfficeHolder	¬Place	¬City	¬Person	¬OfficeHolder
Berlin	1	1	0	0	0	0	1	1
Charles_Darwin	0	0	1	0	1	1	0	1
Eiffel_Tower	1	0	0	0	0	1	1	1
John_F._Kennedy	0	0	1	1	1	1	0	0
Golden_Gate_Bridge	1	0	0	0	0	1	1	1

know in advance that each transaction will be of the size $\sharp I$ for an item base I that contains both positive and complement class items. Furthermore, this also leads to an increase regarding the number of frequent itemsets because the different complement class items co-occur far more often than the positive classes. All this together renders this approach, which is sometimes called naïve association rule mining [5, 91], hardly applicable in the general scenario. Nevertheless, for our use case we operate on a much more limited problem space, in particular, we typically have to handle only a few thousand class descriptions compared to the itemset sizes of around 10^6 or even more found in other applications of association rule mining.

In our example transaction database, we consider the itemset

$$\{\texttt{Place}, \neg\texttt{OfficeHolder}\}$$

which reaches a support value of 3 because these items are contained in the transactions for `Berlin`, `Eiffel_Tower` and `Golden_Gate_Bridge`. For the negative association rule $\texttt{Place} \rightarrow \neg\texttt{OfficeHolder}$, we thus get a confidence value of

$$\frac{\text{supp}(\texttt{Place}, \neg\texttt{OfficeHolder})}{\text{supp}(\texttt{Place})} = \frac{3}{3} = 1$$

From the resulting association rules that follow the pattern $A \rightarrow \neg B$, we create corresponding disjointness axioms. Since association rules are implication pattern, the validity of a rule like $A \rightarrow B$ does not necessarily imply the validity of its symmetric counterpart $B \rightarrow A$. In contrast to association rules, class disjointness axioms like $A \sqsubseteq \neg B$ are indeed symmetric and thus $(A \sqsubseteq \neg B) \models (B \sqsubseteq \neg A)$ holds. To prevent unexpected results, this has to be reflected in the generation of axioms. Due to the symmetry of disjointness axioms, there are still two possible rules which hint to the disjointness of classes A and B, the rules $A \rightarrow \neg B$ and $B \rightarrow \neg A$. If both rules were transferred into their corresponding class disjointness axioms, for each class pair there could be two axioms stating their disjointness with possibly different confidences. When applying a confidence threshold on

these axioms, the disjointness axiom would be added to the ontology if one of the confidence thresholds is sufficiently high. However, for certain cases, this would lead to wrong results. One such case could occur when considering classes A and B with $A \sqsubseteq B$ for which the cardinality of B is much greater than the cardinality of A, e.g., $\sharp A = 100$ and $\sharp B = 1,000,000$. Based on these numbers, we know that

$$\text{conf}(A \rightarrow \neg B) = \frac{999,900}{1,000,000} = 0.9999 \tag{5.8}$$

and thus the assigned disjointness axiom could be chosen for the final ontology based on this high confidence value. Obviously, this would render the resulting ontology incoherent. The subsumption relation between A and B is only clearly detectable for the inverse association rule

$$\text{conf}(B \rightarrow \neg A) = \frac{0}{100} = 0 \tag{5.9}$$

which excludes the corresponding disjointness axiom from the final ontology for all reasonable thresholds. To prevent this kind of problems, we generate the disjointness axioms according to the following rules:

1. only convert association rules $A \rightarrow \neg B$ to disjointness axioms if the inverse rule $B \rightarrow \neg A$ is also a discovered association rule and

2. assign the confidence value for the disjointness axiom as

$$\text{conf}(A \sqsubseteq \neg B) = \min(\text{conf}(A \rightarrow \neg B), \text{conf}(B \rightarrow \neg A)) \tag{5.10}$$

These additional rules prevent the generation of disjointness axioms for subclass relations as given in the example. If generated at all, disjointness axioms for such association rules would not be included in the final ontology since either the association rule directly leading to the disjointness axiom has a too low confidence value or the final confidence value of the axiom is lowered by the corresponding inverse association rule.

5.2.3 Negative Association Rule-based Approach

As described, standard association rule mining algorithms are not originally developed for being used for negative association rule mining. Nevertheless, the mining of negative association rules is a use case which has many potential applications. Thus, there are some algorithms particularly concentrating on the mining of negative association rules [5] including the algorithm proposed by Zhang and Zhang [104]. In comparison to the approach described above, this method mitigates the need of materializing the complement classes and instead works directly on the basic transaction database like the one shown in Table 5.4. The basic idea

behind this approach is to search the database for infrequent positive itemsets. Because of the sparsity of the original transaction database there is an almost exponential number of such infrequent positive itemsets. To reach a well enough performance for such an approach, pruning the search space is an important concern. After generating infrequent itemsets, Zhang and Zhang prune those itemsets which are not considered interesting given the minimum interest level. In this context, an itemsets is called *interesting* if its support exceeds the expected level of support. Based on the remaining negative itemsets, they define an approach to create all possible negative association rules using the probability ratio of each association rule as the corresponding confidence.

5.3 Evaluation

In this section, we describe the experiments performed on the created gold standard for the DBpedia ontology version 3.7 and the corresponding DBpedia dataset version 3.7.

We extended the tool GoldMiner by Völker and Niepert [96] by implementing the three inductive approaches introduced above. For these approaches, the first part of the implementation consisted of querying the DBpedia SPARQL endpoint. Depending on the employed approach, the next step was either to directly compute the correlation between the instance assignments to pairs of retrieved classes for the correlation-based approach or to write the transaction table into files for the association rule approaches. In the latter case, the file representations of the transaction tables were then further processed for finding the relevant association rules. As an implementation for usual association rule mining, we used the Apriori miner tool by Borgelt and Kruse [19] in particular because of its high performance. Its output contained the association rules found during the mining step and were then parsed by our implementation to find the negative association rules from which we finally derived class disjointness axioms. For negative association rule mining, we were not able to find a publicly available implementation. Thus, we implemented the approach proposed by Zhang and Zhang [104] ourselves and used it for discovering negative association rules and derive class disjointness axioms from.

For the Apriori association rule mining step, we applied an absolute support threshold of 10 while not enforcing a confidence threshold since the application of our symmetry handling rules led to the sole generation of high confidence axioms. When applying the negative association rule mining approach, we also used an absolute support threshold and in addition a confidence threshold of 0.8 which showed promising results in first pre-studies. On the correlation approach, we applied the threshold values 0.05 and 0.005 corresponding to negative correlations of -0.05 and -0.005 respectively. Even if these values are both beneath the limits for meaningful correlations, we nevertheless chose these after some first experiments since the results were promising.

For the full evaluation based on our gold standard, we determined different

sets of valid class disjointness axioms. The first one consisted of all class pairs for which at least two annotators considered them as disjoint. In the following, we call this gold standard dbpedia50. For the second gold standard, we only included class pairs for which all annotators agreed in the disjointness. This gold standard is referred to as dbpedia100. For both gold standards, we also included only those non-disjointness statements for which the annotators agreed on the same level as for the disjointness statements. Thus, dbpedia50 only marked a class pair as disjoint if at least two annotators considered the classes to be non-disjoint while dbpedia100 only contained those pairs for which all annotators agreed on the non-disjointness. To evaluate the generated disjointness lists, we then used the set of pairs which are either considered as disjoint or non-disjoint in the current gold standard and left the other pairs out of the evaluation.

During the analysis presented in this section, we are particularly interested in the precision of the learned disjointness axioms. Precision together with recall are measures typically used in the area of Machine Learning for assessing the performance of two-class classification tasks regarding one of the classes. In this case, these measures compare the result generated by machine learning approaches against a manually created gold standard and provide an insight into both the correctness (*precision*) and the completeness (*recall*) of the result. Their definition is based on sets:

- TP containing the *true positive* examples, i.e., all examples annotated as positive in both gold standard and classification result,
- TN containing the *true negatives* examples which are annotated as negative in both gold standard and classification result,
- FP containing the *false positives* which are examples classified as positive but are annotated as negative in the gold standard and
- FN containing the *false negatives* being annotated as positive in the gold standard but classified as negative.

Using these four sets, precision and recall are defined as given in Equations 5.11 and 5.12, respectively.

$$\text{prec} = \frac{\sharp TP}{\sharp(TP + FP)} \tag{5.11}$$

$$\text{rec} = \frac{\sharp TP}{\sharp(TP + FN)} \tag{5.12}$$

Obviously, it is a much simpler task to reach either a precision or recall value of 1.0 when not paying attention to the other measure. Thus, in many cases a certain level of balance between precision and recall is desired. This is captured in the

so-called *F-measure* which combines precision and recall into a single measure. It is defined as the harmonic mean of precision and recall as follows:

$$\mathrm{F}_1 = \frac{2 \cdot \mathrm{prec} \cdot \mathrm{rec}}{\mathrm{prec} + \mathrm{rec}} \tag{5.13}$$

Another measure tightly connected with precision and recall is *accuracy*. It is used to get an assessment for the overall performance of the classification considering both classes of the classification results and is defined as:

$$\mathrm{accuracy} = \frac{\sharp TP + \sharp TN}{\sharp TP + \sharp TN + \sharp FP + \sharp FN} \tag{5.14}$$

As an additional reference to compare the results to, we also provide the performance of the majority baseline on the gold standard as well as the performance baseline gained by setting all siblings as disjoint in Table 5.6. The latter baseline employs the disjoint siblings assumption formulated by Schlobach [84].

Table 5.6: Performance of baselines

	dbpedia50			dbpedia100		
	prec	rec	F_1	prec	rec	F_1
Majority	0.926	1.000	0.962	0.915	1.000	0.956
Disjoint Siblings	0.926	0.850	0.886	0.933	0.850	0.890

Table 5.7 lists the number of axioms generated by the different inductive approaches while Table 5.8 provides the results of evaluating these axioms on all class pairs (P_{all}) against the gold standard. To properly compare them against the materialized gold standard, we also materialized all implicit disjointness axioms for the learning approaches and determined the evaluation measures on the materialized gold standard. In the tables, the basic association rule mining approach is referenced by *basic*, the negative association rule mining shows as *negative* while the correlation-based approach is given as *corr* followed by the employed threshold value.

Table 5.7: Number of axioms generated by inductive approaches exceeding threshold

	# axioms before materialization	# axioms after materialization
basic	39,169	46,885
negative	76	24,973
corr0.005	5,015	40,549
corr0.05	235	27,753

One first finding of the experiments is the fact that all inductive approaches outperform the baseline approaches with regard to precision although the baselines

Table 5.8: Results of inductive approaches on P_{all}

	dbpedia50			dbpedia100		
	prec	rec	F_1	prec	rec	F_1
basic	0.939	0.960	0.949	0.945	0.961	0.953
negative	1.000	0.536	0.698	1.000	0.542	0.703
corr0.005	0.959	0.843	0.897	0.963	0.847	0.901
corr0.05	0.991	0.589	0.739	0.992	0.594	0.743

already perform very well on the dataset. This can be attributed to the fact that both baselines make assumptions which do not hold on parts of the ontology and the dataset. For example, the disjoint siblings assumption is not valid for the classes describing roles as explained during the evaluation of the gold standard. It is also worth to point out that the increase in precision between the dbpedia50 and the dbpedia100 gold standard that is visible for all approaches is due to leaving out pairs for which the annotators were not able to find a full agreement.

Comparing the different inductive approaches, we see that the negative association rule mining approach is leading with respect to precision. However, it falls short in recall compared to the other methods. The correlation approaches are better regarding the recall for both thresholds and the two gold standard variants. These approaches have a slight disadvantage regarding precision nevertheless they show a better F-measure than the negative association rule mining approach. The basic association rule mining approach shows the best result in recall and leads with respect to the F-measure even if not showing the best precision. This advantage regarding the overall result is in line with the results of our previous experiments performed on DBpedia 3.5 [39] where the basic association rule mining approach also showed the best combined result of the different inductive methods. Furthermore, the association rule mining approaches have the advantage of being more extendable than the correlation-based ones. For example, considering not only axioms stating the disjointness of two atomic classes but axioms which include, e.g., unions of classes, the association rule mining approach can handle these scenarios efficiently by just relying on the algorithmic efficiency of the underlying association rule mining algorithm while the correlation-based methods have recompute all relevant combinations of classes. Given these results and observations including its considerably higher recall, we took the basic association rule mining approach as a representative of the inductive methods for further experiments. We evaluated this approach on the different sets of pairs to get a more detailed insight into the performance on these sets. The results of this evaluation are provided in Table 5.9. We see that the approach has problems telling disjoint and non-disjoint class pairs apart for the direct siblings pairs. However, considering the results of the gold standard creation, this lower performance on sibling class pairs is similar to the drastically lower inter-annotator agreement achieved for those class pairs. This leads to the conclusion that the decision regarding the disjointness of a class pair

Table 5.9: Results for the basic association rule mining approach. For the sake of clarity, results for P_{all} provided again.

		P_{all}	P_{direct}	$P_{indirect}$
dbpedia50	prec	0.939	0.666	0.941
	rec	0.960	0.743	0.960
	F_1	0.949	0.702	0.950
dbpedia100	prec	0.945	0.683	0.947
	rec	0.961	0.739	0.961
	F_1	0.953	0.710	0.954

is indeed more complex for sibling classes than for arbitrary class pairs. Since the number of direct sibling pairs is much more limited, it might be worth to manually cross-check results of the mining approach to reach a better result when applying inductive approaches in real-world use cases.

We compared the results of the basic association rule mining approach to the supervised learning approach implemented in the LeDA framework. For this comparison, we used the most current version of the LeDA framework.[4] As classifier, we applied the ADTree [41] algorithm as implemented by the Weka toolkit [46] which showed to perform well in previous experiments. The PROTON ontology (PROTo ONtology)[5] in combination with the gold standard created by Völker et al. [98] served as training dataset. More specifically, we used the PROTON gold standard which marked a class pair as disjoint if it was considered as disjoint by the majority of the annotators which is similar to the method employed for our dbpedia50 gold standard. The results of this evaluation are shown in Table 5.10.

Table 5.10: Results for LeDA on P_{all}.

		All	Lexical	Logical	Corpus
dbpedia50	prec	0.958	0.940	0.939	0.933
	rec	0.248	0.955	0.001	0.975
	F_1	0.395	0.947	0.002	0.953
dbpedia100	prec	0.961	0.945	1.000	0.939
	rec	0.247	0.957	0.001	0.975
	F_1	0.393	0.951	0.001	0.956

In these results, we see that there is only a minor difference in the performance on the dbpedia50 and the dbpedia100 datasets. More interesting is the low performance that the supervised approach achieves for certain feature choices. For the complete and the logical-only feature sets, the recall is much lower than for the other feature sets. Apparently, the logical features perform poorly on the dataset

[4] The latest version of LeDA is available from `https://code.google.com/p/leda-project/`.

[5] `http://www.ontotext.com/proton-ontology`

Table 5.11: Results for LeDA without ontology similarity feature

		P_{all}	P_{direct}	$P_{indirect}$
dbpedia50	prec	0.949	0.631	0.949
	rec	0.945	0.603	0.945
	F_1	0.947	0.617	0.947
dbpedia100	prec	0.955	0.651	0.955
	rec	0.946	0.606	0.946
	F_1	0.951	0.628	0.951

and also affect the overall performance when included into the complete feature set. To gain further insight, we analyzed the logical feature setting further. This analysis revealed the problem to be a specific characteristic of the DBpedia ontology. It contains direct subclass relations for all its classes to the top-level OWL class `owl:Thing`. Due to these statements, the ontology distance features employed by LeDA determined the same distance for almost all class pairs since the connection using `owl:Thing` as intermediate concept led to the lowest possible distance. This resulted in the ontology distance feature to provide close to no additional information. On the other side, the ontology distance showed to be the feature exposing the highest information gain in the PROTON ontology and thus the trained classifier greatly relied on this feature. Consequently, the classifier performed poorly on the largest part of the class pairs. To validate this finding, we excluded the ontology distance feature from training as well as evaluation. This increased the results for the logical features to a precision of 0.94 and a recall of 1.00 with an F-measure of 0.97. The results when applying all remaining features on the different sets of pairs are given in Table 5.11.

Compared to the inductive learning approach, we see that the supervised method generates disjointness axioms at a slightly higher precision on the pairs of P_{all}. At the same time the recall is lower which leads to a slightly better F-measure for the inductive approach. On P_{direct} the association rule mining approach shows minor advantages for precision and performs considerably better with respect to recall. For $P_{indirect}$ the situation is similar to the P_{all} case. From these results, the inductive approach seems to gain a better outcome in the more complex case of deciding whether two direct sibling classes are disjoint or not while the supervised method produces less wrong axioms when provided with arbitrary class pairs.

We had a closer look at the results to identify the main causes leading to wrongly or not at all generated axioms for both methods. For the inductive approach we identified several main sources of such errors. The first one was the existence of single-typed instances where the only additional types were the superclasses of their directly assigned ones. Classes without any instances assigned are the second problem since they are currently not treated separately from the other classes and thus can end up in being disjoint to all other classes in the ontology. Since the annotators in many cases evaluated on the intended semantics

of a class they were able to correctly assess the disjointness while the association rule mining approach did not have this possibility. One possible solution for this problem would be to filter out such instance-less classes beforehand so that they are not falsely defined to be disjoint to some other classes. This could be especially done in use cases where the approach is applied without manual intervention since instance-less classes are probably of less interest in these cases, e.g., since there are no instances whose class assertions can be checked for this class. Thus, one of the main advantages of the inductive approach also turns out to be one of its weaknesses. Its sole reliance on the available instance data makes it directly applicable to many datasets but at the same time shortcomings in these datasets lead to observable errors in the generated axioms. One possible way of coping with this kind of problems would be to include additional data sources. Such additional data sources could, e.g., be other Linked Data datasets whose instances are linked to the same set of classes as the original dataset or the inclusion of less structured sources like textual data as done in the LeDA framework. However, while the first option at least shows potential to be performed automatically, the second option would probably require more manual intervention and thus could reduce the self-sufficiency of the inductive approach.

The supervised approach typically suffers from weaknesses related to the natural language components providing the foundation of many employed features. During the evaluation of missing disjointness axioms, one typical problem was caused by similar class labels. Different features used the similarity of class labels as an indicator for classes not being disjoint. Such labels also led to class pairs falsely not being classified as disjoint. For example, the close lexical relation between the labels of the classes `Plant` and `Planet` led to very low edit distance and thus was a reason for not generating a disjointness axiom for this class pair. In other cases, some features wrongly inferred a lexical hyponymy relation between the labels which then prevented a disjointness axiom from being stated. An example for this kind of error is the class pair `Automobile` and `AutomobileEngine` in which the class labels share a head noun but are nonetheless disjoint. The lexical features employed in the supervised approach only rely on a very coarse approach of word sense disambiguation. In some cases this leads to wrongly assessing classes as similar and thus non-disjoint. For example, the classes `Station` and `RadioStation` were not classified as disjoint though the former class actually describes a public transport station which is obviously disjoint to a radio station.

5.4 Conclusion

In this chapter, we introduced unsupervised, inductive methods for extending an OWL ontology with learned class disjointness axioms. In contrast to previous works which used supervised approaches, these methods only rely on the instance data contained in a dataset for gaining information about the disjointness of different class pairs.

For evaluating these approaches, we created a high-quality gold standard for class disjointness in the DBpedia 3.7 ontology. We extensively described the methodology applied for creating it and identified problems which made it hard even for humans to correctly and coherently annotate the ontology like for the `Library` class. Furthermore, based on the insights gained from the gold standard creation, we showed that many problems found in the ontology could be caused by the crowd-sourced creation approach which led to a mixture of different modeling approaches in the ontology. One possibility to solve or at least limit such problems could lie in providing a comprehensive guide for possible contributors to the ontology which clearly states the most common problems and advises one standard way of handling this problem. This would help to further standardize the modeling approach in the ontology which would finally lead to an ontology being more useful for reasoning-based use cases. The problems identified in this work could be used as a starting point for such a guideline.

Regarding the actual evaluation of learning approaches, our experiments showed that inductive methods are indeed promising ways of discovering class disjointness in instance data. The inductive approaches each showed different advantages which might qualify them depending on the specific use case. While the negative association rule mining approach reached the highest precision for its generated disjointness axioms but suffered from a low recall, the correlation-based method was more balanced between precision and recall. Moreover, the correlation-based approach does not require to retrieve all instance assignments from the dataset but only those of classes for which additional insights are desired. Finally, the basic association rule mining-based approach reached the highest recall value at the expense of a slightly lower precision than the other methods but in total delivered the most balanced result which is also visible from the achieved F-measure value.

In a further evaluation, we compared the basic association rule mining method to the state-of-the-art approach for learning class disjointness axioms implemented by the LeDA framework. These experiments showed up different advantages and shortcomings. The inductive approach, solely relying on the instance data of the dataset, had problems generating correct disjointness axioms in cases where only few or no instances at all were assigned to the relevant classes. However, this allowed for learning disjointness axioms without any additional information or manual intervention. In contrary, the supervised approach exhibited advantages for sparsely populated classes and was more precise when deciding about the disjointness of arbitrary class pairs. Nonetheless, our experiments also showed a number of shortcomings for the supervised method. Firstly, the problem of disambiguating classes to real-world concepts based on their labels, which is implicitly required by many employed features, demonstrated to introduce errors into the process. Secondly, its supervised nature posed additional challenges. Beside the requirement for a training dataset, different characteristics of the training and the evaluation dataset initially led to subpar results. To improve the results, the feature selection had to be adapted to the specific datasets. In particular, this proves that the results for the supervised learning approaches greatly rely on the similarity of the involved

datasets. In experiments not described here, we explored the application of transfer learning to adapt the feature selection to better suit both datasets which helped to improve the results. Nevertheless, even if transfer learning helps to diminish this problem, the supervised approach still requires more external data and a training dataset, making it more dependent on manual assistance.

All in all, we consider the proposed inductive methods and especially the basic association rule mining-based one well-suited for providing a base disjointness-enriched ontology. This ontology can be processed further automatically or could be manually refined by human ontology engineers. It can also provide a basic documentation of the typical usage of classes in the dataset.

Chapter 6

Inductive Learning of Property Axioms

In the previous chapter, we presented work on inductively learning class disjointness for enriching inexpressive schemas. The goal of these approaches is to foster more semantics in the growing Linked Data cloud and thus enable the application of more approaches for cleaning the data and provide a more extensive insight into the usage of classes and properties in the dataset. However, though class disjointness is already helpful and definitely an indispensable step on the way to more helpful and flexible Linked Data, it is not the final destination. Class disjointness only gives additional information about classes and their usage. Regarding the properties in the dataset, additional information is only available via the indirection of using domain and range axioms. Additional property-centric axioms would give more direct information about properties and their usage. This would not only be helpful to detect data errors or document the dataset but also for pioneering the usage of inference-enabled query answering. More knowledge on the characteristics of properties enable additional possibilities for the user to retrieve data. Other use cases are also available. For example, from additional knowledge about properties, it would be possible to support data engineers with proposals which property to use for the current subject and which not. Work in this direction has been started by Scheglmann et al. [83] who infer typical usages of the data to support programmers in working with RDF data or by the approaches introduced by Abedjan and Naumann [1].

Since its release in 2009, the ontology language OWL 2 provides a great variety of axioms on top of those already available in its predecessor OWL. Especially with respect to property axioms, OWL 2 is considerably more powerful than before and allows, e.g., to express the asymmetry of properties as well as their reflexivity or irreflexivity. Furthermore, properties can be defined to be disjoint to other properties which means that between two instances there cannot hold both of the disjoint properties at once. For instance, this can be useful to discover wrongly linked instances in a dataset or for supporting data engineers by eliminating instances from

the set of possible property objects.

Unfortunately, like most schemas do not contain disjointness axioms, they are only sparsely populated by axioms other than class subsumption. Thus, many axiom types available in OWL 2 are rarely used in ontology. However, even property-centric constructs from OWL like domain or range restrictions are not always contained in the ontology. As a result, the reliance on more advanced constructs in applications is not yet worthwhile. On the other side, due to this limited application support for advanced axioms, there is only very little incentive to add such axioms to an ontology. Since the creation of highly expressive ontologies is a time-consuming and complex task, we have a classical chicken-or-egg problem.

Given its large amounts of data and also its high coverage of different domains, the Linked Data cloud gives reason to believe that more expressive schema for its datasets could find faster adoption in applications than for many other datasets. Due to its large datasets in terms of instance counts, it also qualifies as a dataset for applying inductive methods to. Based on the approaches presented before, namely the work by Völker and Niepert [96] and the previously presented work on learning disjointness axioms, we already are able to learn large parts of axiom types supported by the OWL 2 EL profile. In the following, we extend the approach to support even more expressivity. However, since the large amounts of data in the Linked Data cloud also pose a challenge to inference systems, we limit ourselves to those axiom types available in the OWL 2 RL fragment which combines means for deductive query answering and the ability to detect inconsistencies in the data.

This chapter is based on our work presented in 2012 [40]. We first report on the extensions to the inductive approach for closing the gap between the EL fragment of OWL 2 and the RL fragment in Section 6.1. Afterwards, in Section 6.2, we report on the evaluation of our approaches and also object property transitivity which was unevaluated before. We also give details on our experience from using crowd-sourcing approaches for evaluating the results before finally concluding this chapter in Section 6.3.

6.1 Approaches

The approaches which we present in the following, extend our previously proposed association rule mining approaches towards supporting large parts of the OWL 2 RL fragment. As for the basic disjointness approach, we assume a Linked Data dataset to be given which contains instance data. To simplify the following descriptions, we assume the data to be provided by a triple store which supports the retrieval of data by means of SPARQL queries. However, the approaches are not limited to this kind of data access and adapting the methods to other accessing methods is straightforward. This might especially be advantageous for improving the performance of the data gathering step when creating transaction tables from instance data.

6.1.1 Terminology Acquisition

The overall process which finally leads to the desired axioms follows the same basic steps already proposed by Völker and Niepert and depicted in Figure 3.1. As described before, we start with gathering information about classes, instances and properties contained in the dataset. Through assigning unique identifiers to each of these entities, we lay the foundation to create the required transaction tables later-on. Furthermore, during the assembly of these identifier lists, we also introduce unique identifiers for the complement of properties similar to what we did for class complements in Chapter 5. We refer to these identifiers as *property complement identifiers*. In addition, we generate identifiers which represent the inverse of a property, i.e., for each object property p that gets an identifier assigned, we also assign a unique identifier for the inverse p^{-1} which we refer to as *inverse property identifier*. For inverse property identifiers, we also introduce identifiers for their complement $\neg p^{-1}$. Finally, the list of constructs which get unique identifiers assigned is completed by some more complex ones:

- for each property r, we introduce a *functionality identifier* and an *inverse functionality identifier* represented by $t_{sub\leq 1}(r)$ and $t_{obj\leq 1}(r)$, respectively

- for each property r, we introduce *domain identifiers* and *range identifiers* which we depict by $\exists r.\top$ and $\exists r^{-1}.\top$ as already known from the original approach.

We will introduce the semantics of these identifiers in the course of this section. Since we are especially looking into the generation of property axioms using the approaches proposed in the following, special attention is payed to generate possible pairs of instances. Given the fact that typical Linked Data datasets contain extremely large amounts of instances, the number of pairs of instances might be prohibitively high. Thus, we limit ourselves to the generation of instance pairs which have at least one property connecting them. The initial generation of instance pairs can be performed without querying the actual dataset based on the set of instances retrieved before. However, to check the existence of an arbitrary property between both instances, we have to involve the server. To enable the query engine to optimize the required query as much as possible, we do not use SPARQL `SELECT` queries for this purpose but pose queries adhering to the pattern

```
ASK {<INSTANCE URI 1> ?p <INSTANCE URI 2>}
```

where `INSTANCE URI 1` and `INSTANCE URI 2` are replaced by the URIs of the currently considered instances. This `ASK` query results in a boolean result and the instance pair is only added to the set of relevant instance pairs if the query raises a positive result meaning that there is at least one property connecting both instances. After assigning each relevant instance pair a unique identifier, we end up with a list as shown in Table 6.1.

Table 6.1: Examples for instance pairs. The URI prefix `http://dbpedia.org/resource/` has been omitted for all instances.

Pair	Instance 1	Instance 2
1	Pepsi	Pepsi
2	Pepsi	Coca-Cola
3	Coca-Cola	Pepsi
4	Pepsi	United_States
...	...	...

6.1.2 Creation of Transaction Tables

Based on this general terminology acquisition step, we then generate transaction tables for each axiom type to generate which we describe in the following. For axiom types that can be mined from the same transaction table, we give a joint description. To avoid redundancy, we sometimes refer to a *default transaction table* in the following when we mean a transaction table whose rows are representing instance pairs and where each row contains a property identifier if the instance pair is connected by the corresponding property in the dataset.

Object Property Symmetry, Asymmetry and Inverse

For supporting object property symmetry, asymmetry and inverse, we generate a transaction table whose rows each represent an instance pair discovered during the terminology acquisition as in the default transaction table. For each instance pair, we execute the SPARQL query

```
SELECT DISTINCT ?p
WHERE {
  <INSTANCE URI 1> ?p <INSTANCE URI 2>
}
```

which gives us all properties which are stated to hold between both instances. Based on this set of properties, we assemble the table row by adding the property identifier for each property contained in the set. Up to this step, the result is the aforementioned default transaction table. In addition, we extend each row with the property complement identifier for each known property not contained in the query result set. Similar to adding class complement identifiers for learning class disjointness, this is influenced by the open-world assumption as described in Chapter 3. Finally, we also perform the query

```
SELECT DISTINCT ?p
WHERE {
  <INSTANCE URI 2> ?p <INSTANCE URI 1>
}
```

and add the inverse property identifier for each property contained in the result.

Table 6.2: Serialization of transaction table for object property symmetry. × marks all properties used between the given pair of instances.

	related[20]	origin[22]	related⁻[21]	origin⁻[23]	
(Pepsi, Pepsi)					...
(Pepsi, Coca-Cola)	×		×		...
(Coca-Cola, Pepsi)	×		×		...
(Pepsi, United_States)		×			...
(United_States, Pepsi)				×	...
...	...	...	...	...	...

By ordering the instance pairs in a way that for all instance pairs (i_1, i_2) and (i_2, i_1) occur consecutively, the second query would be posed either way and thus does not lead to reduced performance.

For the example data given in Table 6.1, we would generate a transaction table as shown in Table 6.2.

Object Property Functionality and Inverse Functionality

The transaction tables for mining property functionality and inverse functionality axioms are *not* based on the default transaction table. Instead, the transaction tables are created independently from the default set and moreover both axiom types get own transaction tables which are similar to each other. The rows of the transaction tables are formed by all single instances extracted from the dataset. For functionality, we add the functionality identifiers to the transaction if the instance is used as most once in the subject position with the corresponding property. Similarly, for inverse functionality the associated identifier is added if the instance is used at most once in the object position with the corresponding property.

In addition to these identifiers, in the functionality table, we add the domain identifier for a property r to the transaction if the current instance is actually used at subject position of r. Likewise, for inverse functionality, we add the range identifier if the instance is used at least once at object position of r.

Object Property Subsumption and Disjointness

The procedure for mining object property subsumption using statistical schema induction has already been described by Völker and Niepert in their initial work and we already described it in Section 3.3. The transaction table to use for this purpose is the default transaction table without any additional items added. To

extend this mining procedure into the direction of also detecting disjoint object properties, we add property complement identifiers to the transactions. For each instance pair, we add the property complement identifier $\neg r$ if property r does not known to hold between this instance pair.

Reflexive and Irreflexive Object Properties

Though OWL 2 RL does not support the definition of properties as reflexive or irreflexive, we are aware that in some scenarios this type of axioms might be useful. Therefore, we here provide the transaction table for reflexive and irreflexive properties that again has rows representing instance pairs. For this purpose, we introduce another identifier $t_{a=b}$ which is independent from specific properties and instances and represents the equality of the instances contained in the corresponding instance pair. This means that this identifier is added to the transactions (a, b) for which $a = b$ holds. Furthermore, for the detection of reflexive properties the property identifiers are added to the transactions while for detecting irreflexive properties, we add the property complement identifier. Transaction tables containing these items enable us to capture the relation between the properties holding between two instances and their equality.

As a final wrap-up of the different transaction table types, we provide a summary of all detectable property axiom types and the corresponding transaction table structure in the first two columns of Table 6.3. The last four rows of this table also contain the property-related axiom types whose generation already has been proposed by Völker and Niepert before. We added these as a central reference of all property-centric axiom types supported by our inductive approaches.

6.1.3 Association Rule Mining and Axiom Generation

After creating the transaction tables for all desired types of axioms, the next step is the actual detection of frequently occurring itemsets in the transactions and, based on these, finally the deduction of association rules. As we did before, we apply the Apriori algorithm for performing the necessary rule mining step, nevertheless, since we treat this step as a black-box step, the actual algorithm employed for doing frequent itemset discovery and association rule mining can be exchanged without the need of further modifications of the approach.

For our example, this step could lead to the generation of association rules like:

$$\texttt{related}^{-}\texttt{[21]} \rightarrow \texttt{related[20]}(50, 100)$$
$$\texttt{related[20]} \rightarrow \texttt{related}^{-}\texttt{[21]}(50, 100)$$

After the mining step, we gather association rules which adhere to the patterns given in the third column of Table 6.3 which each correspond to one specific axiom type. For example, for object property symmetry, we try to observe patterns like $\{p\} \Rightarrow \{p^{-1}\}$, i.e., the first identifier is a normal property identifier and the second one an inverse property identifier for the same property. Using the transformation

Table 6.3: Summary of transaction table generation for property axioms. Each row in the transaction table either corresponds to one instance $a \in N_I$ or a tuple of instances $(a, b) \in N_I \times N_I$. A concept item C_i is added to the transaction if $a \in C_i^{\mathcal{I}}$ for the corresponding instance a. A property item r_i is added to the transaction if the corresponding property holds between the instance tuple, i.e., $a\ r_i\, b$, the property complement item $\neg r_i$ is added if $a\, r_i\, b$ does not hold. r_i^{-1} is added if $b\ r_i\, a$, $\neg r_i^{-1}$ if not $b\ r_i\, a$. $t_{a=b}$ is contained in a transaction if $a = b$. $r_i \circ r_i$ is contained in a transaction if $a\ r_i\ x$ and $x\ r_i\ b$ for an arbitrary instance $x \in N_I$. $\exists r.\top$ is added if the instance a is used as subject of r, $\exists r^{-1}.\top$ if it is used as object. The items $t_{sub_{\leqslant 1}}(r)$ and $t_{obj_{\leqslant 1}}(r)$ are added to a transaction if the current instance a is used at most once as subject or object of the property r.

Axiom Type	Transaction Table Row	Association Rule
$\mathrm{Sym}(r)$	$(a,b) \to r_1, \ldots, r_n, r_1^{-1}, \ldots, r_n^{-1}$	$\{r\} \Rightarrow \{r^{-1}\}$
$r_i \sqsubseteq \neg r_j$	$(a,b) \to r_1, \ldots, r_n, \neg r_1, \ldots, \neg r_n$	$\{r_i\} \Rightarrow \{\neg r_j\}$
$\mathrm{Ref}(r)$	$(a,b) \to r_1, \ldots, r_n, t_{a=b}$	$\{t_{a=b}\} \Rightarrow \{r\}$
$\mathrm{Irr}(r)$	$(a,b) \to \neg r_1, \ldots, \neg r_n, t_{a=b}$	$\{t_{a=b}\} \Rightarrow \{\neg r\}$
$\mathrm{Asy}(r)$	$(a,b) \to r_1, \ldots, r_n, \neg r_1^{-1}, \ldots, \neg r_n^{-1}$	$\{r\} \Rightarrow \{\neg r^{-1}\}$
$r_i \sqsubseteq r_j^{-1}$	$(a,b) \to r_1, \ldots, r_n, r_1^{-1}, \ldots, r_n^{-1}$	$\{r_i\} \Rightarrow \{r_j^{-1}\}$
$\mathrm{Fun}(r)$	$a \to \exists r_1.\top, \ldots, \exists r_1.\top, t_{sub_{\leqslant 1}}(r_1), \ldots, t_{sub_{\leqslant 1}}(r_n)$	$\{\exists r_1.\top\} \Rightarrow \{t_{sub_{\leqslant 1}}(r)\}$
$\mathrm{InvFun}(r)$	$a \to \exists r_1^{-1}.\top, \ldots, \exists r_n^{-1}.\top, t_{obj_{\leqslant 1}}(r_1), \ldots, t_{obj_{\leqslant 1}}(r_n)$	$\{\exists r_1^{-1}.\top\} \Rightarrow \{t_{obj_{\leqslant 1}}(r)\}$
$r_i \sqsubseteq r_j$	$(a,b) \to r_1, \ldots, r_n$	$\{r_i\} \Rightarrow \{r_j\}$
$r \circ r \sqsubseteq r$	$(a,b) \to r_1, \ldots, r_n, r_1 \circ r_1, \ldots, r_n \circ r_n$	$r_i \circ r_i \Rightarrow r_i$
$\exists r.\top \sqsubseteq C$	$a \to C_1, \ldots, C_l, \exists r_1.\top, \ldots, \exists r_n.\top$	$\{\exists r.\top\} \Rightarrow \{C\}$
$\exists r^{-1}.\top \sqsubseteq C$	$a \to C_1, \ldots, C_l, \exists r_1^{-1}.\top, \ldots, \exists r_n^{-1}.\top$	$\{\exists r^{-1}.\top\} \Rightarrow \{C\}$

pattern, we generate OWL 2 axioms which are annotated with the confidence value for the base association rule. The confidence value allows to filter the generated axioms further before finally adding them into the overall result ontology.

If an association rule fits in a specific pattern and exceeds the confidence threshold, the symmetry axiom matching this association rule is generated. For the association rule pattern given above, the corresponding symmetry axiom would be $\mathrm{Sym}(p)$ annotated with the confidence value of the association rule. In our example, only the second association rule fits into the required pattern. Since the confidence is 100, which is the maximum confidence given a representation in percent, it would not be filtered out by any settings for confidence thresholds. The resulting axiom is a symmetry axiom $\mathrm{Sym}(\texttt{related})$ for the property `related`.

6.2 Evaluation

In this section, we describe the evaluation of the approaches for generating property-related axioms. First of all, we give details on the settings under which we performed the evaluation. Then, we present the results of the actual evaluation which consisted of two parts: one evaluation conducted by ontology engineering experts and another one performed by laymen via crowd-sourced tasks.

6.2.1 Settings

We extended our implementations already contained in the GoldMiner tool by the additional learning approaches presented in the previous section. This implementation was applied to the DBpedia dataset in version 3.7 for generating the transaction tables. Continuing the known pipeline structure, we used the Apriori implementation by Borgelt for finding association rules in the tables. Finally, we mined the discovered association rules for the patterns defined in Table 6.3 and produced the corresponding OWL 2 axioms.

We performed all experiments applying three different confidence thresholds, namely 0.5, 0.75 and 1.0, to examine in how far higher confidence thresholds lead to a higher degree of correct axioms. In contrast to our previous experiments on learning class disjointness axioms, this time the support threshold was fixed at an absolute value of 1 which actually means that we considered all association rules without enforcing a certain support level. The reason behind this choice is that the very specific patterns defined for producing a certain axiom type already led to a low number of valid association rules to be produced. Thus, a lower support threshold was used to raise the number of produced axioms.

As in our previous experiments, we discovered the terminology extraction and the transaction table generation phase to be the most time-consuming tasks. Heavily relying on the performance of the triple store providing the data and processing the SPARQL queries, this step took up to several hours to complete. The size of the final transaction tables was between 6 MB for the domain and range table and

22 GB for the property disjointness table. Nevertheless, the execution of the association rule mining phase on these datasets stayed in the magnitude of minutes. For example, on a 17 million lines transaction table, this phase was completed in 20 minutes.

The numbers of axioms generated for the association rules are provided in Table 6.4 for each type of axiom. In the following evaluation parts, we are concentrating on the axiom types supported by OWL 2 RL and newly introduced in this work. Thus, characteristics like property subsumption and reflexivity have not been generated.

Table 6.4: Total number of generated axioms for given confidence thresholds

minimum confidence	0.5	0.75	1.0
AsymmetricObjectProperty	435	410	384
DisjointObjectProperties	180,876	180,850	178,660
FunctionalObjectProperty	354	268	67
InverseObjectProperties	6	3	0
SymmetricObjectProperty	4	2	0
TransitiveObjectProperty	246	219	141
Total	181,921	181,752	179,252

In contrast to our evaluations on class disjointness, we did not create a complete gold standard for all possible axiom types. This would have been unrealistic due to the number of axiom types and the number of potentially valid axioms in the ontology. Instead, we evaluated the axioms generated by our approaches for their correctness.

6.2.2 Expert Evaluation

For the expert evaluation, we randomly chose a subset of 40 axioms per property axiom type or all axioms if less than 40 were generated. Each of these axioms was evaluated by three ontology engineers for correctness on a two-valued scale, consisting of the categories *correct* and *wrong*. During the evaluation, the axioms were presented to the experts in OWL 2 Functional Syntax. Additionally, for each axiom type, we defined a natural language sentence expressing the semantics of the axiom. For example, an asymmetry axiom for the property `thirdDriverCountry` was given by the sentence

> If A `thirdDriverCountry` B holds
> then B `thirdDriverCountry` A must not hold

Furthermore, the annotators had access to the DBpedia page describing the relevant object properties. These pages provided the comments and further information on the object properties which could help the annotators to decide in more complex

Table 6.5: Results of the expert-based evaluation for subsets of generated axioms with **confidence threshold 0.5**. Listing the number of axioms evaluated by the annotators (# eval), axioms rated as correct (# corr), accuracy (Acc.) and the inter-annotator agreement by means of observed agreement and Fleiss' κ.

Axiom Type	# eval	# corr	Acc.	Observed	κ
AsymmetricObjectProperty	40	37	0.93	0.94	0.58
DisjointObjectProperties	40	39	0.98	0.99	-0.01
FunctionalObjectProperty	40	9	0.23	0.62	0.38
InverseObjectProperties	6	2	0.33	0.87	0.68
SymmetricObjectProperty	4	3	0.75	1.00	1.00
TransitiveObjectProperty	40	2	0.05	0.82	0.12
Total	170	92	0.54	0.85	0.73

cases. Finally, since in some cases the naming of the object property alone was not enough to understand its meaning, the annotators had access to a subset of the results of the SPARQL query

```
SELECT ?a ?b WHERE {?a <PROPERTY URI> ?b}
```

which provided insights into the actual usage of the property.

Based on the results of the expert evaluation, we computed the accuracy and the inter-annotator agreement for the evaluated axioms. Due to the lack of a complete gold standard, it was not possible in these experiments to determine the precision and recall values.

The results of the expert evaluation are given in the Tables 6.5, 6.6 and 6.7. Each table contains the results for a single confidence threshold and lists the numbers of evaluated and correct axioms, the accuracy of the learned axioms and the inter-annotator agreement determined by means of observed agreement and also using Fleiss' κ. For the inter-annotator agreement, we again see the problem of low chance-corrected agreement while the actual observed agreement is very high or almost perfect as we already discovered in Chapter 5. Since the evaluation was not split by axiom type but all axiom types were evaluated at once, it is improbable that the annotators considered independent probability distributions for each axiom type during the annotation as it would be assumed by computing Fleiss' κ independently for each axiom type. More probably, they had a combined distribution for all axiom types which means that the more relevant κ value is the one provided in the "Total" column.

Based on the expert-based evaluation, we can already draw a few conclusions. First of all, it is noticeable that changes regarding the confidence values did not have great influence on the accuracy of our results and thus the greater part of the confidence values did not react considerably to changes to the confidence thresholds. Accuracy values which were already high for lower thresholds only increased slightly if at all. Therefore, being extracted by the approach seems to be a strong

Table 6.6: Results of the expert-based evaluation for axiom subset with **confidence threshold 0.75**

Axiom Type	# eval	# corr	Acc.	Observed	κ
AsymmetricObjectProperty	24	22	0.92	0.93	-0.03
DisjointObjectProperties	40	39	0.98	0.98	-0.01
FunctionalObjectProperty	23	9	0.39	0.59	0.33
InverseObjectProperties	2	2	1.00	1.00	1.00
SymmetricObjectProperty	2	2	1.00	1.00	1.00
TransitiveObjectProperty	35	1	0.03	0.83	0.15
Total	126	75	0.60	0.86	0.72

Table 6.7: Results of the expert-based evaluation for axiom subset with **confidence threshold 1.0**. No inverse or symmetric properties have been generated for this threshold.

Axiom Type	# eval	# corr	Acc.	Observed	κ
AsymmetricObjectProperty	21	19	0.90	0.92	-0.04
DisjointObjectProperties	40	39	0.97	0.98	-0.01
FunctionalObjectProperty	8	2	0.25	0.46	0.16
TransitiveObjectProperty	25	0	0.00	0.81	0.06
Total	94	60	0.64	0.87	0.72

evidence for correctness for the axiom types of object property asymmetry and also object property disjointness. Also symmetric object property axioms were reliable when generated. The most unreliably generated axiom types according to our evaluation were functional object properties as well as transitive object properties. For functional object properties, we also discovered the lowest of the achieved inter-annotator agreements. This leads to the conclusion that the detection of this axiom type not only causes problems in our approach but also it is more complex for humans to recognize.

Nevertheless, an especially visible point is the extremely low accuracy for the object property transitivity which cannot be attributed to an especially hard task since the inter-annotator agreement was high. For further investigating this aspect, we had a look at the generated axiom like the axiom assigning transitivity to the property `birthPlace` which is obviously wrong. To examine the data on which the axiom was discovered, we executed the SPARQL query

```
SELECT DISTINCT ?a ?b ?c
WHERE {
  ?a <http://dbpedia.org/ontology/birthPlace> ?b.
  ?b <http://dbpedia.org/ontology/birthPlace> ?c.
  ?a <http://dbpedia.org/ontology/birthPlace> ?c
}
```

```
LIMIT 100
```

which gave us a list of instances connected by the `birthPlace` property exhibiting the pattern of transitivity. For example, the result list contained the instances `Brian_Vandborg`, `Denmark` and `Peter_Snejbjerg` being connected as depicted in Figure 6.1.

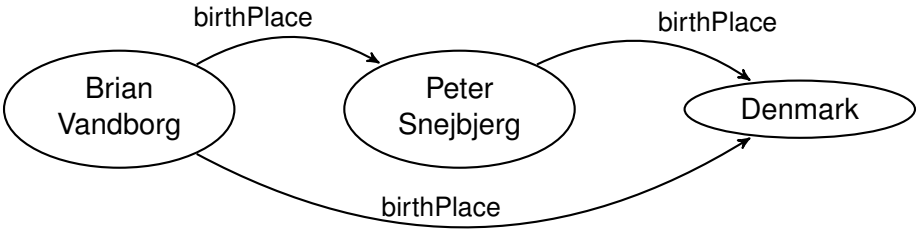


Figure 6.1: Graphical representation of triples leading to wrong transitive property axioms.

Since the properties between those instances were verifiably included in DBpedia, we traced this error further. A closer look at the Wikipedia article disclosed that *Brian Vandborg* was born in *Snejbjerg, Denmark*. In this case, we could identify two sources for the errors. First, the word *Snejbjerg* was linked incorrectly to the person *Peter Snejbjerg* and, second, the DBpedia parser extracted *Denmark* as a `birthPlace` on its own. Actually, in an RDF dataset, the reference to the city alone provides enough information to unambiguously identify it and the additional specification of the country is redundant.

We also identified more similar errors in DBpedia leading to many errors in our learned transitivity axioms. Not all of these were traceable to a Wikipedia error which means that the DBpedia extractor itself introduced errors in the DBpedia data. However, this illustrates that our learned axioms are helpful for discovering errors in datasets. Errors in Wikipedia and DBpedia are also the reason for our approach not to generate any irreflexivity axioms since many instances have erroneous self-references.

6.2.3 Crowd-Sourced Evaluation

Due to the high effort of letting experts evaluate the generated axioms, the number of checked results was limited for the expert-based evaluation. Thus, for complementing the first results, we performed a more extensive study of our results by means of crowd-sourcing.

The idea of *crowd-sourcing* is to divide a larger task into a large number of smaller tasks, the so-called microtasks. These microtasks are distributed to a pool of workers which can participate in the overall task by completing one or multiple microtasks in exchange for a small payment per completed microtask. Finding the

workers is done via services such as Amazon Mechanical Turk[1] which provide a platform for requesters to provide their microtasks, called *Human Intelligence Task* (HIT) by Amazon, and workers to choose the tasks to work on. Since the workers participating in the tasks are unknown to the requester one fundamental problem is how to assure high-quality results. The incentive for completing the task being the motivation of the workers, they might be tempted to just complete as many tasks as possible without taking care of completing them well. Hence, quality assurance is an important step when crowd-sourcing tasks and different approaches for doing this have been proposed as summarized by Allahbakhsh et al. [4].

There also exist some services which specialized on providing quality-assurance services for crowd-sourcing tasks like CrowdFlower[2] which we chose for our further evaluation. In the categorization of quality assurance methods made by Allahbakhsh et al., CrowdFlower belongs to the *Ground Truth* category. Approaches belonging to this category require a subset of the data annotated with gold standard answers, called ground truth. To assess the reliability of the workers, tasks whose answer is known from this gold standard are added to the tasks where the answer is unknown. The reliability of workers can then be assessed from the deviation of a worker's answers from the gold standard answers and unreliable workers can be excluded from further participating in tasks. Only providing the quality assurance service, the actual searching of workers was delegated by CrowdFlower to Amazon Mechanical Turk.

For providing a ground truth, we took the axioms for which all annotators agreed on the correctness or non-correctness during the expert-based evaluation. In addition to these ground truth axioms, we randomly chose generated axioms from the different confidence levels to let them evaluate by the workers. Since we could not expect the crowd workers to have knowledge about ontologies and the semantics of the specific axioms, we hid the actual axioms from them and only textual representations as shown in Figure 6.2 were presented to the workers. We also provided a possibility to write additional explanations regarding the single decisions.

Figure 6.2: Axiom evaluation task as presented to the crowd-evaluation participants

[1] http://www.mturk.com
[2] http://www.crowdflower.com/

This representation was accompanied by a description of the overall task which was clarified by some examples for the evaluation tasks. Similar to the experts, the crowd-evaluation participants had access to usage examples which were retrieved from the DBpedia dataset as described for the expert-based evaluation. Each microtask given to the workers consisted of five different axioms. The actual combination of axioms in the microtasks was determined automatically by the CrowdFlower system. Due to redundancy which was introduced during the quality assurance measures by CrowdFlower, the number of workers for the single axioms varied between three and ten workers.

The results of this part of the evaluation are presented in the Tables 6.8, 6.9 and 6.10 where each table contains the result for a certain confidence level. It is important to note that the sets evaluated during the crowd-based and the expert-based evaluation were disjoint except for the ground truth axioms occurring in the tasks. Due to the low number of axioms generated for inverse object property and object property symmetry axioms, this led to none of these axiom types being evaluated in this evaluation phase. Furthermore, the inter-annotator agreement based on Fleiss' κ could not be calculated because of the varying number of workers for the different axioms. Thus, we grouped the axioms by the number of evaluations and computed the observed agreement for each of these groups then we averaged the observed agreements.

Table 6.8: Results of crowd-sourced evaluation for subsets of generated axioms with **confidence threshold 0.5**. Axioms evaluated by experts are not contained in the crowd evaluation, '-' shows that all generated axioms have been evaluated by experts. "Mean Observed" is averaged observed agreement.

Axiom Type	# eval	# corr	Acc.	Mean Observed
AsymmetricObjectProperty	98	96	0.97	0.89
DisjointObjectProperties	261	258	0.99	0.92
FunctionalObjectProperty	175	48	0.27	0.50
InverseObjectProperties	-	-	-	-
SymmetricObjectProperty	-	-	-	-
TransitiveObjectProperty	159	32	0.20	0.52
Total	697	437	0.63	0.74

A major problem during the crowd-sourced evaluation was posed by the naming of the DBpedia properties since many of them were named confusingly and thus it was hardly possible to deduce their correct usage only based on their name. For example, the object property `training` was used in the dataset for connecting athletes or sports teams to their training locations. Solely based on the property name, many workers expected the property to be used for connecting trainer and trainee, as we found out by analyzing the comments provided by some workers.

Having a look at the full results, we find them to be very similar to those gained

Table 6.9: Results of crowd-sourced evaluation for axiom subset with **confidence threshold 0.75**

Axiom Type	# eval	# corr	Acc.	Mean Observed
AsymmetricObjectProperty	93	92	0.99	0.89
InverseObjectProperties	-	-	-	-
DisjointObjectProperties	261	258	0.99	0.92
TransitiveObjectProperty	141	29	0.21	0.52
FunctionalObjectProperty	138	40	0.29	0.50
SymmetricObjectProperty	-	-	-	-
Total	633	419	0.66	0.74

Table 6.10: Results of crowd-sourced evaluation for axiom subset with **confidence threshold 1.0**. No inverse or symmetric properties have been generated for this threshold.

Axiom Type	# eval	# corr	Acc.	Mean Observed
AsymmetricObjectProperty	89	89	1.00	0.89
DisjointObjectProperties	259	256	0.99	0.92
TransitiveObjectProperty	88	19	0.22	0.52
FunctionalObjectProperty	26	9	0.35	0.50
Total	462	373	0.81	0.81

by the expert-based evaluation. Thus, we are confident that results of the expert-based evaluation not only hold for the evaluated subset but also for the whole set of axioms. To further confirm this, we let an expert assess 190 axioms which were previously only evaluated by crowd workers. The expert agreed on 80% of the axioms with the assembled assessment of all crowd-workers.

6.3 Conclusions and Contributions

In this chapter, we presented additional inductive methods based on association rule mining for learning axioms from instance data. Through this extension, we are now able to generate or enrich ontologies up to the OWL 2 RL fragment of the OWL 2 ontology language which combines both a comparably high expressivity and at the same time desirable computational properties. In our experiments, we showed our methods to deliver promising results on the real-world dataset DBpedia. Furthermore, our evaluation revealed two further points. First, though the DBpedia ontology is created in a crowd-sourcing effort, it is not easy to understand for non-experts. Experts also have to go an additional mile to determine the actual meaning of a property. This is especially caused by incoherent or misleading naming of its entities which confirms further confirms similar findings of Chapter 5. In our opinion, this again urges to create a guideline for naming and modeling style which can be used by the ontology contributors as a reference. Secondly, we found evidence for the usefulness of learned axioms in a data debugging scenario. As shown in the example of the wrong transitivity of the `birthPlace` property, axioms help to find common errors in the data more rapidly because they might overstate patterns what makes common data errors easier to spot. Given these results, a two-stepped data debugging seems to be appropriate where the first step consists of a short evaluation of the learned axioms for obviously erroneous axioms which hint to a wide-spread data error. Afterwards, in a second step the remaining learned axioms can be used to find instances which do not adhere to these axioms.

After all, our main contribution in this chapter is the extension of the previously presented inductive approaches to support additional axioms from the OWL 2 RL fragment which enables more expressive query answering and, as also demonstrated during the evaluation of the approaches, gives more opportunities to detect errors in the data.

Part II

Logical Debugging of Linked Data

Chapter 7

Generating Incoherence Explanations for Learned Axioms

As we showed in Part I, we are able to automatically learn a schema from a dataset which captures the patterns contained in it. The evaluation of the generated axioms against the manually created gold standards showed the axioms to be of acceptable quality but there are still axioms considered wrong by the human annotators. Thus, manually improving the ontology would increase its usability, especially as a documentation of the dataset and the vocabulary's proper usage. To support a human in this improvement process, the notion of incoherence as introduced in Section 2.1 can be used. Incoherence is often considered as a pointer to problems in ontologies since unsatisfiable classes are rarely introduced on purpose. Furthermore, fixing incoherence renders the ontology more suitable in reasoning scenarios since, e.g., for standard approaches it is not possible to infer usable knowledge about incoherent classes since all incoherent classes (and properties) are treated as equivalent to each other according to DL semantics. However, the mere detection of incoherent classes is not enough. In fact, the sets of axioms leading to incoherent classes, most commonly referred to as *explanations* or *justifications*, are of more interest since the incoherent classes are only their symptoms. However, our experiments show that tools for computing explanations struggle with learned ontologies. For example, the well-known OWL reasoner Pellet [88], which is the only reasoner directly supporting the generation of explanations for inferences made from an ontology, showed to be unstable and slow when trying to compute all explanations for a given inference on our learned ontologies. Hermit [44], being another well-known OWL reasoner, does not support the explanation generation at all and combining it with so-called black-box approaches for generating explanations turned out to perform very poorly on ontologies learned by the methods described before. We attribute this to the fact that learned ontologies commonly share some characteristics which distinguish them from most manually built ontologies: redundancy and restricted expressivity.

Redundancy is caused because many ontology learning approaches generate

logically redundant axioms. This leads to possibly having many different explanations for a single defect. In contrast, manually created ontologies can be expected to have much less redundant axioms since an ontology engineer would most likely not state an axiom that is already apparent from the ontology multiple times. This redundancy could be reduced by applying logical inference during the learning process but this could potentially limit the learning approaches efficiency. Furthermore, removing learned axioms too early in the process could lead to the loss of information, e.g., confidence values for redundant axioms could still be interesting if the entailing axiom shows to be incorrect later-on.

Most learning approaches also generate axioms of a more *restricted expressivity* than the actual ontology language is able to represent. While human engineers are only limited by the possibilities of the underlying logics, the automatic approaches usually only support specific types of axioms while other types are excluded. Furthermore, automatically learned axioms might follow certain patterns thus not showing the full range of expressivity. For instance, ontologies learned from textual contents are more likely to refer to named classes than to complex class descriptions. When concentrating on learned ontologies where the expressivity restrictions are known beforehand, these can be exploited during the reasoning process. Due to their applicability to general ontologies, reasoning systems like Pellet [88] or Hermit [43] cannot rely on such assumptions and hence have to employ various optimization techniques to improve the efficiency of the reasoning process. For computing many or all explanations of an entailed consequence, many of these techniques have to be disabled [88] which ultimately leads to performance and stability issues.

In this chapter, we present a novel and robust approach which we developed for computing explanations on learned ontologies as originally introduced and evaluated in collaboration with Christian Meilicke, Johanna Völker and Mathias Niepert [36]. The proof of completeness is mainly the work by Christian Meilicke, nevertheless, we include some parts of it in the following for the sake of self-containedness. Though this approach is applicable to both manually engineered and automatically generated ontologies, it is specially optimized for the latter. In particular, we concentrate on ontologies which contain subsumption and disjointness axioms between classes and properties, domain and range restrictions as well as inverse properties. Our approach is based on a rule-based calculus which allows us to detect unsatisfiable classes and properties caused by the supported axiom types and to compute explanations for detected incoherences.

The remainder of this chapter is structured as follows. First, we give an overview on related work in Section 7.1. Afterwards, in Section 7.2, we describe the approach implemented in TRex.[1] In particular, we present the set of rules which is used for inferring consequences from the already known axioms. In Section 7.3, we report on the experiments which we performed and present the corresponding

[1]The implementation is publicly available at `http://dfleischhacker.github.com/trex-reasoner`.

results. Finally, we summarize the results of this chapter, discuss them and give some possibilities for future work in Section 7.4.

7.1 Related Work

The need for finding explanations for inferred axioms in ontologies and in particular for unsatisfiable classes in ontologies has been recognized early in the history of the Semantic Web idea. An early way of generating explanations for unsatisfiable classes was proposed by Schlobach and Cornet [85] who call the process of finding explanations *axiom pinpointing*. They extend the basic tableau algorithm for the $\mathcal{ALC}$ description logic so that it can be used to deduce explanations for unsatisfiabilities for the special case of so-called unfoldable TBoxes.

Kalyanpur et al. [58] proposed a different approach which extends the Pellet OWL reasoner [88]. Pellet uses tableau algorithms to perform inference on the given ontologies. By extending the tableau structure with a possibilities to trace the dependencies of given inferences, Kalyanpur et al. enable the reasoner to not only report unsatisfiable classes but to also provide a set of axioms that finally led to the unsatisfiability. According to their experiments, this so-called *glass-box* approach only leads to a negligible overhead in the reasoning process for typical ontologies. Furthermore, they proposed a *black-box* approach that uses a reasoner only for answering a limited set of questions, e.g., regarding the subsumption of classes, and apart from that uses the ontology structure to isolate the actual causes of unsatisfiabilities. Since this method only relies on posing basic queries to the actual reasoning component, it is not bound to a specific reasoner. Both techniques work efficiently on real-world ontologies, allowing the analysis of errors in such ontologies, however, these methods are limited to compute only one explanation for each entailment at a time. Thus, Kalyanpur et al. [56] later-on introduced a two-phased black-box approach for effectively discovering all explanations for an entailment which works by generating hitting sets based on the single explanations. In addition to using black-box approach for both single explanation generation and discovering all explanations, they also examined a more efficient hybrid method using the glass-box variant for finding single explanations and processing those further by means of the black-box approach. Their experiments on ontologies showed both approaches to perform well. However, as we discovered in experiments on different learned ontologies, neither the glass-box nor the black-box explanation generation was able to efficiently determine all explanations for unsatisfiabilities. This is most likely caused by the properties of learned ontologies being different from manually created ones.

Further work on black-box algorithms has particularly been done on reducing the number of queries to pose to the reasoner which in turn leads to improved performance of the determination of all explanations. Suntisrivaraporn et al. [89] showed modularization applied to the considered ontology as a way of further improving the performance of black-box methods. This was further extended by Du

et al. [33] providing more fine-grained modularization aspects. For learned ontologies sharing the special characteristics described above, these approaches do not show many performance advantages since their entities are typically more tightly connected.

Wu et al. [100] propose a method for computing all explanations in the OWL *pD** fragment. For this fragment, a full set of entailment rules is available and employed by the authors to perform inference on the ontology while tracking the roots of each entailment. To improve the performance and allow the application to very large knowledge bases and ontologies, Wu et al. distribute their algorithm over multiple machines using a MapReduce-framework. While our approach proposed in this chapter is not distributed, it is nevertheless similar in thus far that we also employ a rule-based calculus for inferring additional knowledge. Notably, this distributed approach would not be directly applicable to our learned ontologies since the OWL *pD** fragment does not cover class disjointness.

7.2 Approach

In this section, we present our approach to computing explanations for unsatisfiable classes and properties. As already mentioned, our approach is a consequence-driven one which uses rules to deduce new consequences from the axioms already contained in the knowledge base. In contrast to the consequence-based fragments defined in the OWL 2 standard, the collection of rules included in the approach presented in the following is highly influenced by the considered task of detecting incoherences in learned ontologies. The aspect of having learned ontologies lets us exactly define which axiom types have to be supported by the rules. Based on this, we only consider axiom types that are supported by our approaches as presented in Part I and also exclude any instance-based inference since the learning approaches exclusively produce schema-level axioms. Furthermore, since our task is the detection of incoherences, we only define rules for axiom types which can cause incoherence. The exact OWL 2 fragment which is supported by our approach is defined by the axiom types listed in the leftmost column of Table 7.1. Note that all classes and properties that appear in Table 7.1 are named classes and properties.

To be able to distinguish between the axioms that hold in the ontology by means of standard model-theoretic semantics and the entailments derived by applying our rules, we use the first-order predicate symbol shown in the second column of Table 7.1.

Table 7.1: Types of supported axioms.

Type of axiom	First-order predicate symbol	Description
$A \sqsubseteq B$	$csub(A, B)$	Class Subsumption
$P \sqsubseteq Q$	$psub(P, Q)$	Property Subsumption
$A \sqsubseteq \neg B$	$cdis(A, B)$	Class Disjointness
$P \sqsubseteq \neg Q$	$pdis(P, Q)$	Property Disjointness
$\exists P.\top \sqsubseteq A$	$dom(P, A)$	Domain Restriction
$\top \sqsubseteq \forall P.A$	$range(P, A)$	Range Restriction
$P^{-1} \sqsubseteq Q$	$psubinv(P, Q)$	Inverse Property Subsumption
$P^{-1} \sqsubseteq \neg Q$	$pdisinv(P, Q)$	Inverse Property Disjointness

$$\Rightarrow csub(A, A) \tag{7.1}$$
$$cdis(A, B) \Rightarrow cdis(B, A) \tag{7.2}$$
$$csub(A, B), csub(B, C) \Rightarrow csub(A, C) \tag{7.3}$$
$$csub(A, B), cdis(B, C) \Rightarrow cdis(A, C) \tag{7.4}$$
$$\Rightarrow psub(P, P) \tag{7.5}$$
$$pdis(P, Q) \Rightarrow pdis(Q, P) \tag{7.6}$$
$$psub(P, Q), psub(Q, R) \Rightarrow psub(P, R) \tag{7.7}$$
$$psub(P, Q), pdis(Q, R) \Rightarrow pdis(P, R) \tag{7.8}$$
$$dom(P, A), csub(A, B) \Rightarrow dom(P, B) \tag{7.9}$$
$$ran(P, A), csub(A, B) \Rightarrow ran(P, B) \tag{7.10}$$
$$psub(P, Q), dom(Q, A) \Rightarrow dom(P, A) \tag{7.11}$$
$$psub(P, Q), ran(Q, A) \Rightarrow ran(P, A) \tag{7.12}$$
$$cdis(A, B), dom(P, A), dom(P, B) \Rightarrow pdis(P, P) \tag{7.13}$$
$$cdis(A, B), ran(P, A), ran(P, B) \Rightarrow pdis(P, P) \tag{7.14}$$
$$psubinv(P, Q), dom(Q, A) \Rightarrow ran(P, A) \tag{7.15}$$
$$psubinv(P, Q), ran(Q, A) \Rightarrow dom(P, A) \tag{7.16}$$
$$psubinv(P, Q), psubinv(Q, R) \Rightarrow psub(P, R) \tag{7.17}$$
$$psubinv(P, Q), psub(Q, R) \Rightarrow psubinv(P, R) \tag{7.18}$$
$$psub(P, Q), psubinv(Q, R) \Rightarrow psubinv(P, R) \tag{7.19}$$
$$pdisinv(P, Q), psub(R, Q) \Rightarrow pdisinv(P, R) \tag{7.20}$$
$$psubinv(P, Q), pdis(Q, R) \Rightarrow pdisinv(P, R) \tag{7.21}$$
$$psubinv(P, Q), pdisinv(Q, R) \Rightarrow pdisinv(P, R) \tag{7.22}$$
$$pdisinv(P, Q) \Rightarrow pdisinv(Q, P) \tag{7.23}$$
$$pdisinv(P, P) \Rightarrow pdis(P, P) \tag{7.24}$$

Given an ontology $\mathcal{O}$, the actual entailment is performed based on rules (7.1) to (7.24) as follows. First, we create an initial set of formulas that are equivalent to the axioms stated in the ontology $\mathcal{O}$. Then, we iteratively extend the set of formulas by applying the rules to the set of all stated and derived formulas until no further formula can be derived. We refer to the resulting set of formulas as closure of $\mathcal{O}$ or $\mathbf{E}_\mathcal{O}$. Based on this closure, if $cdis(A, A) \in \mathbf{E}_\mathcal{O}$ or $pdis(P, P) \in \mathbf{E}_\mathcal{O}$ holds, we can conclude that class A or property P, respectively, is unsatisfiable. If there is no such class A and no such property P, $\mathcal{O}$ does not contain any unsatisfiable classes and properties and, hence, $\mathcal{O}$ is coherent.

Given this set of rules, we have to check them regarding *correctness* and *completeness*. The correctness of each single rule follows directly from the standard DL semantics. With respect to the way used to detect unsatisfiable classes, assume that we derive a formula $cdis(A, A) \in \mathbf{E}_\mathcal{O}$. This means that $\mathcal{O} \models A \sqsubseteq \neg A$ and hence also implies $A^\mathcal{I} = \emptyset$ for each interpretation $\mathcal{I}$. The equivalent conclusion also holds for properties. We conclude that our approach is sound regarding the computation of entailments, and thus also with respect to detecting unsatisfiable classes and properties.

Regarding the completeness of the rules, we resort to the following proposition whose proof is available in a technical report [37].

Proposition 1. If $\mathcal{O}$ is incoherent, there exists a class A with $cdis(A, A) \in \mathbf{E}_\mathcal{O}$ or a property P with $pdis(P, P) \in \mathbf{E}_\mathcal{O}$.

Finally, we will argue that our reasoner is able to compute all explanations for all unsatisfiable classes and properties. For this purpose, we first describe our approach of computing explanations in more detail. As already mentioned above, we apply the completion rules iteratively to derive new entailments. To reduce the checking of possible candidates for deriving new formulas, we perform the completions steps in an ordered way. In particular, we proceed as follows with $\mathbf{E}_\mathcal{O}$, $\mathbf{E}'_\mathcal{O}$ and $\mathbf{E}''_\mathcal{O}$, all three initialized as empty sets.

Class Subsumption We add all formulas $csub(A, B)$ corresponding to stated axioms to $\mathbf{E}'_\mathcal{O}$. Then, we add those formulas that are entailed by rule (7.1) to $\mathbf{E}'_\mathcal{O}$. Afterwards, we apply rule (7.3) on $\mathbf{E}'_\mathcal{O}$ until we cannot derive new formulas. This deduces all transitive class subsumption relations. Since no further $csub(A, B)$ appears in the head of any other rule, we know that $\mathbf{E}'_\mathcal{O}$ is $csub$-saturated.

Property Subsumption The first part of deducing property subsumption axioms is very similar to those done for the class subsumption axioms before. First, we add all formulas $psub(A, B)$ and $psubinv(A, B)$ corresponding to stated axioms to $\mathbf{E}''_\mathcal{O}$ and add those formulas that are entailed by rule (7.5) to $\mathbf{E}''_\mathcal{O}$. Then, we apply rules (7.7), (7.17), (7.18), and (7.19) on $\mathbf{E}''_\mathcal{O}$ until we cannot derive new formulae. These rules are handling property subsumption transitivity but also the influence of inverse property statements with respect to

property subsumption. Since there appears no $psub(P, Q)$ or $psubinv(P, Q)$ in the head of any other rule, we know that $\mathbf{E}''_{\mathcal{O}}$ is $psub$- and $psubinv$-saturated.

Domain and Range For domain and range axioms, we need both class-centric axioms and property-centric axioms. Thus, we work on axioms in $\mathbf{E}_{\mathcal{O}}$ which we set to $\mathbf{E}_{\mathcal{O}} = \mathbf{E}'_{\mathcal{O}} \cup \mathbf{E}''_{\mathcal{O}}$. We add all formulas $dom(P, A)$ and $ran(P, B)$ corresponding to stated axioms to $\mathbf{E}_{\mathcal{O}}$. Then, we apply rules (7.9), (7.10), (7.11), (7.12), (7.15), and (7.16) on $\mathbf{E}_{\mathcal{O}}$ until we cannot derive new formulas. These rules propagate the domain and range restrictions along the property hierarchy. Since no $dom(P, A)$ or $ran(P, B)$ appears in the head of any other rule, we know that $\mathbf{E}_{\mathcal{O}}$ is dom- and ran-saturated.

Class Disjointness We add all formulas $cdis(A, B)$ corresponding to stated axioms to $\mathbf{E}_{\mathcal{O}}$. Afterwards, we apply rule (7.2) and (7.4) on $\mathbf{E}_{\mathcal{O}}$ until we cannot derive new formulas. Since there appears no $cdis(A, B)$ in the head of any other rule, we know that $\mathbf{E}_{\mathcal{O}}$ is $cdis$-saturated.

Property Disjointness Again mirroring the class-based axioms, we add all formulas $pdis(P, A)$ and $pdisinv(P, B)$ corresponding to stated axioms to $\mathbf{E}_{\mathcal{O}}$. Then, we apply all remaining rules until we cannot derive new formulas. $\mathbf{E}_{\mathcal{O}}$ is now saturated with respect to all types of formulas.

7.2.1 Generation of Explanations

The process given above guarantees that we do not miss entailments. However, if we stop the entailment process as soon as it is not possible to derive any new entailment, it will not be possible to compute all explanations. This can happen because there might be alternative ways to derive a single entailment. By terminating the entailment process too early, these derivations would not have been discovered and hence the corresponding explanations would still be missing. Thus, we have to use a different criterion for moving from one step to the next and finally for terminating the whole process. The idea is to continue with the next step (or to terminate) only if there exists no $\alpha \in \mathbf{E}_{\mathcal{O}}$ such that the explanation of α has been modified during the last iteration.

Let now $expl(\alpha)$ denote the set of all explanations for a given formula α that is added to $\mathbf{E}_{\mathcal{O}}$ during executing the process described above. For the sake of simplicity, we only mention $\mathbf{E}_{\mathcal{O}}$ in the following, which might refer to $\mathbf{E}_{\mathcal{O}}$, $\mathbf{E}'_{\mathcal{O}}$ or $\mathbf{E}''_{\mathcal{O}}$ depending on the current phase of the process. We have to distinguish between two cases.

- α corresponds to a stated axiom in $\mathcal{O}$. We set $expl(\alpha) = \{\{\alpha\}\}$.
- α is derived by one of the other rules. We set

$$expl(\alpha) = expl(\alpha) \cup \{\{expl(\beta_1), \ldots, expl(\beta_n)\}\}$$

where $\beta_1, \ldots, \beta_n$ refers to those formulas that triggered the rule.

Due to the recursive character of an explanation, $expl(\alpha)$ can be understood as a disjunction of conjunctions, that might again be built from a disjunction of conjunctions, and so forth. Thus, the approach, as it has been described so far, constructs an or-and-tree of explanations. To reduce the overall complexity of the resulting tree, we want to ensure that $expl(\alpha)$ is always stored as a *disjunctive normal form* (DNF), i.e., $expl(\alpha)$ is always a disjunction of conjunctive clauses. To guarantee the explanations to be in DNF, we apply the distributivity law every time we combine explanations. Afterwards, we minimize the resulting DNF by removing conjunctions that are supersets or duplicates of other conjunctions. Checking for duplicates is important with respect to our termination criteria, because a DNF to which we try to add a duplicate or a superset should not be counted as an explanation that has been modified.

Now, we show that our approach computes all minimal incoherence preserving subsets of an incoherent ontology $\mathcal{O}$. Schlobach and Cornet [85] have defined a MIPS M (minimal incoherence preserving sub-TBox) as a subset $M \subseteq \mathcal{O}$ such that M is incoherent and each $M' \subset M$ is coherent. An explanation of an unsatisfiable class (or property) is called a MUPS (minimal unsatisfiability preserving sub-TBox) in the terminology of Schlobach and Cornet. Consequently, each MUPS is a MIPS or a superset of a MIPS. We now use $MIPS(\mathcal{O})$ to refer to the set of all MIPS in an incoherent ontology $\mathcal{O}$. Furthermore, let $expl_u(\mathcal{O})$ refer to the union of explanations for unsatisfiable classes or properties that are computed by our approach, i.e.,

$$expl_u(\mathcal{O}) = \bigcup_{\substack{cdis(A,A)\in\mathbf{E}_\mathcal{O}\wedge \\ pdis(P,P)\in\mathbf{E}_\mathcal{O}}} expl(cdis(A,A)) \cup expl(pdis(P,P))$$

We now show that $MIPS(\mathcal{O}) \subseteq expl_u(\mathcal{O})$. For that proof, we have to take into account that our approach is monotonic in the sense that $expl_u(\mathcal{O}) \supseteq expl_u(\mathcal{O}')$ if $\mathcal{O} \supseteq \mathcal{O}'$. This monotonicity directly follows from the fact that we only add explanations when applying completion rules and we stop this application when no additional explanation can be added. Thus, if $\mathcal{O}$ is a superset of $\mathcal{O}'$ we will never compute fewer explanations for $\mathcal{O}$ than for $\mathcal{O}'$. Let us now apply our method to each $M \in MIPS(\mathcal{O})$. Each M is by definition an incoherent ontology. According to Proposition 1, we will thus at least detect one unsatisfiable class or property for M. Since the computation of the unsatisfiable class (or property) is, within our approach, directly coupled to the computation of an explanation, we will always compute an explanation with respect to M, i.e., $expl_u(M) \neq \{\}$. Since M is a MIPS, there exists no incoherent subset M' of M. Thus, only the full MIPS can serve as an explanation of the incoherence and so we end up with $expl_u(M) = M$. Further, we know that $M \subseteq \mathcal{O}$ and thus we conclude, based on the monotonicity

of our approach, that $expl_u(M) \subseteq expl_u(\mathcal{O})$. We conclude that

$$MIPS(\mathcal{O}) = \bigcup_{M \in MIPS(O)} expl_u(M) \subseteq expl_u(O)$$

.

We have thus shown that our approach detects all explanations for unsatisfiable classes and properties, as long as those explanations are not subsets of other explanations. However, in the context of debugging applications, it is not important to keep track of the explanations for which we did not yet prove that our approach can detect them. When exploiting explanations in a debugging context, we would resolve all unsatisfiabilities discovered by the approach using the explanations. This would also (implicitly) resolve all undiscovered unsatisfiability explanations.

7.2.2 Implementation

We implemented this approach in a prototype that is mainly based on a matrix representation for each type of formula. We define, for example, a boolean matrix for all formulas $csub(X, Y)$, where X is associated to a row and Y is associated to a column in the matrix. We first initialize the matrix with all axioms stated in the ontology $\mathcal{O}$. Then, we apply rule (7.1) adding entries to the diagonal of this matrix. The set of entailments $\mathbf{E}_\mathcal{O}$ corresponds to the entries in our matrix representation. Each cell (X, Y) in the matrix has also assigned a set of explanations $expl(csub(X, Y))$. All other types of formulas are represented in a similar fashion. For each type, we first initialize the associated matrix based on the content of $\mathcal{O}$ and then apply the rules as described above to entail new entries and the corresponding explanations. Finally, the diagonal in the matrices for the predicates $cdis$ and $pdis$ refer to the set of unsatisfiable classes and properties as well as to their corresponding explanations.

Note again, that we have developed the approach for debugging ontologies that have been learned automatically. Such ontologies will typically contain subsumption axioms between most pairs of classes that subsume each other, even though most of these axioms can be derived from other axioms that have also been learned. The same holds for disjointness axioms. Thus, we expect that most matrices for learned ontologies are dense or not as sparse as matrix representations of carefully modeled ontologies. In such a setting using a matrix representation is less critical with respect to memory and runtime issues as it will be the case in other scenarios.

Finally, though the theoretical foundations presented above support it based on the rules rules (7.15) to (7.24), we have not yet implemented support for inverse properties in the current prototype. This is because the dataset that we used for our experiments, contains only a small number of axioms that involved inverse properties. Furthermore, inverse properties are more complex regarding runtime efficiency and would lead to a considerable growth of the employed matrices which would limit the overall performance given the current architecture of the approach. Therefore, we decided to remove all inverse properties axioms from the datasets that we used in the experiments presented in the following section.

7.3 Experiments

To show the feasibility of the overall approach and also to verify our implementation, we performed several experiments which we describe in the following.

7.3.1 Settings

In this chapter, we describe an approach which is focused on generating explanations for learned ontologies. Thus, we performed the experiments on the ontologies created by the learning process on the DBpedia ontology version 3.7 as described in Chapter 5 and 6. Thus, the ontologies only contained the axioms of the DBpedia ontology and in addition the axiom types supported by the learning approaches. For analyzing the impact of different ontology sizes on the reasoning performance, we created subsets of the full ontology. We first generated a base ontology by randomly sampling 20% of the total number of axioms from the full ontology. Then, we gradually added randomly selected and not yet contained axioms from the full ontology. During the sampling process, we regularly took snapshots of the current ontology. This resulted in a set of 11 ontologies $\mathcal{O}_0$ up to $\mathcal{O}_{10}$ so that for each ontology we know $\mathcal{O}_i \subset \mathcal{O}_{i+1}$, $i \in [0, 9]$. The last snapshot, $\mathcal{O}_{10}$ is equivalent to the full ontology $\mathcal{O}$. An analysis of the ontologies' complexity revealed that all of the generated ontologies belong to the $\mathcal{ALCH}$ complexity class. Further statistics regarding the ontologies can be found in Table 7.2.

Table 7.2: Statistics about ontologies used in experiments.

Ontology	Axioms	Classes	Properties	Unsat. Classes	Unsat. Properties
$\mathcal{O}_0$	23,706	300	654	3	5
$\mathcal{O}_1$	32,814	304	673	6	7
$\mathcal{O}_2$	41,941	309	689	9	14
$\mathcal{O}_3$	51,056	316	702	15	29
$\mathcal{O}_4$	60,166	319	714	26	50
$\mathcal{O}_5$	69,271	321	724	32	82
$\mathcal{O}_6$	78,375	323	730	49	112
$\mathcal{O}_7$	87,468	324	736	63	162
$\mathcal{O}_8$	96,555	324	737	83	209
$\mathcal{O}_9$	105,642	324	742	132	336
$\mathcal{O}_{10}$	114,726	324	742	152	396

Towards the overall objective of fixing incoherences in ontologies, there arise two basic use cases that we consider in the following. The first use case is the detection of unsatisfiable classes and properties. Naturally, this is one of the first steps in fixing the incoherences as it points out the actual problems in the ontology to further look at. The second use case is about finding all explanations for unsatisfiabilities. Based on the results of the first use case, this explanation generation

step can be limited on the incoherent classes and properties instead of considering the full ontology.

We compared TRex to the two state-of-the-art reasoners Pellet and Hermit. While TRex fully supports both use cases, Hermit and Pellet are only able to handle subsets. Regarding the first use case, Hermit[2] provides direct programmatic access to the set of unsatisfiable classes whilst retrieving unsatisfiable properties is not directly possible. Instead, we resorted to retrieving all subproperties of `owl:bottomObjectProperty`. Hermit does not provide support for generating explanations, so it is not suited for our second use case. The Pellet reasoner[3] also supports direct retrieval of unsatisfiable classes. In contrast to Hermit, it neither supports direct access to unsatisfiable properties nor does it allow the retrieval of subproperties of `owl:bottomObjectProperty`. Thus, we are not able to directly retrieve unsatisfiable properties with Pellet without further modifications.

A feature which distinguishes Pellet from Hermit is the support for computing explanations. To have the possibility to compare our explanation results with other explanations, we implemented a way of reducing the detection of unsatisfiable properties to the detection of unsatisfiable classes. For this purpose, we extended the ontologies with the axiom $C_P \sqsubseteq \exists P.\top$ for each object property P in the ontology where C_P is a fresh class introduced for the respective property. Based on this, we know that C_P is unsatisfiable iff P is unsatisfiable. In our experiments, this variant of Pellet is referred to as *PelletMod*.

7.3.2 Results

The results of the first use case are depicted in Table 7.3.[4] All reasoners discovered the same number of unsatisfiabilities except for Pellet because of its inability to detect unsatisfiable properties. Overall, Hermit is the fastest reasoner for retrieving the set of all unsatisfiable classes and properties. In particular, Hermit provides the best scalability in our experiments since the runtime behavior is second to none of the other reasoners. The runtimes of Pellet and PelletMod increase much more with respect to the ontology size. The runtimes of TRex are the highest for all ontology sizes. TRex is designed to always determine explanations for all inferable axioms. Pellet only computes explanations if those are explicitly requested for specific axioms. Furthermore, TRex has higher initialization costs. However, as also discoverable from the results, these initialization costs are hardly affected by the growing number of axioms. Both characteristics of TRex can be explained by the usage of matrices in the TRex implementation. The dimension of these matrices are determined by the number of classes and properties in the ontology and their initialization has to be done independently from the number of axioms. Furthermore, since the number of classes and properties does only change to a low

[2] `http://www.hermit-reasoner.com`, Version 1.3.6

[3] `http://clarkparsia.com/pellet`, Version 2.3.0

[4] All experiments have been conducted on a Quad-core Intel Core i7 with 3.07GHz and 24GB RAM. The results are averaged over 5 runs.

degree according to Table 7.2. Thus, the matrix sizes and their initialization costs are almost the same over all considered ontologies. The actual number of axioms has a lower influence on the runtime behavior.

Table 7.3: Runtimes in milliseconds for the detection of unsatisfiabilities.

Ontology	Pellet	PelletMod	Hermit	TRex
$\mathcal{O}_0$	392	411	450	6,630
$\mathcal{O}_1$	621	654	629	7,169
$\mathcal{O}_2$	910	997	720	7,839
$\mathcal{O}_3$	1,232	1,297	849	8,425
$\mathcal{O}_4$	1,485	1,854	1,916	9,889
$\mathcal{O}_5$	1,970	2,088	1,158	9,411
$\mathcal{O}_6$	2,419	2,617	1,295	9,572
$\mathcal{O}_7$	2,897	3,063	1,468	12,559
$\mathcal{O}_8$	3,460	3,585	1,549	10,124
$\mathcal{O}_9$	3,823	3,899	1,721	11,148
$\mathcal{O}_{10}$	4,327	4,439	1,864	12,006

For the second use case, we provide the results in Table 7.4. These results are the runtimes for retrieving the explanations for each of the unsatisfiable classes and properties found by the respective reasoner. Thus, the runtimes of Pellet only include the explanation retrieval for unsatisfiable classes. For comparing the runtimes of Pellet and PelletMod to those of TRex, it is important that TRex runtimes are measured for retrieving *all explanations* for each unsatisfiability in the ontology while for Pellet and PelletMod only *a single explanation* per unsatisfiability has been retrieved. The number of explanations as found by TRex is provided in the right-most column. In contrast, Pellet generates one explanation for each unsatisfiable class while PelletMod generates one for each unsatisfiable class or property. As already said in the motivation, the explanation generation facilities of Pellet, and hence also Pellet, are highly unstable. While the retrieval of a single explanation worked well, trying to retrieve multiple explanations for a single unsatisfiability consistently led to errors for the ontologies in our experiments. When trying to work around these problems, we observed extremely high runtimes severely limiting the practical applicability of the systems. However, it remained unclear whether those high runtimes were caused by the correct execution of the algorithm until the problem occurred or by the problems themselves. For that reason, we omitted to present these results, which would in any case be based on incomplete sets of explanations, and resorted to single-explanation retrieval for Pellet and PelletMod.

As we see from the given table, the runtime of TRex for retrieving all explanations for all unsatisfiabilities is increasing exponentially with the number of axioms contained in the ontology while the other reasoners only suffer from an approximately linear increase. However, this increase can be explained by means

Table 7.4: Runtimes in milliseconds for generating explanations for unsatisfiable classes and properties. "All Explanations" means all MIPS.

	Single Explanation				All Explanations	
	Pellet		PelletMod		TRex	
Ontology	Runtime	# Expl.	Runtime	# Expl.	Runtime	# Expl.
$\mathcal{O}_0$	848	3	863	8	6,758	8
$\mathcal{O}_1$	1,317	6	1,365	13	7,594	13
$\mathcal{O}_2$	1,899	9	1,956	23	9,011	26
$\mathcal{O}_3$	2,463	15	2,693	44	9,892	54
$\mathcal{O}_4$	3,341	26	3,530	76	11,666	100
$\mathcal{O}_5$	4,070	32	4,322	114	11,732	158
$\mathcal{O}_6$	5,068	49	5,235	161	12,980	250
$\mathcal{O}_7$	5,979	63	6,309	225	17,495	386
$\mathcal{O}_8$	7,082	83	7,396	292	21,726	686
$\mathcal{O}_9$	7,805	132	8,228	468	44,966	2,031
$\mathcal{O}_{10}$	8,947	152	9,480	548	66,781	2,722

of the aforementioned difference in the number of explanations. Pellet and PelletMod only retrieve one explanation per unsatisfiability, which means that the total number of explanations is linear to the number of unsatisfiable classes and properties. In contrast, the total number of explanations retrieved by TRex is not linearly bound but instead growing exponentially in the number of axioms. This is also depicted in Figure 7.1. In this figure, we plotted both the runtimes of TRex and the number of computed explanations on the y-axis. Again, we observe a relatively high initialization cost. However, at some point in time the runtimes of TRex seem to grow linear in the number of computed explanations.

Explanation Completeness

Finally, we set up another experiment to obtain additional evidence for the completeness of the explanation component and the correctness of our implementation. Furthermore, this experiment serves as a first show case for our objective to clean incoherent ontologies using the explanations retrieved from the explanation component. The experiment implements Algorithm 5 which uses the reasoning service to get all explanations for the incoherence of an ontology $\mathcal{O}$. For each explanation, it removes one randomly chosen axiom from the ontology and thus invalidates this explanation. It continues resolving the other explanations by removing further axioms but only if the current explanation has not been resolved by the axiom removal for a previously processed explanation. All in all, the set of removed axioms H forms a randomly determined hitting set over all explanations in $expl_u(\mathcal{O})$ computed by TRex. Thus, assuming the correctness of the explanation generation, the removal of all axioms in H from the ontology should always result in a co-

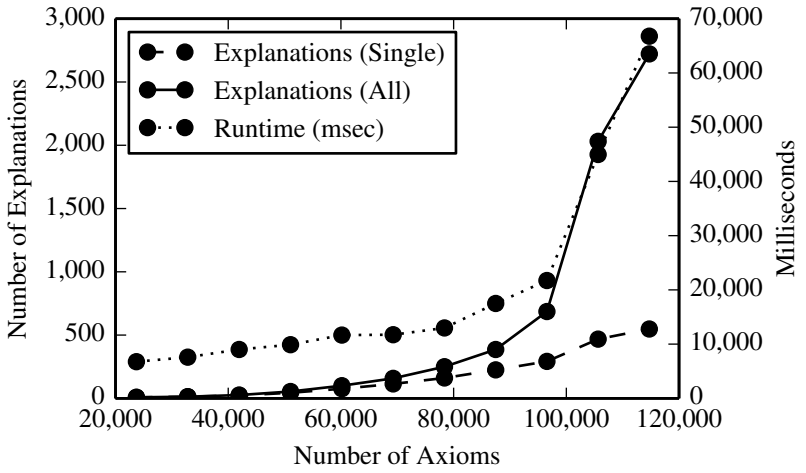


Figure 7.1: TRex explanation runtimes and the total number of retrieved explanations

herent ontology. We ran the implementation of this algorithm 200 times on the ontology $\mathcal{O}_{10}$. The computed hitting sets contained between 201 and 223 axioms. To validate the results, we tested the resulting ontology for coherence using TRex and Hermit. For each run, both reasoners did not find further unsatisfiable classes or properties. Thus, in the tested cases, the explanation generation and reasoning component implemented in TRex showed further evidence for its correctness.

Algorithm 5 Randomized greedy ontology debugging

function RANDOMIZEDGREEDYDEBUG($O, expl_u(\mathcal{O})$)
  $H \leftarrow \{\}$ ▷ set for storing already removed axioms
  for all $e \in expl_u(\mathcal{O})$ **do** ▷ e is an explanation, i.e., a set of axioms
    if $e \cap H = \varnothing$ **then**
      $a \leftarrow$ randomly chosen axiom from e
      $O \leftarrow O \setminus \{a\}$
      $H \leftarrow H \cup \{a\}$
    end if
  end for
end function

7.4 Conclusion

In this chapter, we argued towards the usage of specialized reasoning facilities for cases in which the full expressive power of ontology languages like OWL 2 is not

required. Though OWL 2 defines profiles which provide a subset of the full expressivity while being more efficient to compute than full OWL 2, these subsets still try to provide as many language constructs as possible. In particular, these profiles are required to support general applications and thus cannot restrict themselves as comprehensively as a specially fitted approach. For example, none of the OWL 2 profiles leaves out the support for instance-level inference which is not required if the target datasets are known not to contain any instance informations. Though in general, supporting unused constructs does not lead to considerable performance limitation due to the highly optimized nature of reasoning tools, they could lead into problems in cases where some or all optimizations have to be disabled. When explanations for inferred axioms should be generated, it is not possible to work with the full set of optimizations since these would hinder their proper computation. This leads to a highly increased runtime which even gets more drastic when not only single explanations but all explanations for a given inferred axiom have to be determined. This might also be one reason for most systems not supporting the generation of explanations at all and furthermore for the problems the supporting systems suffer from.

For the special use case of debugging and repairing learned ontologies, we defined a set of entailment rules tailored after the capabilities of our inductive ontology learning approaches. These rules only support schema-level inference for a subset of OWL 2 and also are limited to axiom types which can have influence on the coherence of ontologies. Furthermore, we introduced an approach for determining the actual entailments based on these rules and, as an even more important aspect, the generation of explanations for unsatisfiable classes and properties discovered by means of these rules. We also theoretically showed that we are able to determine all relevant explanations for ontologies falling into the supported expressivity range. Based on our prototypical implementation TRex, we showed the practical feasibility of the idea and performed experiments providing further results also with respect to the actual debugging of incoherent ontologies. Due to the way of implementation chosen for TRex, we experienced higher runtimes for detecting incoherences and generating single explanations in the ontology than state-of-the-art systems. However, we also showed our system to be able to provide all explanations for unsatisfiabilities while neither Pellet nor Hermit were able to do this. The matrix representation for axioms used in our prototypical implementation might also lead to performance problems when working on ontologies containing large numbers of classes and properties. Nevertheless, this implementation detail was chosen based on the specific characteristics of our use case in which the learned ontologies can only contain a limited number of classes and properties since no new entities are generated but merely their relations towards each other are determined in more detail. Another advantage of specifically crafted reasoning systems is the increased influence on their internals which allows to more directly retrieve information relevant for several use cases. For instance, the current implementation also provides access to cycles in the inferred subsumption hierarchy. *Subsumption cycles* are explanations for $\mathcal{O} \models A \equiv B$ where $A \equiv B \notin \mathcal{O}$. Such an equivalence

is not necessarily an undesired consequence, however, it might be worthwhile to analyze the involved axioms. Now suppose that we want to compute subsumption cycles for a highly incoherent ontology where most classes, including two classes A and B, are incoherent. Due to that fact that A and B are subsumed by bottom, we have $\mathcal{O} \models A \equiv B$. The explanations for the unsatisfiability of A and B are thus also responsible for the equivalence of A and B. This is not the case in our approach, because $\bot$ is not included in our formalization. Subsumption cycles determined by means of our approach are thus not affected by unsatisfiabilities. This might help to spot errors in the learned axioms. An example for this are ontology learning systems which are able to learn subsumption and equivalence axioms separately. In cases where the learned subsumption axioms imply an equivalence but no equivalence axiom is learned, subsumption cycles can help to more closely examine the cycles which might not be possible in highly incoherent ontologies when using traditional reasoning systems.

Possible areas for improving the implementation particularly consist of changing the internal representation structure used in the reasoning system. The current implementation's matrix structure poses a limit onto the number of classes and properties which can be processed in the system. Changing this implementation detail could lead into improvements in scalability based on the number of entities in the ontology. Another possibility to optimize the current approach is the representation of explanations. First experiments have been done to represent the explanations as ordered binary decision diagrams, however, these did not yet turn out to be successful.

All in all, the implementation of specialized systems seems to prove beneficial if the restrictions on the expressivity compared to the standard OWL 2 profiles are considerably high and thus allow to waive a greater part of the inference rules. An additional benefit which might prove important for many applications is the extended influence on internals of the approach compared to highly optimized, general purpose systems.

Chapter 8

Repairing Incoherent Ontologies

Providing explanations of incoherences to an ontology engineer is a first step in supporting him in improving the ontology. Nevertheless, for large ontologies having a high degree of incoherence, many explanations can be found that have to be fixed and it might be a high effort to get an overview on all these explanations to finally determine the most suitable solution. Thus, an even larger degree of automatized support is desirable.

For the special case of learned ontologies, we can take advantage of another property for achieving this higher level of support. In many cases, the underlying approaches employed in ontology learning do not directly generate binary decision whether a given axiom holds but first collect evidences for the validity of the axiom. These evidence collection results in a confidence value which is then used to decide whether the axioms holds or not. Many learning approaches make the confidence values, which they gathered internally, available in the final ontology by means of annotations assigned to the corresponding axioms. Thus, when debugging ontologies which contain confidence-annotated axioms, we not only have the explanations for the incoherences but for each learned axiom, we also have an assessment of the confidence in its validity. An obvious use of this additional information is to decide which axiom from an explanation should be removed from the ontology based on this confidence measure.

In this chapter, we examine means of automatically repairing incoherent ontologies. All in all, we are arguing that a fully automatic repair of incoherent ontologies is hardly possible as is the automatic learning of flawless ontologies. However, by further looking into automatic repair approaches, we can get additional insights which later-on could help to find a balance between low manual effort and high quality. Moreover, in some use cases it might be hardly or not at all possible to provide manual guidance to automatic approaches, for example, because the effort to do so would be to high compared to the expected results. Thus, it is also important to examine these fully automated approaches.

The remainder of this chapter is structured as follows. In the next section, we elaborate on related work. We mainly cover work about debugging and repairing

ontologies generated by ontology learning approaches. Afterwards, in Section 8.2, we present the four different approaches which we evaluate for making incoherent ontologies coherent. This includes two approaches which apply greedy strategies on the sets of explanations generated by reasoning tools. The two additional approaches employ the concept of Markov logic networks to find coherent axiom subsets of the original ontology. We use the ontology reasoning tool TRex as presented in the previous chapter for generating explanations and apply the RockIt tool by Nössner et al. [74] when working with Markov logic networks. The different approaches are evaluated in Section 8.3 before we summarize and conclude in Section 8.4.

8.1 Related Work

Given the central importance of ontologies for the Semantic Web in combination with the potential complexity of modeling them, much work has been done on assuring their quality. During the ontology creation, modeling guidelines and frameworks provide hints on how to depict specific real-world scenarios in an ontology and further means to validate a given way of modeling. One instance of such an approach is the OntoClean approach [45] that introduces a number of meta-properties borrowed from the field of philosophy. These meta-properties are assigned based on the characteristics of entities that are relevant for placing them into specific classes according to the specific way of conceptualization chosen for the ontology. Based on these meta-properties, certain restrictions are defined regarding when classes can be in a taxonomic relation. One example is the meta-property of *rigidity*. A class being rigid means that it is essential to all its instances, i.e., the instances cannot stop being instances of this class in certain situations. An anti-rigid class, however, can get more classes or lose classes depending on the specific situation. Guarino and Welty provide the class `Human` as an example for a rigid class while the class `Student` is considered to be anti-rigid. According to the OntoClean constraints, anti-rigid classes can only subsume anti-rigid classes. For example, if the anti-rigid class `Student` would subsume the rigid class `Human`, an instance ceasing to be a student would violate the rigidity of `Human`. By defining additional meta-properties and constraints based on them, OntoClean provides a way of formalizing ontological assumptions and for recognizing violations of these assumptions introduced by the taxonomy. Adhering to such strict rules during modeling helps to create ontologies that are of better overall quality. Obviously, applying such strategies requires a deep understanding of the modeled domain and, hence, is hardly applicable using automatic ontology generation approaches.

Other works take less formalized approaches but concentrate on suspicious patterns or “bad smells” occurring in the ontology. Corcho et al. [30] introduce a set of anti-pattern to recognize errors in ontologies they found to be common in manually created ontologies. Furthermore, they provide a set of guidelines on how to model complex facts in simpler constructs. The anti-patterns are categorized

into detectable logical anti-patterns which can be discovered by means of reasoning and cognitive logical patterns that are possible modeling errors but not detectable by a reasoner. For each anti-pattern, the authors also provide an alternative way of modeling that expresses the assumed intended meaning of the discovered anti-pattern. Based on these patterns, they propose an ontology debugging strategy that detects occurrences of the anti-patterns or logical constructs for which modeling guidelines exist and provides the defined alternative way of modeling. Roussey and Zamazal [82] examine the usage of SPARQL queries for detecting occurrences of anti-patterns in ontologies. They also address some basic transformations to overcome differences in modeling that would otherwise hinder the detection of the patterns and in addition provide a limited capability to perform basic inference.

The web-based OntOlogy Pitfall Scanner! (OOPS!) by Poveda-Villalón et al. [78] targets in the same direction by providing automatic detection of common problems by means of patterns and best practices. The patterns employed in OOPS! are sorted into different categories such as human understanding or logical consistency. These pattern-based approaches are concentrating on typical errors introduced by humans into manually modeled ontologies and some logical weaknesses recognizable either directly by patterns or with little support of a reasoning tool. Due to this concentration on errors in typical human-made ontologies and a subset of the logically recognizable problems, these approaches are not well-suited for processing automatically generated ontologies. However, these methods could be helpful when applied in a second step and used by a human ontology engineer for improving learned ontologies.

Another type of approaches for debugging ontologies focuses on the quality of ontologies on a logical level. For this purpose, consistency of the ontology is the more obvious aspect since it is a requirement for most inference approaches being applied to the ontology and inconsistencies point to modeling flaws either for a specific instance or in the overall ontology's logical model. Though not having direct influence on the usability of the ontology for many use cases, coherence is nonetheless considered a pointer to modeling problems. This is backed by the assumption that unsatisfiable classes (or properties) are almost never introduced on purpose into an ontology since they do not encode additional knowledge about the modeled domain. By using inference techniques, they detect unsatisfiabilities in the ontology and repair those in most cases by removing one of the axioms causing the unsatisfiability. These axioms are identified by means of explanations (see Chapter 7). Using the explanation-enabled Pellet reasoner [58], Kalyanpur et al. generate explanations and provide them in their Swoop tool [57] to the ontology engineer supporting the search for an appropriate fix. The RaDON tool by Ji et al. [55] provides similar functionality to support human engineers. In addition, it provides automatic repair for incoherent and inconsistent ontologies based on the determined explanations by removing the axioms causing the unsatisfiability. For handling incoherence, RaDON uses the kernel revision operator introduced by Qi et al. [80] which concentrates on revising ontologies that are changing over time. If a newly received axiom leads to incoherence, this is handled by removing an

axiom from the original ontology. The axioms responsible for the unsatisfiability are determined by the Pellet reasoner and then further processed using a hitting set tree algorithm as also employed by Kalyanpur et al. [56]. Qi et al. examine three different incision functions for finding the axiom to remove. For this purpose, different criteria are used. One criterion is based on the number of occurrences of an axiom in different MIPSs and only considers the most often occurring axioms for removal. Two additional approaches also use confidence values that might be available for automatically generated axioms. In a first step, the lowest confidence axioms for each MIPS are determined and afterwards a hitting set of axioms to remove is calculated. The first approach we are going to introduce later-on, is comparable to this one, however we do not use the hitting set tree algorithm by Reiter but rather rely on a greedy procedure which in theory produces less optimal results but is more efficient. It is worth noting, that the algorithm by Qi et al. does not produce minimum-cardinality hitting sets, either. Qi et al. also introduce a more efficient method which only retrieves all MUPSs for each unsatisfiable class instead of the full set of MIPSs at once. All confidence-based methods proposed by Qi et al. only minimize over the lowest confidence axioms for each MIPS. Thus, these methods would never consider that the removal of several axioms of low confidence could be weighed up by the removal of a single slightly higher confidence axiom. This is possible in the Markov logic-based methods that we describe below since they are able to consider the number of axiom occurrences in explanations and the confidence values of axioms at the same time.

8.2 Approaches

In this section, we present the different approaches that we examine in this chapter for repairing incoherent ontologies. These approaches are all similar in that they are heavily based on using the confidence measures available in the learned ontologies when deciding which axiom to remove. We do this under the premise that higher confidence values hint on a higher probability of an axiom being correct. For example, given the inductive approaches presented in the first part of this work, this means that we tend to remove axioms to which fewer instances adhere compared to one to which more instances adhere.

As foreshadowed in the introduction of this chapter, the approaches can be roughly split into two categories. The first type of approaches is crafted after the basic structure of greedy algorithms. One of these greedy algorithms is a confidence-based variant of Algorithm 5 that is commonly used for debugging ontologies and thus serves as a baseline. The second greedy approach iteratively constructs the full ontology instead of removing axioms from the full ontology. The algorithms falling into the second category are using Markov logic networks (MLN). Here, the first approach operates on pre-computed explanations for the unsatisfiabilities. It tries to find a lowest-confidence set of axioms whose removal makes the ontology coherent. The fourth approach is purely based on MLN and does not use explana-

tions generated by a reasoning system. Instead it implements the inference rules as given in Chapter 7 directly in Markov logic.

For the first three approaches, we assume the set of all explanations of incoherences to be given. In this work, we perform the explanation generation step using the TRex system as described above.

8.2.1 Baseline Approach

The baseline approach is specified in Algorithm 6 and is similar to Algorithm 5. Its basic idea bears on the characteristic of an explanation that it is the minimum set of axioms which leads to the unsatisfiability. Thus, when removing an arbitrary axiom contained in the explanation from the ontology, this specific unsatisfiability does not exist any longer. While the previous algorithm used in Chapter 7 just removed a randomly selected axiom, we now include the confidence value into the decision and always remove the axiom with the lowest confidence value. Since two different explanations are not necessarily disjoint, an explanation can be already solved when one of its axioms was removed during solving a previous explanation. In these cases, no further actions are required. Thus, we keep track of all axioms removed from the ontology and check for each explanation whether it has been solved by an earlier removal. Obviously, this approach is neither optimal regarding the total number of axiom removed from the ontology nor regarding the total confidence value of all removed axioms. For instance, a single high confidence axiom which participates in all explanations would never be removed as long as all explanations contain at least one axiom with a lower confidence value. In the following, we refer to this approach as A1.

Algorithm 6 Greedy ontology debugging

Require: $\mathcal{O}$ is a learned ontology, $expl_u(\mathcal{O})$ is the set of explanations for all incoherences

function GREEDYDEBUG($O, expl_u(\mathcal{O})$)
  $H \leftarrow \{\}$ ▷ set for storing already removed axioms
  for all $e \in expl_u(\mathcal{O})$ **do** ▷ e is an explanation, i.e., a set of axioms
    if $e \cap H = \varnothing$ **then**
      $a \leftarrow$axiom with lowest confidence value in e
      $O \leftarrow O \setminus \{a\}$
      $H \leftarrow H \cup \{a\}$
    end if
  end for
end function

8.2.2 Axiom Adding Approach

The second algorithm as described in Algorithm 7 differs from the previous one in that it does not start with the full ontology but with the base ontology and enriches it axiom by axiom. For this purpose, it iterates over all learned axioms and adds them one by one starting with the highest confidence axiom and continuing with lower confidence ones. After each axiom addition, the algorithm checks whether the resulting ontology $\mathcal{O}'$ fully contains the axioms of one explanation, i.e.,

$$\exists e \in expl_u(\mathcal{O}) : \mathcal{O}' \cap e = e$$

If such an e exists, the current ontology is incoherent. Since the ontology was coherent before, otherwise it would have contained one full explanation in the step before, the incoherence is caused by the lastly added axiom. Thus, we remove this axiom and the process continues with the axiom having the next lower confidence score. By guaranteeing that no explanation is fully contained in the ontology, we prevent the occurrence of all detected incoherences. In the further course of this chapter, we refer to this approach as A2.

Algorithm 7 Axiom Adding

Require: $\mathcal{O}$ is a learned ontology, $expl_u(\mathcal{O})$ is the set of explanations for all incoherences

function HITTINGSETDEBUG($\mathcal{O}, expl_u(\mathcal{O})$)
  $H \leftarrow \{\}$ ▷ set of removed axioms
  $L \leftarrow$ learned axioms contained in $\mathcal{O}$ sorted by descending confidence
  $\mathcal{O}' \leftarrow \mathcal{O} \setminus L$ ▷ $\mathcal{O}'$ is ontology without learned axioms
  for all $a \in L$ **do**
    $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{a\}$
    if $\exists e \in expl_u(\mathcal{O}) : e \subseteq \mathcal{O}'$ **then**
      $\mathcal{O}' \leftarrow \mathcal{O}' \setminus \{a\}$
      $H \leftarrow H \cup \{a\}$
    end if
  end for
end function

After the termination of this algorithm, H contains a hitting set for the set of explanations, i.e., H is a set of axioms so that $\forall e \in expl_u(\mathcal{O}) : e \cap H \neq \varnothing$. It is important to note that due to the greedy nature of the algorithm, H is a minimal but not a minimum-cardinality hitting set. The same statements holds for the algorithm proposed first where the set H also holds a hitting set after the execution of the algorithm completed.

8.2.3 MAP Inference-Based Approach

In contrast to the two approaches presented before, this approach is not a greedy one. Instead, it formulates the problem as a Markov logic network (MLN) whose solution leads to a coherent ontology.

Markov logic networks were introduced by Richardson and Domingos [81]. They are a way of formulating uncertain logical knowledge based on Markov networks. For this purpose, Markov logic extends first order logic by allowing to annotate formulas with weights. In contrast to pure description logic where all formulas represent hard constraints and a world not satisfying all constraints is no valid world, in Markov logic a world violating a constraint is not an impossible world but merely less probable. The higher the weight associated to a formula the less probable a world violating it. Because of this property, it is even possible to have formulas in the knowledge base that contradict each other. Furthermore, by adding infinite weights to formulas it is possible to set these formulas as hard constraints that might not be violated.

More formally, a Markov logic network (MLN) is given by a set of pairs (F_i, w_i) where each F_i is a first-order logic formula and each w_i a real number. Together with a set of constants C the logic network can be used to determine a ground Markov network. On this Markov network it is then possible to define the probability distribution over possible worlds x by

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right)$$

with F being the number of formulas in the MLN and n_i the number of true groundings of F_i in x.

However, for our use case, we are more interested in the scenario to find the most likely state of a world y given evidences e, i.e.,

$$\arg\max_y P(y|e)$$

which is also called maximum a-posteriori (MAP) inference. We use the Markov logic query engine RockIt by Noessner et al. [74] for finding the maximum state of the MLN. Thus, the MLN formulations given in the following are oriented towards the syntax of RockIt. However, the syntax of other MLN inference engines is similar and it should be possible to transfer the following models into their syntax without much effort. The input for performing a MAP query is two-fold. First, we need a so-called model which defines the schema which can later-on be used for describing the actual data and also provides logical relationship between the schema parts. The second part is the definition of ground values which contains the actual data on which the query is performed and is expressed based on the defined model.

In Figure 8.1, we show the basic model. This model first defines a predicate to mark an axiom as active in Line 1. We consider an axiom as active if it is included

in the final (coherent) ontology. To include the confidence values of the learned axioms into the optimization process, we define the `activeConf` predicate in Line 2 which assigns the corresponding confidence to an axiom. Line 3 and 4 are used to represent single explanations. We cannot express predicates of arbitrary length in a model but only predicates having a fixed length. Since the set of all explanations can contain explanations made of different numbers of axioms, we have to provide predicates for each explanation size. This is done by defining N-ary predicates named `conflictN` for all N up to the size of the largest explanation. In Figure 8.1, we assume a maximum explanation size of 2 and thus only support explanations of length 1 by the predicate `conflict1` and length 2 using the predicate `conflict2`. Line 5 defines the optimization to be done over the confidence values of all active axioms by only allowing to include a confidence value into the final value if the corresponding axiom is set to active. Finally, to actually ensure that unsatisfiabilities are resolved, Lines 6 and 7 state that for each explanation one of its axioms must not be set to active. As for the explanation predicates, this has to be done in one statement for each possible length of explanations. This approach is identified by A3 in the remainder of this chapter.

```
1  active(axiom)
2  *activeConf(axiom, _float)
3  *conflict1(axiom)
4  *conflict2(axiom, axiom)
5  conf: active(x) v !activeConf(x, conf)
6  !conflict1(x0) v !active(x0).
7  !conflict2(x0,x1) v !active(x0) v !active(x1).
```

Figure 8.1: RockIt model for the MAP inference-based approach

Using this model as base, we generate the ground values for the MAP inference step. This is done based on the ontology and the set of explanations. In the following steps, only those axioms are relevant which occur in at least one explanation. To identify these axioms, we assign a unique integer value to each. All others do not contribute into any unsatifiabilities on the ontology and we do not need to remove these in any case. For each unlearned axiom, i.e., each axiom that was in the ontology before the enrichment by means of ontology learning, an active statement is introduced. Directly introducing the active statements makes this axiom a hard constraint so that it cannot be treated as inactive during the MAP inference process. We introduce one `activeConf` predicate for each learned axiom and its confidence value. Each explanation gets represented by the `conflict` predicate corresponding to its size containing the identifiers of all contained axioms. Using this input data, RockIt determines a list of `active` predicates for all active axioms in the most probable world. We include all these active axioms into the base ontology and, as we already described, get a coherent ontology.

8.2.4 Pure Markov Logic Approach

In the fourth approach, which we refer to as A4, we directly combine the entailment rules as defined in Section 7.2 and Markov logic networks for repairing incoherent ontologies. In particular, this allows us to determine the performance of an approach which does not compute explanations before repairing the ontology but directly works on the entailment rules to compute the maximum confidence, coherence ontology. Our approach is highly inspired by the work of Noessner and Niepert [73] but instead of using inference rules for the logic $\mathcal{EL}$, we implement the rules for debugging learned schema-only ontologies in Markov logic. Since all rules are represented by means of implication, this is easily possible.

The resulting model is provided in Appendix A. In contrast to the third approach, we cannot only work on the axioms without further knowledge of their content. Rather, we express the connections between the entities of the ontology, i.e., the classes and the properties contained in the ontology. This is done by predicates for the different supported axiom types, each connecting a pair of classes, a pair of properties or a property and a class. For instance, we define the binary predicate `csub` for expressing a class subsumption relations between two classes and a `ran` predicate for assigning a class as range of a property. To include confidence values for learned axioms, we define ternary predicates which add the confidence values in addition to the parameters of their binary counterparts. Both, binary and ternary predicated, are connected by statements like

```
conf: csub(c1, c2) v !csubConf(c1, c2, conf)
```

to define that a given confidence values may only be considered if the corresponding axiom actually holds.

The further parts of the model work on these axioms and define the entailments based on the existence of specific axioms. They are direct translations of the previously given implication rules into their conjunctive normal form. The formulas `!cdis(c1,c1)` and `!pdis(p1,p1)` are of special importance for the full approach since they disallow classes and properties to be disjoint to themselves and, hence, ensure that coherent sets of axioms are generated.

For the ground values, this model is used as vocabulary and the axioms contained in the ontology are expressed by the corresponding predicates, for unlearned axioms by the binary predicates and for learned axioms by the ternary predicates also including their confidence values.

Querying for the MAP state based on this model and the ground values then again leads into a list of axioms which are active in a highest-confidence coherent ontology.

8.3 Evaluation

For evaluating the four approaches introduced above, we experimentally compared the four approaches regarding their runtime and also the complexity of ontologies

to which the approaches are applicable. Both aspects are relevant since the size and the complexity of automatically generated ontologies can pose challenges regarding both dimensions. Considering that repairing approaches might have to assist human ontology engineers in fixing problems in learned ontologies, another important facet is the transparency of the performed repair steps. The more transparent and thus traceable the reasons for the removal of certain axioms are, the easier a repair approach can be integrated into an interactive scenario. For some experiment runs, we also evaluated the sets of removed axioms regarding their correctness with respect to human judgment. By doing this, we were able to get an insight into the quality differences of the ontologies resulting from a manual repair to those done by automatic means.

We first describe the setup under which we performed the experiments. Afterwards, we give details on their results.

8.3.1 Settings

For the evaluation, we worked on different ontologies all generated by means of ontology learning approaches. The first ontology, which we refer to as $\mathcal{A}$, is the ontology generated using the basic association rule mining-based approach described in Chapter 5. Thus, this ontology is the DBpedia ontology version 3.7 enriched by class disjointness axioms based on the corresponding DBpedia instance data. We reuse this dataset since we have the high-quality gold standard available so that we can use it for finding the actually wrong axioms and compare them to those deleted by the repair approach. As a second dataset, we employ the one already used for evaluation in Chapter 7. This dataset has a higher expressivity than the first dataset since it not only contains learned class disjointness but also additional axiom types as property disjointness or domain and range restrictions. To assess the influence of an ontology's size on the performance of the approaches, we keep the division into eleven ontologies containing different numbers of axioms. In the following, we call these 11 ontologies $\mathcal{B}_0$ to $\mathcal{B}_{10}$. Finally, we performed the experiments on an ontology fully generated from a text corpus by the Text2Onto [28] tool. This dataset was already used by Qi et al. [80] for their experiments. It is interesting for our experiments since, in contrast to the enriched DBpedia ontologies, it is a fully learned ontology which might differ considerably regarding its basic characteristics. This ontology is called $\mathcal{C}$ in the following. Since we are only interested in coherence, we ignore the instances contained in $\mathcal{C}$. Table 8.1 summarizes the most important characteristics of all ontologies.

On these datasets, we run the different approaches described in Section 8.2 and compared them regarding their runtime and the number of axioms removed from the ontology. Based on our class disjointness gold standard for the first dataset, we computed the correctness of the axiom removals. For this purpose, we define correctness as also used by Qi et al. as

$$\frac{\text{\# correctly removed axioms}}{\text{\# removed axioms}} \tag{8.1}$$

Table 8.1: Statistics about ontologies used in experiments.

Ontology	Axioms	Classes	Properties	Unsat. Classes	Unsat. Properties
$\mathcal{A}$	48,186	394	855	8	8
$\mathcal{B}_0$	23,706	300	654	3	5
$\mathcal{B}_1$	32,814	304	673	6	7
$\mathcal{B}_2$	41,941	309	689	9	14
$\mathcal{B}_3$	51,056	316	702	15	29
$\mathcal{B}_4$	60,166	319	714	26	50
$\mathcal{B}_5$	69,271	321	724	32	82
$\mathcal{B}_6$	78,375	323	730	49	112
$\mathcal{B}_7$	87,468	324	736	63	162
$\mathcal{B}_8$	96,555	324	737	83	209
$\mathcal{B}_9$	105,642	324	742	132	336
$\mathcal{B}_{10}$	114,726	324	742	152	396
$\mathcal{C}$	22,416	9,827	548	3,992	455

Since we use the term "correctness" to refer to the correctness with respect to human judgment, this value does not have to be related to logical satisfiability. Rather, a value of 1 would mean that only axioms were removed that a human ontology engineer also considers to be erroneous. For the second DBpedia dataset, an ontology engineer inspected the list of axioms removed from one of the incoherent ontologies regarding their correctness. Thus, we were able to compare the performance of the approaches regarding the actual correctness of the resulting ontology. For the third dataset, we decided not to do this kind of evaluation since the effort for fully understanding the automatically generated ontology would have been too high.

All experiments were performed on a system with an Intel Core i7 3.4GHz with 32GB of RAM. As mentioned, for the Markov logic-based approaches, we used the RockIt[1] MAP query engine which in turn uses the ILP solver Gurobi.[2]

8.3.2 Results

Applied to the first ontology, the approaches using explanations for incoherences performed similar, all of them removing the same 10 axioms from the ontology and having similar runtimes of about 40 seconds. During the evaluation of the removed axioms, the correctness turned out to be only at 0.4. Approach A4 run 12 seconds and removed only 6 axioms with a correctness of 0.

We examined the low correctness value and the high overlap regarding removed axioms and discovered that it comes from the fact that the debugged ontology has one central point of incoherence which is centered around the disjointness of orga-

[1] `https://code.google.com/p/rockit/`, Version 0.3.228

[2] `http://www.gurobi.com/`, Version 5.6.0

nization and educational institution classes. For instance, the class `Library` is a subclass of both `Organisation` and `Building` which are marked as disjoint in the disjointness gold standard. Since the subclass axiom, which is the actual cause of the overall problem, was contained in the base ontology, the approaches were not allowed to remove it and tried to find a minimal set of axioms mitigating the problem. Seemingly, the approaches based on explanation generation were not able to find a minimum cardinality set of axioms to remove since the Markov logic-based method found a smaller set of axioms. The latter however did not remove any axioms whose removal was justified according to human assessment. During its evaluation, one disadvantage of not computing explanations was discovered. Fully based on Markov logic, there is almost no possibility to reconstruct the reasons for the removal of certain axioms which makes human intervention hardly possible whereas the access to explanations enables humans to better track and understand the reasons for certain removals. In particular, this is relevant in interactive ontology debugging scenarios.

For the second dataset, we manually assessed the removed axioms for ontology $\mathcal{B}_5$. Since this ontology contains more different axioms and more potential incoherences than $\mathcal{A}$, there are much more variations in the number of removed axioms and their correctness than for the first ontology. The results are given in Table 8.2.

Table 8.2: Results for approaches on ontology $\mathcal{B}_5$

Approach	Runtime	# Removed axioms	# Correct axioms	Correctness
A1	12,502	54	43	0.80
A2	15,029	46	40	0.87
A3	19,006	106	73	0.67
A4	23,864	98	75	0.77

The greedy approaches performed better regarding the number of removed axioms and the correctness. They only removed about half of the axioms the MLN-based approaches remove. This was probably caused by some axioms with lower confidence being removed by the MLN methods but again hard to track down because of the black box characteristics of the MLN approaches. For the smaller ontology, the greedy approaches were even better with respect to the runtime. However, the MLN-only method was more capable of handling an increasing number of axioms as shown in Figure 8.2. The runtimes of the explanation-based approaches increased more significantly than the the runtimes of the MLN-only approach. This was caused by the increasing number and size of explanations and the time required for collecting them beforehand. Furthermore, the number of explanations had a more drastic influence on approach A2 since its runtime is not linear in the number of explanations in contrast to approach A1.

The performance advantage of the MLN-only approach was even more drastically shown by the experiments on the third ontology. Since the ontology contained nearly 10,000 classes the explanation generation for all incoherences was not pos-

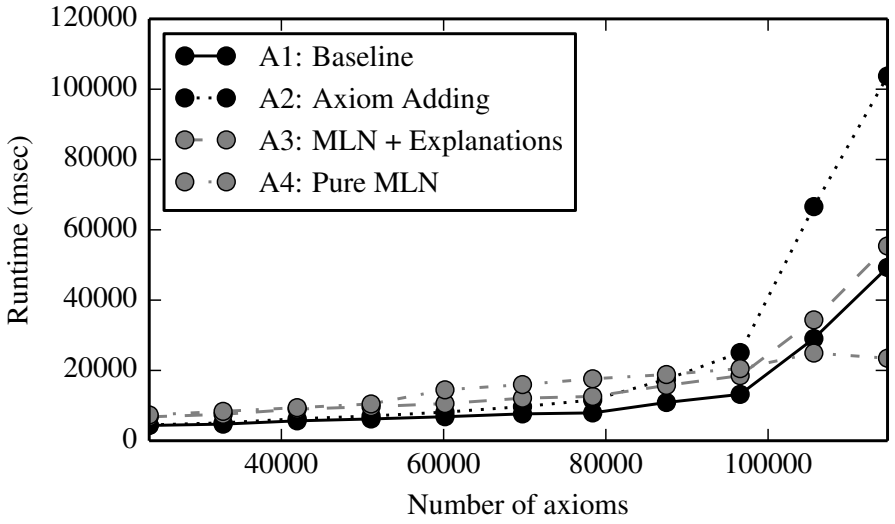


Figure 8.2: Comparison of the runtime behavior of the different repair approaches

sible in reasonable time.[3] However, this limitation was also caused by our TRex implementation that is optimized for a lower number of classes. Because only approach A4 does not depend on the explanation generation, it was the only one applicable to this dataset. With a total runtime of about 32 seconds and a total number of removed axioms of 3,097 it showed a performance suitable for most practical use cases, especially when considering the high number of incoherent entities in the ontology. This qualifies the approach for the usage on large ontologies potentially containing many incoherences and for cases where no human intervention is desired. Additionally, compared to the original results of Qi et al. [80], the MLN-only approach was able to process the whole ontology at once instead of having to add additional axioms in chunks, then checking and repairing the ontology and add another chunk of axioms. Our approach also had a lower runtime than the one reported for the original approach. Interestingly, we removed more axioms for reaching a coherent ontology. Both aspects could also be influenced by the iterative addition of axioms.

8.4 Conclusion

In this chapter, we presented and compared four different approaches for repairing incoherent ontologies. We focused on learned ontologies which exhibit common characteristics. In particular, many learning approaches assign their internally generated confidence values to the learned axioms. The usage of these confidence

[3] We aborted the computation after one hour.

values helps to assess which axioms to remove and which to leave in the ontology.

Besides more traditional greedy repair approaches, we also evaluated two approaches using Markov logic networks. Notably, we adapted a Markov logic-based approach using the schema-only entailment rules we developed before for finding incoherences in learned ontologies. During the experiments, the explanation-based approaches showed promising results, however, the explanation generation turned out to be the major bottleneck especially for ontologies containing large numbers of schema entities.

These shortcomings are not shared by the Markov logic-only approach which does not rely on externally generated explanations. This approach showed promising results with respect to both runtime and scalability. However, the main problem of avoiding the generation of explanations shows up when human engineers try to find out the reasons for specific repair actions. Since only the actual removals are available but no further information is given, it is hardly possible to assess how reasonable a removal was from a human point of view. During the evaluation, repairs performed based on the actual explanations of incoherences were much better to trace back to their roots and to check regarding their validity. Moreover, the approach that combined both generated explanations and Markov logic networks also showed some aspects of this problem. Though the explanations were accessible to human assessment, the actual removal decision was harder to understand than for the greedy approaches. We suppose this to be caused by the fact that the Markov logic approach tries to reach a globally optimal solution while the greedy approaches prefer local solutions. The latter seems to be more easily understandable to humans which have difficulties to mimic the global view on the ontology that is used by the algorithm. A more comprehensive study and discussion of this effect seems to be of interest.

Combining these findings, we argue for the usage of greedy, explanation-based repair approaches in cases where human engineers are expected to be involved in the process. For very large ontologies and in cases where human consolidation is unlikely, approaches as the pure Markov logic approach seem to be qualified.

Chapter 9

Schema-Based Error Detection

In the previous chapters, we gradually approached a full process of learning expressive ontologies for the purpose of using its logical information to detect errors in Linked Data. This chapter now examines the actual error detection based on the learned ontologies and reports on the manual evaluation of the gathered error detection results. For this purpose, we consider the ontology axioms to model constraints that hold in the real world. Thus, violations of these axioms point to discrepancies of the data with the real-world scenario and are potential data problems that should be examined closer. In the context of using the logical schema to detect errors, the amount of data contained in a typical Linked Data repository poses some additional challenges. The currently available OWL reasoner are not able to handle most Linked Data datasets due to their very large number of instances contained in these datasets. Hence, letting a reasoner detect inconsistencies that occur in the dataset when combined with the newly learned expressive schema is not practically possible. There are two possible solutions for this problem. First, a reasoner could be applied only to the schema for getting the implicit schema knowledge which is then used to identify problematic parts in the Linked Data dataset. However, the application of a reasoner could lead to an extremely high amount of implicit knowledge. This large amount of information would lead into performing many checks on the dataset and thus reduce the overall performance of the detection step. Hence, in this work, we resort to a different variant. Instead of using a reasoner for achieving full coverage of all derivable axioms, we use a pattern-based approach. For each relevant schema axiom and also specific combinations of schema axioms, possible arrangements of triples are identified whose existence shows a violation of this part of the schema and thus a potential problem in the data. The dataset is then checked for occurrences of these identified patterns. Since it is one of the most wide-spread and standardized means of accessing Linked Data sources, SPARQL is often used for expressing the relevant patterns and, by querying a SPARQL-enabled endpoint, can be used for finding triples showing the potential violation. Compared to a reasoning-based method, the pattern-based detection of errors might not be complete in a sense that it exhaustively detects all

violations in the dataset. However, this is made up by the direct applicability to many endpoints and by the fact that only data violating a pattern has to be transferred from the endpoint. Furthermore, more complex patterns can to some degree compensate for this at the expense of longer runtimes when evaluating the query. For example, consider a schema containing the small dataset:

```
:Dog rdfs:subClassOf :Animal.
:Animal a rdfs:Class.
:Rex a :Dog.
```

According to the semantics of subclass relations, the instance `Rex` is both a `Dog` and an `Animal`. Since this is not stated explicitly or materialized to the instance, simple queries just asking for all instances of `Animal` would not return `Rex`. For this specific case of type assertions inferable from the type hierarchy, a SPARQL query like

```
SELECT ?instance WHERE {
  ?instance a ?cls.
  ?cls rdfs:subClassOf* ?relevantClass.
}
```

would return all instances assigned to the relevant class itself or to a class defined to be one of its subclasses. In this case, the transitivity of the subclass relation is handled by the property path feature introduced in SPARQL 1.1.[1] As shown in this example, it is possible to achieve a limited degree of inference by carefully crafting the actual query. Nevertheless, this might lead to more complex and hence worse performing queries which is why the inference has to be balanced against the query performance.

In the following, we first provide an overview on other works using ontology axioms for detecting errors in datasets and also on other approaches that allow to detect logical errors in datasets. These approaches include both pattern-based methods like described above and logics-based ways for error detection. Since we rely on the RDFUnit tool to actually query the dataset for violations of the schema axioms, we introduce the function of this tool in Section 9.2 and describe the additional patterns we added to more effectively use the learned axioms for error detection. Afterwards, Section 9.3 describes the settings of our experiments and their results in these different settings. Here, we also give details on the actual influence of the ontology's coherence on the error detection performance. Finally, we conclude this chapter by summing up the lessons learned regarding the usage of learned ontologies for debugging Linked Data datasets.

9.1 Related Work

As already described in the introduction, there are two major ways of detecting errors in data based on ontological knowledge. Most works that concentrate on general data and ontology error detection rely on the usage of logical *inference-based*

[1] `http://www.w3.org/TR/sparql11-query/#propertypaths`

methods. On the contrary, recent works dealing with error detection in Linked Data show the tendency to employ *pattern- or rule-based* techniques that allow a more efficient application to large datasets.

The inference-based approaches rely on reasoning systems to detect inconsistencies caused by a conflict between the instance data and the ontological schema of the dataset. This requires a schema expressive enough to actually cause inconsistencies in the dataset. In most cases, this means that the schema has to contain at least some kind of negation such as disjointness. Furthermore, these approaches are limited to smaller datasets. This is caused by the fact that most methods need to generate explanations for discovered inconsistencies. As also seen in Chapter 7, explanation generations suffers from poor performance as do reasoning systems like Pellet [88] and Hermit [44] in general when applied to large datasets. This limits the overall performance. However, for ontologies of limited size, the ontology engineering tools Protégé and SWOOP provide support for detecting inconsistencies not only in the terminological part of the ontology but also in the instance data. Lehmann and Bühmann [65] proposed and implemented a complete workflow which enriches an ontology with subsumption or equivalence axioms learned by the DL-Learner approach [64]. After adding new axioms, the resulting ontology is checked for incoherence or inconsistencies. If there are any problems detected, the user is asked to edit or delete the axioms contained in the explanation of the unsatisfiability. To also support larger datasets, the explanations are only computed on a subset of the full data that is determined based on the unsatisfiable class itself. They present examples of applying their workflow on a number of smaller ontologies and also on the DBpedia dataset for which they are able to find some wrong property assertions. In contrast to our experiments, which we present in the following, the work by Lehmann and Bühmann more concentrates on keeping the original data consistent with the newly learned axioms. They do not evaluate how many actual data errors are detected. Furthermore, they only focus on learned subsumption and equivalence and do not consider additional axiom types like class disjointness that would be important for detecting errors in data.

Baclawski et al. [12] presented a way to check consistency of ontologies provided in different representation formalisms employing the Prolog language for inference. Their implementation also allows to detect the cause of possible inconsistencies. However, they do not support OWL what makes this approach hardly usable for typical Linked Data sources and, moreover, there is no information about the performance of this approach which makes it hard to assess if it could be applied to large datasets.

As we already described in Chapter 3, OWL relies on the open-world assumption which means that it is not possible to prove a fact as wrong based on a failure to prove it to be correct. Though this is intended and reasonable for the distributed knowledge representation scenario, it limits the usability in other scenarios. For example, when a single organization is responsible and in control of the modeling process, it might want to check the adherence to certain constraints like existence of a specific property for all instances. To address this and inspired by works in

the area of deductive databases, Tao et al. [90] introduce an alternative semantics for OWL in which axioms are interpreted as integrity constraints according to a closed world semantics and hence can be used directly for checking the validity of a dataset without relying on negation contained in the ontology. They also provide a mapping of the possible constraints to a corresponding SPARQL query to test for violations of the constraint so that SPARQL is sufficient to check all supported constraints.

Following the same basic idea of detecting violations of constraints, additional pattern-based approaches have been proposed. These mostly differ from the work by Tao et al. in their support for different axiom types. Furthermore, the largest part of these approaches has been developed with application in the field of Linked Data in mind paying special attention to performance on large amounts of data.

Peron et al. [79] proposed a pattern-based approach for inconsistencies in RDF datasets that concentrates on assignments. They search the dataset for conflicts arising from property assignments. For example, the method considers it to be a domain inconsistency if a subject instance of an object property does not belong to the defined domain of the property. Equivalently, range inconsistencies are defined in cases where the range of an object instance does not belong to the property's range. This comprehension of inconsistency is similar to the integrity constraint semantics employed by Tao et al. For the detected problems, solutions are proposed including the inference of new class assertions for the instance or using a less specific class in the domain or range restriction. In addition, they introduced a way of analyzing discovered problems by means of a histogram that is based on the actual distribution of classes in the domain or range of a property. This analysis allow to find a most specific, more general class that fixes a detected inconsistency. Peron et al. evaluated their method on DBpedia 3.6 for finding errors in the data by discussing several examples of commonly encountered errors.

The error detection method introduced by Töpper et al. [92] is also concentrating on detecting problems caused by property restrictions. They learn additional domain and range restrictions for the ontology by determining the types most commonly used with in the corresponding position with a property. Similar to our work presented in this chapter, Töpper et al. do not rely on an integrity constraint semantics but instead enrich the ontology automatically with disjointness axioms that are learned by means of a vector space model as already described when presenting related work in Chapter 5. For detecting errors, they consider each property assertion and check whether the class learned as domain or range is disjoint to the class assigned to the subject or object instance, respectively. If such a conflict is detected, they generate suggestions on how to fix the conflict, e.g., by removing the learned disjointness axioms or fixing the actual data. All in all, this work is very similar to the work we are presenting here. The main difference is that we also evaluate based on additional axiom types like property disjointness not covered in the work of Töpper et al. and that their approach is directly integrated into the DBpedia extraction framework leading to a much higher coupling to DBpedia as dataset.

For detecting errors in instance data, Sheng et al. [87] exploited the subclass relations defined from classes in the DBpedia ontology to classes from the UMBEL ontology. Since UMBEL defines disjointness relations between many of its classes, the links from the DBpedia ontology allow to deduce disjointness axioms holding for pairs of DBpedia ontology classes. This gathered disjointness is then used to find inconsistencies in the DBpedia dataset by checking a set of patterns like instances assigned to both classes of a disjoint class pair. However, for checking property domain and range, the authors also employ an integrity constraint-like approach where a difference between the stated class memberships of an instance and the domain and range classes of a property already lead to a conflict. Furthermore, only statistics on the number of discovered conflicts are presented but no further evaluation of the correctness of the conflicting triples is performed, thus missing an important point when considering actual data quality.

Very recently, Kontokostas et al. [59] proposed to automatize checking of Linked Data by applying the concept of test cases from software development. They argue that, in contrast to software testing, the advantage in the Linked Data domain lies in the semantic information already formalized in schemas and ontologies. To exploit this knowledge, they introduce the concept of test-case auto generators (TAG) that create test cases based on the occurrence of specific axiom patterns in the ontology. For example, one TAG instantiates a test case for each pair of disjoint classes so that the test case searches for instances assigned to both classes which would point to a data problem. Another auto generator creates tests for finding mismatches between the classes an instance is belonging to and the domain and range restrictions of properties the instance is used with. Thus, this is another example of applying the integrity constraint semantics. More domain-specific test cases, which are not deducible from schema knowledge, can be created manually. Since this manual creation of test cases can require much effort, Kontokostas et al. suggest to collect manually created test cases for commonly used vocabularies and provide them in a central test case library for reuse. The full workflow is implemented in the RDFUnit tool. We are going to use it for our experiments later in this chapter and, thus, describe the approach and the tool in more detail in the next section. An evaluation of the test-based method is also performed by the authors. Since the DBpedia ontology is only of low expressivity, it gets enriched by means of the inductive ontology learning approaches as proposed by Bühmann and Lehmann [22, 23]. This shows a high similarity to the experiments that we perform in the following. However, the evaluation performed by Kontokostas et al. is limited to the number of discovered test case violations without assessing the correctness regarding the detection of actual errors.

A method that can hardly be assigned to the categories of being inference- or pattern-based is presented by Paulheim [76]. It tries to detect wrong links existing between datasets. For that purpose, each relevant link is represented by a multi-dimensional feature vector determined from the types assigned to the resources connected by the link or from properties used in conjunction with the resources. Using multi-dimensional outlier detection approaches, Paulheim finds links that

form outliers in this vector space, i.e., the link between both resources is atypical and thus suspicious to be wrong. In an evaluation, the links from the DBpedia dataset to the Peel Session and the DBTropes datasets were checked with this approach applying different outlier detection approaches and the results turned out to be promising. The disadvantage of this approach lies in the fact that the decisions regarding which links are considered as wrong might be less understandable since there is no intermediate step that formalizes the typical usage of certain properties. In particular, the multi-dimensionality of the created feature vectors could hinder a more fine-grained analysis of the problem.

For detecting errors in object properties, manual evaluation approaches like used in the TripleCheckMate system [60] and evaluated by Acosta et al. [2] would be also applicable in principle. However, these works are currently concentrating on detection errors in datatype properties.

9.2 Approach

As already motivated in the introduction of this chapter, we resort to a pattern-based approach for detecting conflicts between the learned ontological knowledge and the axioms contained in the dataset. Since the RDFUnit system already provides the required functionality for this purpose, we are building on it as a foundation for our experiments. To give an understanding for the RDFUnit workflow, we introduce the basic notions required for our application scenario in the following. Consequently, where not stated differently, the main source for the overview is the initial presentation of RDFUnit by Kontokostas et al. [59]. The RDFUnit framework tries to transfer the concept of test cases from the area of software development over to the domain of Linked Data. While test cases for software have to be manually specified, testing Linked Data shows some advantages concerning the creation of test cases regarding the logical structure. Since many Linked Data datasets use properties and classes for which additional knowledge is available in an ontology, this information can be exploited for test cases. Given a dataset, RDFUnit applies so-called test-case auto generators (TAG) that create test cases based on certain ontology axioms of patterns of axioms. For this purpose, a TAG consists of a detection and an execution part. The former discovers the relevant axioms in the ontology by means of SPARQL queries against the ontology structure. Based on each result of the detection query, in the execution part, a test case is instantiated. This test case itself also is a SPARQL query that gets applied to the dataset and retrieves all data which is violating a test case. As an example, Kontokostas et al. describe the generation of test cases based on class disjointness axioms. By applying the detection query

```
SELECT DISTINCT ?T1 ?T2 WHERE {
  ?T1 owl:disjointWith ?T2.
}
```

to the ontology, all disjoint class pairs are detected in the ontology. For each such pair, a test case is generated according to the following pattern

```
SELECT DISTINCT ?s WHERE {
  ?s rdf:type %%T1%%.
  ?s rdf:type %%T2%%.
}
```

where the placeholders `%%T1%%` and `%%T2%%` get replaced by the URIs of the disjoint classes. Hence, the resulting query retrieves all instances that are assigned to two disjoint classes at the same time. The different violations of all test cases are aggregated and displayed to enable a human knowledge engineer to check the results and perform the required fixes to the data for improving its quality.

The default set of TAGs in RDFUnit also uses domain and range axioms for generating test cases. However, these test cases are treating domain and range according to constraint semantics and do not consider class disjointness. Thus, actually correct links might show up as potential errors only based on the fact that instances connected by means of a property do not have one of its domain or range (sub-)classes assigned. Incorporating class disjointness into the domain- and range-based detection thus can allow more fine-grained test cases.

For our experiments with learned disjointness, we defined additional TAGs that do also consider class disjointness when creating test cases based on domain and range axioms. We are shortly describing these patterns in the following.

DOMAINCLASS This generator detects properties p whose domain class is defined to be disjoint to a class C. The resulting test case checks for instances which are used as subject of p and at the same time are assigned to have type C.

RANGECLASS This generator performs the same detection as the previous one except for considering the classes defined as range of properties.

DOUBLEDOMAIN This generator creates test cases from properties p_1 and p_2 whose domain classes are defined to be disjoint. The test case detects instances which are used as subject of both p_1 and p_2.

DOUBLERANGE Similar to the previous case but detecting properties p_1 and p_2 whose range classes are defined to be disjoint. Instances violate the resulting test case if they are object to both properties at the same time.

DOMAINRANGE Finds properties p_1 and p_2 where the domain of p_1 is set to be disjoint from the range of p_2. The resulting test case detects instances that are at the same time subject of p_1 and object to p_2.

In addition to these patterns defined by us, we also use the following patterns defined by Kontokostas et al.

OWLASYMMETRICPROP Detects properties that are set as asymmetric. The resulting test case checks for instances a and b whether the property holds in both directions.

OWLDISJP Detects pairs of properties that are defined to be disjoint. The resulting test case checks for pairs of instances that are connected by both properties at the same time.

OWLIRREFLEXIVE Detects properties that are set as irreflexive. The resulting test case checks for instances that are connected to themselves by means of such a property.

9.3 Experiments

We performed experiments in three different settings for assessing the possibilities of using learned expressive schemas to detect logical errors in Linked Data datasets. All experiments have been conducted on the DBpedia dataset in version 3.7 with the RDFUnit tool for detecting potentially problematic data. Using RDFUnit means that we had two main parameters: the ontology used to deduce the test cases from and, in addition, the rules to actually deduce the test cases from the schema information. For our experiments, we concentrated on the first parameter, the ontologies, and only adapted the set of TAGs so that it best matched the ontologies for the given experiment. We selected the TAGs to use out of the default set of generators provided with RDFUnit and our additionally defined generators described in the previous section. For each experiment, we provide the set of activated rules in the corresponding descriptions.

Regarding the ontologies, we evaluated on those gained during the experiments in Part I. As a first experiment, the gold standard ontology for class disjointness described and created in Chapter 5 was used to generate test cases. This allowed us to determine the performance of recognizing errors in data assuming the availability of a manually created, high quality set of disjointness axioms. The second experiment was performed on the ontology enriched with learned disjointness based in the basic association rule mining algorithms proposed in Chapter 5. This experiment enabled us to compare the differences between a manually created and a learned schema regarding the error detection.

Besides the corresponding axioms themselves, when checking data by means of class disjointness, domain and range restrictions are also of great importance since they give possibilities to infer additional, implicit class assertions for the instances in the dataset. Since in the first two experiments the ontology was only enriched with class disjointness, the error detection had to rely on the already contained domain and range axioms. In a third experiment, we examined in how far the error detection result changes when we not only enrich an ontology by class disjointness axioms but by almost all axiom types supported by the association

rule mining-based approaches, including domain and range restrictions generated by the methods proposed by Völker and Niepert [96].

It is important to note that, during this evaluation, we not only considered detected violations on the instance level as error but also problems arising from the schema information contained in the original DBpedia ontology. This has been done because by providing a schema, the dataset commits itself to adhere to this schema and thus allows data consumers to gain advantages from its availability. In contrast, erroneous schema information can cause problems when the consumer relies on its information to understand the data contained in the dataset or to infer additional knowledge from the schema. Thus, we treat detecting of schema errors at the same level of importance as detecting errors in the instance data.

9.3.1 Disjointness-Enriched Ontologies

For the ontology enriched with manually created class disjointness axioms and the ontology enriched by automatic means, we performed equivalent steps during the experiments. As gold standard ontology, we resorted to the dbpedia100 gold standard described in Chapter 5. Hence, we only included disjointness axioms for which all three annotators agreed on the disjointness of the two classes which led to $36,491$ class pairs being disjoint. Regarding the evaluation of an ontology enriched with learned disjointness, we used the ontology learned from the DBpedia dataset at a confidence level of 0.95. This confidence level caused a total number of $39,135$ class pairs to be disjoint. For each ontology, we performed one RDFUnit run using the ontology as the schema to create test cases based on the TAGs listed in Table 9.1. This table also provides the results of the error detection for both experiments.

Table 9.1: Number of problems detected by test case type using the disjointness enriched ontologies

Test case type	Gold standard	Learned
DOMAINCLASS	$55,557$	$83,179$
DOMAINRANGE	$152,194$	$202,988$
DOUBLEDOMAIN	$122,192$	$217,525$
DOUBLERANGE	$375,937$	$476,657$
RANGECLASS	$101,935$	$132,928$

These results show that the gold standard-based run detected considerably less conflicts in the dataset which can be directly attributed to the higher number of disjointness axioms contained in the ontology. However, since the ontology-level modeling and the actual usage of ontology entities in the dataset are not always guaranteed to coincide, these numbers on their own do not give an insight into the actual error detection performance. To further examine in this direction, we manually evaluated a sample of the detected problems regarding whether there

actually existed a data problem. This manual evaluation was performed on a set of 250 samples per ontology, assembled by randomly selecting 50 detected violations for each test case type. According to the manual annotation of the results shown in Table 9.2, for both approaches more than 60% of the detected errors showed an actual problem in the data. The detection based on the manually engineered disjointness axioms, performed better regarding its total correctness and also with respect to all except for one test case types.

Table 9.2: Statistics on violations that correctly indicated quality issues.

	Gold standard		Learned	
Test case type	#	%	#	%
DOMAINCLASS	29	58.0	26	52.0
DOMAINRANGE	36	72.0	36	72.0
DOUBLEDOMAIN	22	44.0	16	32.0
DOUBLERANGE	38	76.0	41	82.0
RANGECLASS	46	92.0	36	72.0
Total	171	68.4	155	62.0

For both experiments, the test case types which were heavily based on domain restrictions performed the worst. This can be attributed to the way triples are created in DBpedia. As described in Section 2.2.3, the largest part of the instances contained in DBpedia is generated from infoboxes that are mapped to a type in the ontology. A triple using an instance as subject can be only generated when there is a field in the instance's infobox template that is mapped to the property. Thus, a triple being erroneous only because of a wrong subject instance cannot be introduced in a single infobox but only during the creation of the infobox mapping. This greatly reduces the number of errors introduced by accidental links for the test case types DOMAINCLASS and DOUBLEDOMAIN while we found the accidental links to be one major error class for other test case types.

During the manual evaluation, we coarsely categorized the discovered problems for further analysis. The already mentioned category of errors that were present on the instance data level, e.g., wrong links between instances or a wrongly used property in a triple, was assigned to 81 violations for the gold standard-based ontology and 72 violations discovered using the learned ontology. Most of these errors were caused by name confusion, i.e., linking to an instance having a similar name to the actually intended target, and extraction errors like extracting the country part of a "City, Country" information as an object on its own.

Another commonly discovered error was caused by wrong declarations of domain or range restrictions in the ontology of which we discovered 88 using the gold standard and 80 by means of the learned ontology. This shows the potential of not only detecting instance-level errors but also schema-level errors. We again see that error detection on the learned ontology performs slightly worse than on the gold standard but not by a large margin.

When we were more closely inspecting the violations that we did not label as data errors during our evaluation, we also discovered an accumulation for certain categories of errors. Interestingly, some of these problems were already discussed during the creation of the gold standard. For example, the disjointness between an `Organisation` and `Building`, like in the case of Library discussed in Section 5.1, showed to cause a number of violations that we did not considered as errors. However, the confusion between both classes might make it hard to actually handle the data in a consistent way. Thus, changes in the representation of such cases might lead to more usable data. One possible way of modeling would be the introduction of intermediate blank nodes representing the building and connecting the organization to the address defined in its infobox. The creation of such intermediate blank nodes could be triggered when introducing a property which cannot be directly assigned to the current instance due to the disjointness of the instance's class and the property's domain.

Another particularity of the ontology which led to several violations of disjointness, both manually created and learned, is the property `occupation`. In the ontology, this property is defined to have the domain `Person` and the range `PersonFunction` for assigning persons to their activities. However, this property is used in the dataset in many cases for describing an involvement in something without further restriction on its type. For instance, it is not only used to assign football players to `American_Football` but also to assign people an university they are somehow related to. Because of the wide spectrum of usages of the property `occupation`, stating a range of `owl:Thing` instead of a more specific one would be more appropriate.

Finally, some violations were caused by wrong disjointness axioms contained in the ontologies. As expected based on the results of the evaluation of the learned axioms, there were many more of these created by the learning method (49 violations). In contrast, for the gold standard, only one disjointness axiom turned out to be debatable. This axiom stated the disjointness between the classes `Person` and `Band`. During the manual creation of the axioms, this disjointness was introduced in view of a person not being a band himself but instead possibly being the only member of a band. In the DBpedia dataset this view is not supported since the properties with range `Band` are directly linking to single artists leading to four violations in total. Nevertheless, even for the learned disjointness, the greatest share of the violations is caused by one disjointness axiom which could be easily removed during data inspection in a more interactive workflow and thus would not produce any more violations to manually check. This also holds for most of the other violations occurring on the schema level. Fixing the cause of one of these violations could be easily used for filtering out all other violations having the same source and thus enable rapid fixing of errors combined with improving the overall ontology model.

9.3.2 Property-Enriched Ontology

In the previous experiments, we saw that disjointness-enriched ontologies are capable of indicating several types of errors in a dataset, also including problems in the existing ontology like wrong domain or range restrictions assigned to properties. Thus, even for ontologies that already are more expressive regarding the contained axioms, it might be fruitful to apply ontology learning approaches to enrich it even further since this gives more possibilities to detect errors in the base axioms. Furthermore, adding other types of axioms to the ontology could give extra chances for finding problematic data.

To further examine the potential of enriching the ontology with more expressive axioms for detecting errors, we performed a third experiment. For this experiment, we used GoldMiner on the DBpedia 3.7 dataset and not only generated additional class disjointness axioms but also performed enrichment as described in Chapter 6 and learned domain and range axioms as proposed by Völker and Niepert [96] and described in Section 3.3. We again applied a confidence threshold of 0.95 to all learned axioms. In addition to the class disjointness already learned before, we also enriched the ontology by object property disjointness, domain and range restrictions, asymmetric object property axioms and irreflexive object property axioms.

The detection of violations regarding the ontology axioms was again done by RDFUnit. Since RDFUnit was not able to handle the full enriched ontology, we reduced the number of axioms in the ontology as follows. First, we only included a random sample containing 20% of the full set of learned class disjointness axioms. Secondly, we filtered out all disjoint object property axioms having a confidence value of 1.0 since this confidence value indicates that there are no violations in the dataset which are detectable by the corresponding RDFUnit test case. Table 9.3 shows the activated TAGs along with the number of violations detected.

Table 9.3: Number of problems detected by test case type using the extensively enriched ontology

Test case type	Number of detected violations
DOMAINCLASS	$30,327$
DOMAINRANGE	$83,840$
DOUBLEDOMAIN	$103,922$
DOUBLERANGE	$211,785$
OWLASYMMETRICPROP	$5,115$
OWLDISJP	$109,674$
OWLIRREFLEXIVE	$3,886$
RANGECLASS	$28,565$

As for the class disjointness-enriched ontologies, we manually evaluated the violations regarding whether they are actual errors in the dataset or original ontology or if they are caused by errors in the learned axioms. For each test case type, we

randomly chose 50 violations and assessed them manually. The resulting correctness values are given in Table 9.4. In particular, these results show an improvement in accuracy when compared to the results on the experiment on the learned ontology before. The accuracy for test cases based on a domain restriction decreased which can be attributed to the same aspect as described for the disjointness-based test cases and in addition a number of wrongly learned domain and range axioms that amplified the problem further. For the range-based test case types, the ratio of violations that actually pointed to quality issues in the data increased considerably which shows the potential of employing additional enrichment steps on the ontology for detecting errors in the data. Interestingly, only three out of 63 test cases that did not point to actual data errors for the range-based test case types were caused by wrongly learned domain and range axioms. Furthermore, two out of three extra axiom types added for this experiment showed to perform very well for this purpose. Both asymmetric and irreflexive properties axioms show to deliver a very good base for finding additional problems in the dataset. In particular, these axioms allow to find wrong self-references for instances and also symmetric relations between two instances which indicate errors in the data like symmetrically used `subsequentWork` properties. The learned object property disjointness axioms perform worst of all enriched axiom types. Apparently, the chosen confidence level is not capable to select axioms well-suited for detecting errors. When applying a confidence threshold of 0.99 to the property disjointness axioms, we are able to increase the accuracy in error detection to 70%. It is important to note that the performance for detecting errors is not directly coupled to the quality of the property disjointness learning since we, as described above, removed all property disjointness axioms with a confidence of 1.0 and kept all those with a lower confidence.

As we did for the first two experiments, we also categorized the discovered data errors. Our findings here confirmed the initial results. In total, 185 violations were caused by wrong links between instances or wrongly used properties while 93 violations could be traced back to wrong domain and range axioms in the original ontology. Again major problems were caused by the relation between organizations and buildings or places as well as by the `occupation` which was used without any restriction to its object's type. The modeling of fictional characters turned out to be another problem in the data though this seems to be an issue also hard to solve in manual modeling since fictional characters cannot be guaranteed to follow any rules that hold for instances of real persons.

To assess the influence of the enriched ontology's coherence on the result, we applied the *Axiom Adding* ontology debugging approach presented in Section 8.2.2 to the enriched ontology. We determined which violations detected by RDFUnit would have been found beforehand by first applying ontology debugging. This was done by comparing the list of axioms removed during debugging to those causing violations in RDFUnit test cases. In total, the debugging approach removed 55 axioms to make the ontology coherent. As before, the set of removed axioms contained a number of actually correct axioms like the disjointness between the classes

Table 9.4: Statistics on violations that correctly indicated quality issues.

Test case type	Correctly detected errors	%
DOMAINCLASS	21	42.00
DOMAINRANGE	34	68.00
DOUBLEDOMAIN	14	28.00
DOUBLERANGE	47	94.00
OWLASYMMETRICPROP	49	98.00
OWLDISJP	19	38.00
OWLIRREFLEXIVE	46	92.00
RANGECLASS	48	96.00
Total	278	69.50

`College` and `University`. This would have prevented some violations from occurring which were caused by this disjointness, however, since colleges and universities are different from each other, this would have led to reducing the error detection capabilities. The same holds for the disjointness between `Organisation` and `Person`. Interestingly, five axioms that were removed for getting the ontology coherent were related to the area of organizations and places. In the previous section, we already described that the mixed modeling of both concepts in DBpedia led to some violations which we counted as not pointing to data errors. Thus, a prior application of ontology debugging methods would have led to improved accuracy values in this area. However, the disjointness between both classes might hold given a different way of modeling. This leads us to the conclusion that the application of ontology debugging methods can help to improve the error detection since it points to some problematic axioms before the actual test case generation. Nevertheless, a manual supervision seems to be advised to prevent the debugging step from removing correct axioms which later-on limit the detection accuracy.

9.4 Conclusion

In this chapter, we showed the potential of enriching ontologies with additional axioms for detecting errors in a dataset. By applying the inductive ontology enrichment methods proposed in the beginning of this work and additional statistical schema induction approaches, we first added a large amount of new axioms to the DBpedia ontology and then used these axioms for generating pattern-based test cases checking for data that violated these axioms. We hand-annotated a sample of the detected test case violations regarding the correctness of the data causing them which enabled us to assess the accuracy of using schema axioms for detecting errors. In addition to only using automatically generated axioms, we also used our manually created gold standard of class disjointness to check for violations in the dataset. This way we were able to show that learned axioms do not perform considerably worse than manually compiled ones. During the evaluation, we discovered a

number of shortcomings in the DBpedia ontology mostly related to the domain and range axioms. Apart from that, we gained insights into some additional challenges emerging when modeling the DBpedia ontology. For example, the representation of fictional characters that are much less constrained than normal persons and thus hard to fit into a common framework of axioms. We consider this another strength of combining inductive ontology learning with error detection since it allows to get new perceptions of the actual usage of the ontology entities in the data and thus enables the human engineer to improve the ontology and possibly the data based on usage patterns and actual demand.

Furthermore, through applying the test cases generated from the different axiom types contained in the ontology, we were able to identify wrong links and wrongly used properties in the dataset. Thus, not only improvements regarding the ontology can be fostered by this way of error detection but also actually wrong instance data can be found and corrected. If a test case violation turns out not to be caused by an error in the data but by an error in the learned axioms, the reliance on explicit axioms also shows advantages since the erroneous axiom can be marked as such and other violations raised due to this axiom can be ignored. This allows to reduce the manual effort for debugging the data.

All in all, we consider the approaches evaluated in this section well-suited for detecting errors. Based on our results, a semi-automatic workflow is greatly recommended since it permits to guide the process by early-on identifying wrongly learned axioms and, hence, improves the overall performance and also helps to enhance the ontology beyond the automatic enrichment. Although automatically debugging of the ontologies was able to sort out some axioms which wrongly led to violations, it also tended to remove axioms important for finding errors in the dataset. Thus, we also have to advise manual supervision for this step.

Part III

Detection of Numerical Errors in Linked Data

Chapter 10

Preliminaries: Outlier Detection

There are many scenarios in which it is fruitful to detect irregularities in data. For example, in physics experiments, measurement errors can cause such irregularities that should be detected and excluded from further processing to achieve more precise results. But also in everyday life, such irregularities can be relevant. Transaction data for stolen credit cards often show irregularities since the illegitimate possessors try to make as many high-profit transactions as possible in the short time before the card loss is recognized. Recognizing irregular data points can then trigger further actions like the removal of data points caused by measurement errors or to blocking of the credit card to prevent further fraud.

The research in the direction of *outlier detection*, sometimes also called *anomaly detection*, is focused on the development of methods for detecting such data points, more specifically on "finding patterns in data that do not conform to expected behavior" as formulated by Chandola et al. [25]. These unexpected patterns are most often referred to as *anomalies* or *outliers*. Since we employ outlier detection as an underlying method in Chapter 11, we will give an overview on its basic notions and ideas as well as on the methods for detecting outliers which we are going to use. The main part of this overview is based on the aforementioned work by Chandola et al. [25] which is the main resource throughout this section unless stated otherwise.

The process of detecting outliers in a dataset can be coarsely divided into two steps. First, knowledge about what qualifies outliers in the dataset has to be acquired. Secondly, this knowledge has to be used to efficiently find the actual outliers in the dataset. Obviously, these two steps are not fully independent of each other since the kind of knowledge gained during the first step greatly influences the way of actually detecting the outliers.

Chandola et al. propose a multi-level categorization of outlier detection methods which we describe in the following.

The first dimension to categorize a method is the degree of *supervision* it needs. As in machine learning, this supervision shows itself in the kind of data which is available in addition to the input dataset in which the outliers should be detected.

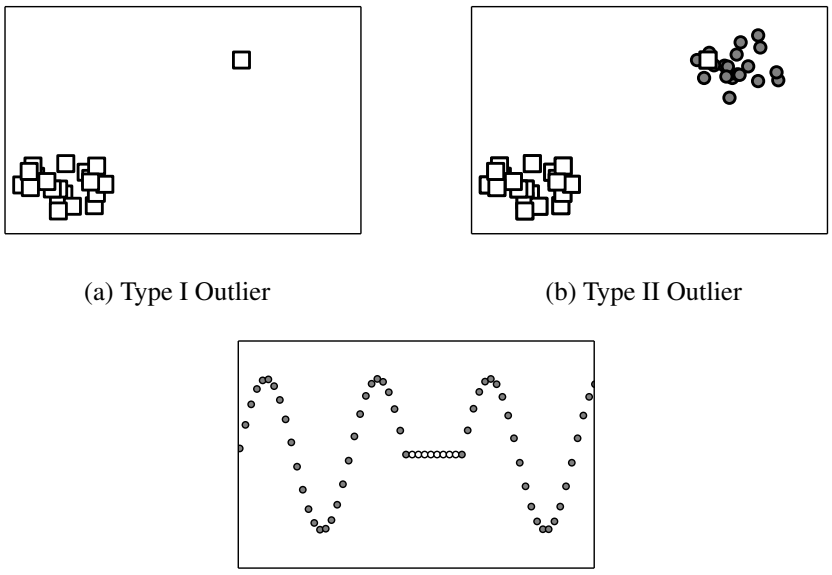
The most effort is required for *(fully) supervised* methods. Such methods need labeled data in which for each datapoint there is an annotation that specifies whether the datapoint is an outliers or a normal value. A second kind of methods are the *semi-supervised* ones. These methods also require labeled data but only regarding one class of labels, i.e., only the normal values or only the outlier values have to be annotated. Consequently, this means that a dataset only containing normal values or only containing outliers is a valid input for these methods. The third category consists of *unsupervised* methods which do not get any data in addition to the input dataset. Unsupervised methods have to infer the characteristics of normal values and outlier values directly from the input dataset.

The second dimension reflects the *type of outliers* a method can detect. *Type I Outliers* are individual data points which are outliers regarding the majority of values in the dataset like the upper right point in Figure 10.1a. *Type II Outliers* are similar to the first type but, in contrast to this, they are only outliers compared to the other values in a specific context. Independently from the context, these outliers would not necessarily show up as outliers. This case is depicted in Figure 10.1b in which the white data point in the upper right is not detectable as outlier regarding the whole dataset. Only when ignoring the grey data points and thus concentrating on the white "context", this point can be seen as an outlier. Finally, *Type III Outliers* differ from the first two types in the fact that they are not only single outlying values but they are a set of data points which together exhibit anomalous patterns though the single data points might not qualify as outliers on their own. This type is illustrated by Figure 10.1c where the major part of the data points follows a sine function but the white points in between do not show this behavior.

The third categorization aspect is based on the type of output the methods provide. In the *binary* scenario, assessments are provided, i.e., for each datapoint it is stated whether this point is an outlier or not. A more fine-grained assessment is delivered by *scoring* methods which assign each datapoint a numerical values which states its degree of "outlierness". By applying a threshold on the scores generated by a scoring method, the output of these methods can be easily transformed into a binary assessment.

In the following descriptions, we will concentrate on unsupervised methods which are able to detect Type I and to some degree also Type II outliers in a dataset. This is motivated by our later use case in which annotated data is not available and Type III outliers are not to be expected. Regarding the output of the methods, we prefer scoring methods which provide more flexibility in assessing outliers and thus give more possibilities to choose between precision and recall in the detection of outliers. However, as described above, the transition between scoring and binary methods is not totally strict.

As already mentioned, unsupervised outlier detection methods can only rely on the input dataset for telling outlier and normal data points apart. For this purpose, all of these methods assume that the normal values are the majority in the dataset.



(a) Type I Outlier

(b) Type II Outlier

(c) Type III Outlier

Figure 10.1: Examples for different outlier types

10.1 Statistical Outlier Detection

One common method of performing outlier detection employs parametric statistical methods. These methods assume that the expected behavior of the values follows a specific probability distribution. When even assuming that the actual underlying distribution is known in advance, the problem of determining its exact characteristics is only a matter of finding the corresponding parameters to instantiate the distribution. In cases where the input dataset is sufficiently large, its values can be used as a sample on which the parameters are computed. For instance, given values which are expected to follow a univariate (or one-dimensional) normal distribution, the sample mean μ and their standard deviation σ can be used to fully instantiate the corresponding density function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Given this density function, it is possible to determine the probability of a random value falling into a given interval. A simple outlier detection is based on the fact that this knowledge also allows to specify the probability of a value not to fall into a given interval. This interval is known as *confidence interval*. For the normal distribution with parameters μ and σ, the interval is defined by $[\mu - c \cdot \sigma, \mu + c \cdot \sigma]$ where c depends on the level of confidence, i.e., the probability for a value to fall into the interval. Setting $c = 3$ leads to an interval in which more

than 99.7% of the normally distributed values fall while $c = 2$ defines an interval with a confidence of 95.4%. An outlier detection approach based on this method would mark values as outliers which do not belong to the confidence interval for a given c. Obviously, the higher the choice for the value c, the higher the probability that a value not in the interval is in fact an outlier value. This approach leads to a binary classification of values into outlier and normal values but it is easily possible to produce outlier scores using the actual probability distribution. For this purpose, we can use the probability that a value v drawn from the distribution is greater or equal to the currently considered value, i.e., $P(X \geq v) = 1 - P(X < v)$. Thus, lower values point to a higher confidence that the value is an outlier. As a slightly more efficient approach which does not include the actual distribution, it is also possible to compute the smallest constant c so that v is included in the interval $[\mu - c \cdot \sigma, \mu + c \cdot \sigma]$ by just using $c = |\frac{v}{\sigma}|$. Obviously, higher c mean that v is less probable and thus being more suspect of being an outlier.

The main advantage of these statistical methods is their simplicity and computational efficiency. However, in practice, they expose certain weaknesses. For instance, parametric approaches require a knowledge about the distribution describing the generation of the input values which might not be available in many scenarios. This can be solved by using non-parametric approaches which, for example, use histograms inferred from the input values to represent the distribution or employ kernel density estimation to infer continuous probability distributions based on the input values. Furthermore, when using parametric methods, the computation of the parameters from the input data can be influenced by outliers contained in the data. For example, outliers can lead to a mean value highly different from the the mean of the normal data. This influence is less severe when the input dataset is large and contains enough normal values but it might be nevertheless noticeable depending on the strength of outlierness.

10.2 Nearest-Neighbor-Based Outlier Detection

Another category of outlier detection approaches is based on the assumption that outliers can be identified by comparing their characteristics to those of their nearest neighbors. Chandola et al. distinguish two kinds of such approaches. The first kind considers a specified number of nearest neighbors of a data point and categorizes this point as outlier if the distance to the neighbors is higher than a given threshold or differs considerably from other points' distances to their neighbors. According to Chandola et al., these approaches perform well for detecting global outliers, i.e., outliers which can be detected from the full dataset. However, they perform poorly when the dataset contains regions of different densities. This shortcoming is addressed by approaches which compute densities for regions of the dataset and use these to assess whether a data point is an outlier. This problem is depicted in Figure 10.2. In this example, global outlier detection approaches would only detect p_1 as an outlier since it has a high distance to all of its neighbors. However, p_2 also

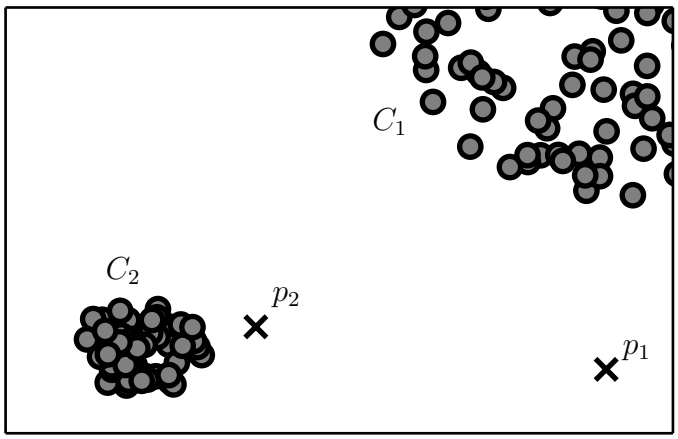


Figure 10.2: Example in which global outlier detection approaches would not be able to detect p_2 as an outlier (equivalent to example by Breunig et al. [20])

shows a deviation from the behavior of all other points in the dataset since it has a considerably higher distance to its neighbors compared to the points of C_2. At the same time, the distance to its neighbors from C_2 is similar to the distance of the points in C_1 to each other. However, p_2 does not actually belong to this recognizable cluster of values. A more region-aware outlier detection method would be able to detect p_1 as well as p_2 as outliers in this example.

Local Outlier Factor

One representative of this second category is the *local outlier factor* (LOF) method proposed by Breunig et al. [20]. It adapts methods commonly known from density-based clustering algorithms like DBSCAN [34]. As also done by Breunig et al., we denote the distance between two objects p and q by $d(p, q)$ and write $d(p, C)$ for the minimum distance of an object p to an object q in a set C, i.e., $d(p, C) = \min\{d(p, q) \mid q \in C\}$. The full dataset is represented by D.

One of the base notions used in context of LOF is the so-called *k-distance of an object* p.

Definition 11 (k-distance)**.** For any positive integer k, the k-distance of an object p is given by the distance $d(p, q)$ between p and an object q for which

1. $\sharp\{q' \in D \setminus \{p\} \mid d(p, q') \leq d(p, q)\} \geq k$ and
2. $\sharp\{q' \in D \setminus \{p\} \mid d(p, q') < d(p, q)\} \leq k - 1$

holds. The k-distance of an object p is denoted by $k\text{-distance}(p)$.

Thus, the k-distance of p is the minimum distance (condition 2) in which p has at least k neighbors (condition 1). Based on this, the *k-distance neighborhood of an object* p is defined.

Definition 12 (k-distance neighborhood)**.** The k-distance neighborhood of an object p is given by

$$N_{k\text{-distance}(p)}(p) = \{q \in D \setminus \{p\} \mid d(p,q) \leq k\text{-distance}(p)\}$$

It contains all objects in D whose distance to p is not greater than the k-distance of p. These objects are called *k-nearest neighbors of p.*

If D contains multiple values which have the same distance to p, this definition implies that the cardinality of $N_{k\text{-distance}(p)}(p)$ can be greater than k. As Breunig et al., we shorten $N_{k\text{-distance}(p)}(p)$ to $N_k(p)$ if this does not lead to confusions.

These notions provide the basics to specify another central notion required for defining the computation of an object's LOF: the reachability distance of two objects p and q for a specified value k.

Definition 13 (Reachability distance)**.** Let k be a natural number. The reachability distance of an object p with respect to an object q is given by

$$\text{reach-dist}_k(p,q) = \max\{k\text{-distance}(p), d(p,q)\}$$

This distance introduces a smoothing into the actual distance between both objects. For large distances between p and q, the actual distance is used while for smaller distances the k-distance of p is relevant. Since for objects more close to p a higher fluctuation in the distance is expected, this leads to the mentioned smoothing. In Example 4, we demonstrate the three introduced notions.

Example 4. Assume a 2-dimensional dataset D containing the following values as also depicted in Figure 10.3.

$$p_0 = (1,1) \qquad p_1 = (3,4)$$
$$p_2 = (4,4) \qquad p_3 = (4,3)$$
$$p_4 = (5,3)$$

From these examples, we get the following k-distances for p_0:

- 1-distance$(p_0) = d_1 = 3.61$
- 2-distance$(p_0) = d_1$, since this distance includes both p_1 and p_3
- 3-distance$(p_0) = d_2 = 4.24$

Furthermore, the corresponding k-distance neighborhoods are $N_1(p_0) = \{p_1, p_3\}$, $N_2(p_0) = \{p_1, p_3\}$ and $N_3(p_0) = \{p_1, p_2, p_3\}$. For later examples, we also need the following 2-distance neighborhoods $N_2(p_1) = \{p_2, p_3\}$ and $N_2(p_3) = \{p_2, p_4\}$. Now, we can compute the reachability distances of which we give some examples below.

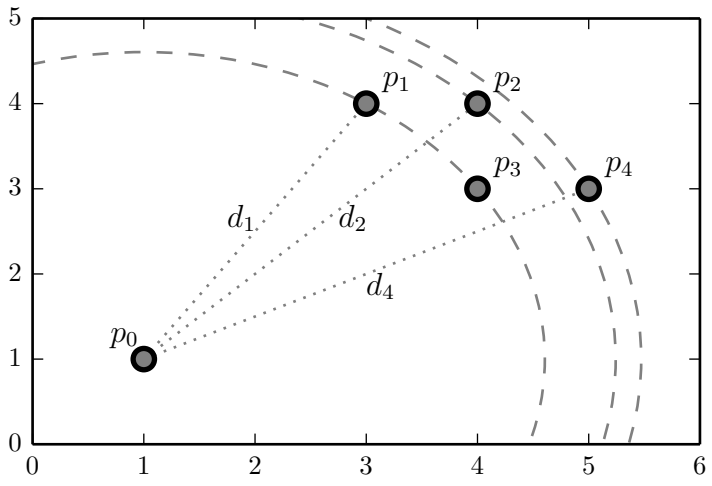


Figure 10.3: Plot of values used in the local outlier factor example.

- For $k = 2$: $\text{reach-dist}(p_0, p_1) = \max\{d_1, d(p_0, p_1)\} = 3.61$
- For $k = 3$: $\text{reach-dist}(p_0, p_1) = \max\{d_2, d(p_0, p_1)\} = 4.24$

For the actual computation, the values for k are fixed to a constant $MinPts$ which defines how many neighbor objects are used for the computation of the reachability distance. Thus, $MinPts$ defines the degree of smoothing to use for the computations. This value is first employed in the definition of the so-called local reachability density of an object p.

Definition 14 (Local reachability density)**.** Given an object p and a value $MinPts$, the local reachability density of p is given by

$$\text{lrd}_{MinPts}(p) = \left(\frac{\sum_{o \in N_{MinPts}(p)} \text{reach-dist}_{MinPts}(p, o)}{\sharp N_{MinPts}(p)} \right)^{-1}$$

This local reachability density is the inverse of the average of the reachability distances of all $MinPts$-nearest neighbors of p. Thus, it gives an estimate of how close the nearest neighbors of p are to their own nearest neighbors.

Definition 15 (Local outlier factor)**.** Given $MinPts$, the local outlier factor of an object p is defined as

$$\text{LOF}_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{\text{lrd}_{MinPts}(o)}{\text{lrd}_{MinPts}(p)}}{\sharp N_{MinPts}(p)}$$

Due to its definition, the local outlier factor provides a value quantifying the divergence of p's distance to its neighbors to the distances of p's neighbors to their own neighbors. If this divergence is high, p lies in region of low density and is isolated compared to its nearest neighbors. This identifies p as an outlier. If p is no outlier, the divergence is very low which leads to a LOF close to or even less than 1.

Example 5. In this example, we continue with the dataset D as given in Example 4. For all our considerations in this example, we set $MinPts = 2$. First, we determine the LOF of p_0 for which we need the local reachability densities of p_0 and also of its nearest neighbors p_1 and p_3.

$$\begin{aligned}
\mathrm{lrd}_{MinPts}(p_0) &= \left(\frac{\text{reach-dist}_2(p_0,p_1) + \text{reach-dist}_2(p_0,p_3)}{\sharp N_2(p_0)}\right)^{-1} \\
&= \left(\frac{3.61 + 3.61}{2}\right)^{-1} = 0.28 \\
\mathrm{lrd}_2(p_1) &= \left(\frac{\text{reach-dist}_2(p_1,p_2) + \text{reach-dist}_2(p_1,p_3)}{\sharp N_2(p_1)}\right)^{-1} \\
&= \left(\frac{\sqrt{2} + \sqrt{2}}{2}\right)^{-1} = 0.71 \\
\mathrm{lrd}_2(p_3) &= \left(\frac{\text{reach-dist}_2(p_3,p_2) + \text{reach-dist}_2(p_3,p_4)}{\sharp N_2(p_3)}\right)^{-1} \\
&= \left(\frac{1.0 + 1.0}{2}\right)^{-1} = 1.00
\end{aligned}$$

Thus, we get a LOF value for p_0 of

$$\begin{aligned}
\mathrm{LOF}_2(p_0) &= \frac{\frac{\mathrm{lrd}_2(p_1)+\mathrm{lrd}_2(p_3)}{\mathrm{lrd}_2(p_0)}}{\sharp N_2(p_0)} \\
&= \frac{\frac{0.71+1.00}{0.28}}{2} = 3.08
\end{aligned}$$

For comparison, we also compute the LOF for p_3 for which we know that $N_2(p_3) = \{p_2, p_4\}$. In addition to the already known local reachability densities, we need those for p_2 and p_4 based on $N_2(p_2) = \{p_1, p_3, p_4\}$ and $N_2(p_4) = \{p_2, p_3\}$. The resulting values are then used in the final computation of the LOF

value.

$$\begin{aligned}\mathrm{lrd}_2(p_2) &= \left(\frac{\text{reach-dist}_2(p_2,p_1) + \text{reach-dist}_2(p_2,p_3)}{\sharp N_2(p_2)}\right)^{-1} \\ &= \left(\frac{1.0+1.0}{2}\right)^{-1} = 1.00\end{aligned}$$

$$\begin{aligned}\mathrm{lrd}_2(p_4) &= \left(\frac{\text{reach-dist}_2(p_4,p_2) + \text{reach-dist}_2(p_4,p_3)}{\sharp N_2(p_4)}\right)^{-1} \\ &= \left(\frac{\sqrt{2}+\sqrt{2}}{2}\right)^{-1} = 0.71\end{aligned}$$

$$\begin{aligned}\mathrm{LOF}_2(p_3) &= \frac{\frac{\mathrm{lrd}_2(p_2)+\mathrm{lrd}_2(p_4)}{\mathrm{lrd}_2(p_3)}}{\sharp N_2(p_3)} \\ &= \frac{\frac{1.00+0.71}{1.00}}{2} = 0.85\end{aligned}$$

Given the dataset, p_0 would be intuitively considered as an outlier while p_3 is a normal value which belongs to a region with more other values. The results of the outlier detection by means of LOF are in accordance with this intuition since the score of p_0 is much higher than the one of p_3.

One weakness of the LOF approach is that there is no strict rule when to consider an object to be an outlier since the score is influenced by the characteristics of the full dataset. For example, in datasets which contain regions that greatly differ with respect to their densities, the computed LOF value will be higher even for normal objects than in datasets whose regions are of similar density. There are some approaches which try to compensate for this problem. The local outlier probability (LoOP) approach by Kriegel et al. [61] provides a value which gives the probability for an object to be an outlier instead of an unbounded score. However, this approach does not always simplify the identification of outliers considerably since high differences in the densities of regions still lead to great differences in the produced probabilities.

Chapter 11

Detecting Numerical Errors in Linked Data

By using object properties between instances it is possible to model a great amount of different information. Besides simple relations between two instances, e.g., a city belonging to a country, more complex relations can be stated by means of reification. However, object properties are limited to describe relations between instances. This prevents expressing large parts of important information which cannot be modeled between instances but only by assigning literal or textual data to an instance. Data properties as defined in RDF and OWL provide means to express such data. For instance, by using data properties it is possible to state the name of a person as in the following triple

`p:john` `foaf:name` "John Doe"

or state the population count of a city

`p:New_York` `p:population` "19,651,127"

In many datasets, data properties are at least as important as object properties. Depending on the context and the type of information contained in the dataset, they might be even more relevant than the relations between instances.

For our data cleaning approaches as presented before, data properties pose a special challenge. In principle, it is easily possible to transfer the basic learning algorithms to data properties. Domain and range restrictions could be learned as usual based on the datatypes instead of class memberships. According to the OWL 2 specifications, it is also possible to state disjointness between two data properties, thus, disallowing an instance and a specific data value to be connected by both data properties at the same time. In theory, a hierarchy of data properties could be also defined. All this could be supported by the property axiom learning algorithms as presented in Chapter 6 with minor modifications. This would not give satisfying results since the restrictions posed for the properties either work on the very coarse-grained level of datatypes or on the extremely fine grained level of

individual data values. Both makes it hard to determine generally valid property axioms which are at the same time restrictive enough to provide additional possibilities of detecting erroneous data. Thus, to find errors in the data values of Linked Data, a more in-depth analysis of the values by a specifically adapted approach is required as already suggested in Chapter 3.

The fact that great amounts of Linked Data are automatically generated and that these generation methods are error-prone, makes an automatized detection of errors in data values desirable. During the extraction, many factors can lead to the introduction of errors into the data. By just relying on manual data inspection, these errors are hardly recognizable due to the large amount of data produced during the extraction. For example, parsing errors that occur during the generation of Linked Data out of semi-structured data sources might lead to erroneous Linked Data values. A typical case which can cause such parsing problems are different usages of thousands delimiters. While people from English-speaking countries typically use commas for delimiting thousands blocks (e.g., $1,000$) and periods for splitting integer parts from the fractional parts of numbers, other European countries use a period for splitting the thousands groups (e.g., 1.000) and commas to specify the fractional part. Even knowledge about the expected usage of these delimiters cannot always prevent parsing errors since especially crowd-maintained data sources like Wikipedia cannot be guaranteed to be free from improper usages. Similar problems can occur when the data is entered according to a wrong base unit, e.g., when the expected unit for a value is kilometers but the number is provided in meters.

For detecting erroneous values in a dataset, the great amount of available data, although it hinders the manual inspection of the values, can also be advantageous for us. We can assume that most values are correct and can employ this for getting insights into the behavior of correct values. Many data errors lead to data values that have different characteristics compared to the correct values. For example, a wrongly interpreted thousands delimiter leads to values in a different order of magnitude which will most probably make this value suspicious when compared to correct values. Thus, we can use information about the expected behavior of the data to recognize erroneous data values.

As already explained in Section 10, outlier detection can help to detect values which deviate from the expected behavior so much that they are suspicious of being produced by a different mechanism. Considered in its entirety, the way of detecting errors in Linked Data as we described it in the previous parts of this work can be also considered to perform an outlier detection. In the ontology learning step, we generate an ontology that adheres to the majority of the data contained in the dataset. Small parts of the data that do not follow a certain axiom do not prevent this axiom from getting learned. Instead, in the later error detection step, those parts of the data are marked as potentially erroneous. Thus, when taking the definition by Chandola et al. [25] that outlier detection is “finding patterns in data that do not conform to expected behavior”, the learning step is the discovery of the expected behavior for the complete dataset and the subsequent detection of data violating

the schema axioms can be considered the detection of not conforming behavior.

In this chapter, we propose and evaluate a method for detecting wrong numerical values in Linked Data using outlier detection approaches as originally published as a joint work with Heiko Paulheim, Volha Bryl, Johanna Völker and Christian Bizer [38]. This approach is based on the idea that values which are marked as outliers during an outlier detection run are more likely wrong than those not detected as outliers. First, we determine outliers regarding a single data repository, e.g., on all values assigned by means of the `population` property. For this purpose, we present a way of discovering data subpopulations induced by classes and properties and apply outlier detection to these subpopulations. For example, on the full dataset, the populations of continents would be outliers for the `population` property values since their population values are larger than the predominant population values of cities or countries by several orders of magnitude.

This first step leaves us with a set of values which are suspicious since they considerably differ from the other values for a specific property. However, such outliers can also occur naturally, e.g., there are some cities which have a much higher population count than cities usually have. Such cases are called *natural outliers*. To prevent us from detecting such values as erroneous, we introduce a second step. We exploit the `owl:sameAs` links of the instances for collecting values for the same property from other repositories. This facet of our approach uses one of the core concepts of *Linked* Data rarely used in other works: the links between repositories. The additional values retrieved from other repositories are then used to cross-check the original value. If the additional values agree with the original one, we most probably discovered a natural outlier and thus should not consider it as erroneous. In cases where the original value also is an outlier regarding the other values, we found additional evidence for a data error.

In the following, we first give an overview on similar work on finding errors in Linked Data. Afterwards, in Section 11.2, we describe our approach of using outlier detection for detecting erroneous numerical values. This approach is evaluated with experiments on the DBpedia dataset in Section 11.3. In Section 11.4, we conclude this chapter by analyzing the findings and give possible directions for further work.

11.1 Related Work

Compared to the detection of errors on the logical level, the detection of errors in the values of data properties is a relatively new field of research. The TripleCheckMate tool by Kontokostas et al. [60] provides a framework for performing crowdsourced evaluations of Linked Data. Given a data source, it selects triples and presents those to users for validation of correctness. Triples recognized as being erroneous can be marked and the user is able to classify the specific type of error according to an ontology of data quality dimensions and categories such as accuracy problems or consistency of representation, both being relevant for errors in

data properties.

The possibilities of crowdsourcing the quality assessment for Linked Data based on TripleCheckMate were evaluated by Acosta et al. [2]. In this evaluation, they first organized a challenge for Linked Data experts to find and categorize errors in the DBpedia dataset by using TripleCheckMate. Afterwards, discovered errors of three selected categories were given to layman workers on the Amazon Mechanical Turk platform to let them verify the expert assessment. To not overwhelm the workers, Acosta et al. developed appropriate representations of the tasks that hide the actual Linked Data. Both the results of the experts and those of the laymen were evaluated against a gold standard for the relevant triples created by the authors. The results of this evaluation depended on the considered type of error. For wrongly extracted or missing values and for detecting wrong links, the crowd workers showed a better performance than the experts. The experts performed better for finding wrong datatypes assigned to data values. However, as the authors also remark, the better performance of the layman workers might be caused by the better evaluation interface. Nevertheless, the evaluation showed that crowdsourcing to experts or laymen as a valid way of evaluating Linked Data quality as long as the questions are preprocessed for the particular audience and the task does not depend on specific knowledge as it is for the detection of wrong datatypes. These results are to some degree similar to those that we got when evaluating ontology axioms by means of crowdsourcing as presented in Chapter 6.

The previously described method relies on humans for assessing the actual correctness of the data. Though this leads to high precision regarding the detected errors, it also limits the scalability of the approach. This limitation is lifted in the approach presented by Kontokostas et al. [59] that we already mentioned in Chapter 9. Based on the idea of unit tests from software development, they not only use schema axioms to check for errors like wrong linking of instances but also define test cases on the level of data properties. Besides testing for correct datatypes used with a specific property, RDFUnit allows comparing two literal values assigned to the same resource or checking whether a single value is contained in a given value range. This kind of test case can be used for detecting values which are only suspicious in certain combinations, e.g., a death date that is earlier than the birth data of the instance or detecting abnormal values like exceptional high heights of humans. Compared to the schema-level test cases, the creation of datatype-based tests needs more manual effort since they cannot be directly deduced from already existing schema information. For reducing this initial effort, the authors propose to compile test cases for commonly used vocabularies in a publicly available library so that all users of the vocabulary can directly apply the corresponding test cases. The manual creation of test cases helps to also catch semantic relations between properties and obviously allows to include human knowledge into the error detection process. However, depending on the degree of vocabulary reuse, the manual effort for setting up appropriate test cases might be high. Even the patterns contained in the library should be manually checked for applicability since the actual semantics of usage might differ between different datasets though using the same

vocabulary. In contrast to this work, our approach presented later-on in this chapter tries to reduce the human involvement in the first phase to a minimum. This allows the direct application of the error detection to unknown datasets. Though human experts are still required for actually assessing the validity of detected possible errors, our process can help to reduce the manual effort. This holds even more since in many cases it might still be required to manually check the errors detected by RDFUnit test cases. Furthermore, a combination of both methods might be interesting. Using a more automatized approach like ours as first step could help to get insights into the dataset that later can be used to formulate more appropriate test cases.

Instead of relying on manually crafted rules or full manual evaluation, Wienand and Paulheim [99] introduced a method for automatically detecting erroneous values in numerical Linked Data. They use outlier detection to find erroneous numerical values assigned to instances. The authors evaluated different settings regarding preprocessing the dataset and applying the actual outlier detection. During the preprocessing, the values got grouped by single instance types or clustered by type vectors for reducing the number of wrongly recognized erroneous values. Furthermore, different outlier detection methods were evaluated. For example, they evaluated the interquartile range (IQR) method which considers values not lying in a specified value range around the median as outliers. Another evaluated outlier detection method relied on the so-called Kernel Density Estimation (KDE) that approximates a probability distribution and uses it for determining whether a value is an outlier or not. In a pre-study, the clustering showed better results than the single type preprocessing but suffered from very high runtimes. Furthermore, regarding the outlier detection approach, the KDE-based approach showed a slightly higher precision, however, it also had runtime issues. Thus, as the most well-balanced setting, the single type clustering using the IQR method was considered by Wienand and Paulheim for the final evaluation. This setting showed a precision of around 81% up to 89% depending on which values were included into the evaluation. Based on the results, the authors performed an extensive evaluation of the different errors types detected by their method. In the course of this evaluation, they identified eleven structural errors in the DBpedia dataset including wrong parsing of single numerical values and unit conversion problems. Our work presented in the following builds on the same foundations as Wienand and Paulheim's work. However, we extend the approach in two important directions. First, we introduce an alternative method of finding relevant subpopulations of the full dataset which combines high performance with the possibility to detect more complexly defined subpopulations. This is particularly important since runtime issues are the most common issues leading to not using the best-performing approaches in the original paper. Secondly, we propose a post-processing step that helps to reduce the number of natural outliers being detected as erroneous values.

Another approach towards more automatic support for detecting errors in Linked Data is proposed by Cherix et al. [26] with their CROCUS tool. Their idea is to represent each instance numerically and then apply the DBSCAN clustering algorithm

for finding instances that are outliers regarding this representation. For the further processing, the authors first retrieve the Symmetric Concise Bounded Description, i.e., the graph reachable by traversing a given number of edges starting from the actual instance. Based on these triples, the authors apply different metrics to produce the numerical representations. They count the occurrences of each property, the count of each property with a specific range and, as a third metric, converting the properties' values to numeric ones by directly taking numerical values or determining the length of string values. The application of the DBSCAN algorithm clusters the values. If a cluster has less than the specified number of values, all its values are considered to be outliers which are then handed to humans for further evaluation. Cherix et al. evaluated on the artificially generate LUBM dataset extended with erroneous values and on a subset of DBpedia only containing information about German universities. In both cases, their approach performed well with respect to F-measure. In contrast to our approach presented later, their approach does not consider an automatic restriction to reasonable subsets of instances for improving the detection result. Instead, they manually limited their evaluation to instances of the class `University`. Furthermore, they also do not address the problem of natural outliers further which is one of our special focuses.

11.2 Approach

In the following, we describe our overall approach of detecting wrong values in a Linked Data dataset. First, we shortly describe how we determine the properties to check for wrong numerical values before we present the actual process of outlier detection. As already mentioned above, applying outlier detection to the full set of data values might not result in good results since the typical pattern for these values might depend on the type of the corresponding object. We explain this aspect further and introduce our way of determining subsets of data on which we apply the outlier detection. Finally, we describe the actual detection of erroneous values from the outlier detection results including the cross-checking using data from alternative sources.

11.2.1 Dataset Inspection

In our approach, we assume no prior knowledge about the dataset. Thus, the first application step is to gather additional information about it. These steps are most easily performed when the data is provided by a SPARQL endpoint. However, it is also possible to adapt the methods to other data provisioning methods such as RDF data dumps or other query languages such as the Metaweb Query Language (MQL) used in the Freebase dataset.

During the dataset inspection, we determine the number of instances contained in the repository as well as the names of the properties that are used in the dataset. For debugging, we are only interested in data properties used with numerical val-

ues. Because we cannot rely on having an OWL vocabulary which divides the properties into object and data properties, we determine how often each property is used with a numerical value in the object position.[1] This is done using the SPARQL query

```
SELECT ?p, COUNT(DISTINCT ?o) AS ?cnt
WHERE {?s ?p ?o. FILTER (isNumeric(?o))}
GROUP BY ?p
```

Based on these usage information, we can filter the properties to which we apply the error detection approach. For instance, we can filter out all properties that are only used with a single distinct numerical value since in this case no proper error detection is possible anyways.

Finally, this process results in a set of properties which qualify for the application of our error detection approach in the next steps.

11.2.2 Generation of Possible Constraints

In the further workflow, we process each of the considered properties separately. It is important to note that the detection of wrong values is always done on the level of pairs of an instance and a specific value assigned to this instance by means of the property. This is required since a single instance might have several different values assigned using the same property. For example, some cities might have multiple ZIP codes which are assigned to its instance by a property like `hasZipCode`.

Considering all instance value pairs at once might *mask* potentially erroneous values. For example, consider the property `population` which assigns the population count to a populated place like a village or a country. When we compare the population counts of villages and countries, it is obvious that the biggest part of countries has a much higher population count than villages. Thus, in a dataset containing the population counts for both countries and villages, we can expect two clusters of values. Figure 11.1 shows an example histogram of such a dataset. The white bars mark values from villages while the grey bars represent those of countries. Without taking the type information into account, the grey bars with population counts of around $20,000$ do not raise any suspicion since there are many village instances in this range. The suspicious country population counts are masked by the village population counts. However, when only considering the population counts of countries, these values are outliers because they deviate greatly from most other countries' values.

A similar problem could occur in the already mentioned case of having population counts of continents and countries in the same dataset. Since the number of continents is very small compared the number of countries, the considerably higher population counts of the continents would seem to be outliers considering the whole dataset.

[1] In this context, we treat `xsd:int` and `xsd:float` as well as their subtypes as numerical values.

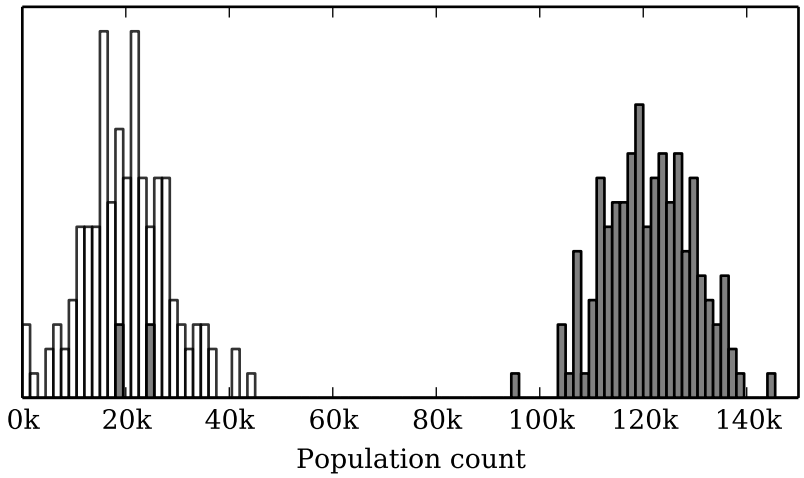


Figure 11.1: Histogram of population counts for villages (white) and countries (grey). Some population counts of countries are masked by the population counts of villages.

To handle these types of problems, we do not consider the whole dataset formed by all instance-value pairs connected by a property but focus on subsets of the full dataset. For generating these subsets, we take advantage of the additional information assigned to instances in Linked Data datasets like the given types for the instances and their properties. In the following, we define the different ways of constraining the original dataset to produce the smaller, constrained subsets. A constrained subset only retains the instance-value pairs for which the instance fulfills the applied constraint.

- *Class constraints:* A class constraint on class C applied to an instance set limits it to instances that belong to this class.
- *Property constraints:* A property constraint p limits the instances to those connected to an arbitrary object (instance or data value) by means of p.
- *Property value constraints:* A property value constraint is defined by a property p and a value v which can be either an instance or a data value. It limits the instances to those which are connected to a value v by means of p.

Class constraints as also applied by [99] are the most obvious way of utilizing the class structure already contained in the dataset. They allow capturing the masking for the `population` property described before.

This way of constraining the dataset reaches its limits for datasets that do not contain a comprehensive class structure. As also shown by Paulheim and Bizer [77], it is possible to deduce type statements for instances using informa-

tion about their usage of properties. The two property-based constraint types employ this fact and enable us to consider property information for compensating incomplete class structures. For example, given a class `Vehicle` and a property `maximumAltitude`, this property can compensate for a missing class assertion to a class `Aircraft` and thus allow to detect, e.g., too high `weight` values for instances that could be otherwise masked by other `Vehicle` instances such as ships. The choice which properties to use as constraints is based on the property's number of usages in the current instance set.

If most instances show the same usage patterns regarding properties, even property constraints are not able to provide sufficient information for differentiating different types of instances. In this case, property value constraints might allow further division of the full instance value set. In contrast to the property constraints, which only filter instances based on whether they are used with a specific property or not, property value constraints restrict based on the usage of a specific property with a specific property value. For instance, consider an error detection run regarding the property `averageTemperature` on a set of city instances. Since the average temperature of a city heavily depends on its location, it is hard to recognize any suspicious temperature values by considering all cities at once. Even if the city instances are connected to their respective countries by means of a `country` property, using property constraints would not give any benefit since all cities are subject of a `country` property. Nevertheless, by only looking at cities which are connected to a specific country, like all cities in the United Arabian Emirates (UAE), wrong temperature values can be detected. Because the average temperature in the UAE is higher than the temperature in most other countries, too low `averageTemperature` values assigned to a city in the UAE could be masked by cities from other countries. Limiting the detection to cities from the UAE, the low average temperature is suspicious and thus detectable as being erroneous.

Both property-based constraint types share the problem that they introduce a high number of constraints since the number of properties is typically higher than the number of classes contained in a dataset. This leads to a higher computational effort for choosing the constraints when determining the usage counts for the single properties. The effort is even higher for property-value constraints since in addition to the usage counts for the properties, the actual values used with the properties have to be inspected leading to a potentially much higher number of constraints. Thus, it is advisable to use property-based constraints only when the usage of class constraints does not allow sufficient restrictions of the instances.

11.2.3 Finding Subpopulations

Applying outlier detection to all of the potentially many subpopulations, which can be defined on a dataset, is impractical especially because the runtime of outlier detection algorithms heavily depends on the number of values they are applied on. To reduce the numbers of outlier detection runs, we introduce an intermediate step in which we determine the subpopulations to apply outlier detection on. During

this exploration step, we examine the subpopulations generated by restricting the instance value set using a constraint or a set of constraints.[2] The exploration is organized in a lattice-like structure as shown in Fig. 11.2. This lattice structure is similar to the one used by Melo et al. [71]. Each node in the lattice is assigned a set

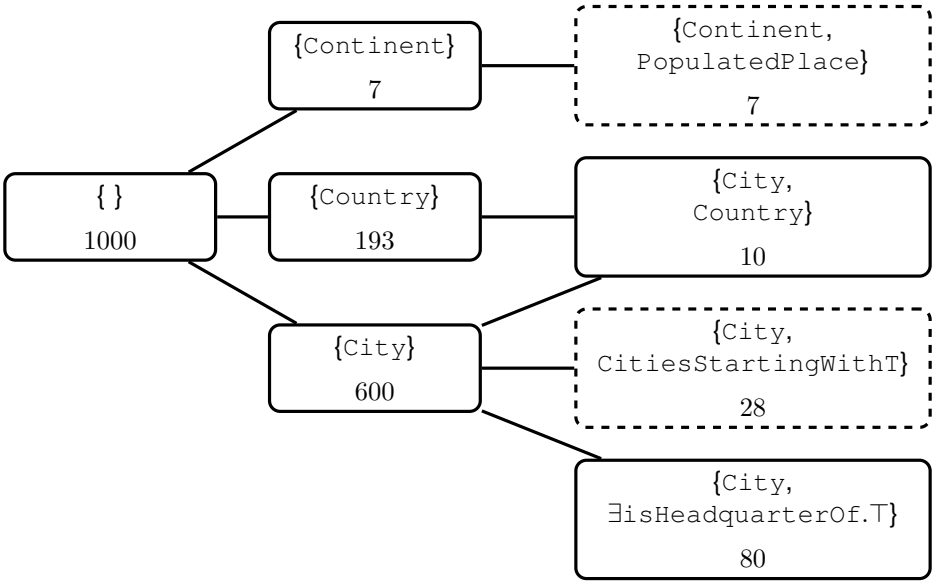


Figure 11.2: Example for subpopulation lattice for property `population`. Numbers in the lower part of each node give the number of instances fulfilling the constraint set. Dashed nodes get pruned, the lower one for too low KL divergence, the upper one for not reducing the instance set further.

of constraints which determines the instances considered at this node. Two nodes are connected by an edge if the child node can be produced from the parent node by adding exactly one single constraint to the parent's constraint set. Furthermore, for each node we determine the number of instance value pairs adhering to the assigned constraints. The root node has the empty constraint set assigned and thus represents all instances and corresponding values of the currently considered property. For this set of instances, we compute a histogram which represents the distribution of values in the subpopulation. Starting with the root node, our approach manages a queue of all not yet extended nodes and thus extends the lattice in a breadth-first-manner.

When processing a node from this queue, we create its child nodes, each having an additional constraint compared to the parent node. The additional constraints are those from the set of possible constraints which are not yet used in the parent node. If a node for the resulting set of constraints already exists in the lattice, we do not consider the new node further. Otherwise, we determine the instances

[2] In the following, we use the term "constraint" to refer to both a single constraint or a set of constraints when this does not lead to confusion.

that adhere to this new set of constraints and compute the histogram of the value distribution. Based on this value distribution, we enforce a set of pruning criteria to keep the search space clean. This allows our approach to be independent from any further knowledge about the constraints respectively the classes and properties used in the constraints. In particular, we do not depend on a hierarchy specified for the classes and properties which makes our approach applicable to virtually all Linked Data datasets regardless of their schema's existence and expressivity. As a first pruning criterion, we consider the number of instances which adhere to the set of constraints. If only a low number or maybe no instances at all are valid for the current node, we prune the node and do not consider it further since the set of constraints is too specific and outlier approaches would not be able to detect any viable pattern.[3] Another criterion is the instance reduction ratio, i.e., the change ration in the number of instances between the parent node and the new node. If the additional constraint leads to a reduction of less than 1%, we prune the new node. For instance, this case could occur when we add a class constraint on `PopulatedPlace` to a constraint set which was previously also constrained on `Country`.

Finally, we explicitly consider the change in the distribution of values between the parent and the child node. For this purpose, we use the value histograms for the subpopulation of the parent node and the subpopulation of the parent node and compute the Kullback-Leibler (KL) divergence between the value distributions represented by both histograms. The KL divergence is a measure for the difference between two probability distributions introduced by Kullback and Leibler [62]. In general, it is defined for two discrete probability distributions P and Q by

$$D_{KL}(P||Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}$$

where i iterates over all possible values in the distributions. If the divergence computed between both histograms is lower than a given threshold value, we assume the additional constraint to be independent from the previously applied constraints given the currently processed property, i.e., the actual distribution did not change but only the number of instances. In these cases, an outlier detection run on the new node would not yield additional insights and thus we prune nodes which do not achieve a sufficiently high divergence from their parent nodes. For example, this pruning could happen when adding a class constraint on a class `NamesStartingWithT` to a constraint set for a property representing the population count. Assuming that the population counts of cities whose names start with a T are distributed equivalently to those of all cities, the distribution and consequently the KL divergence would not change considerably. Thus, the further restriction cannot be expected to reduce the masking of erroneous values and pruning the node prevents an additional outlier detection run which would not lead to additional discoveries.

[3] In our experiments, a value of 5 was used.

Each additional constraint reduces the number of instances further and hence the sampling error gets higher with every additional constraint. We try to reduce this effect by two additional modifications. First, we apply Laplace smoothing to the value histograms by assuming one additional value occurrence for each bin in the histograms. Secondly, we normalize the computation of KL divergence by including the change ratio between the parent node and the child node. This leads to the following modified formula used for computing the divergence between the two nodes $parent$ and $child$ contained in the lattice. The number of instances for a node n is represented by $|n|$ while h_{parent} as well as h_{child} are the histograms representing the respective value distribution which each have B bins.

$$\text{divergence}(parent, child) = \left| \frac{|child|}{|parent|} \cdot \sum_{i=1}^{B} \ln\left(\frac{h_{\text{parent}}(i)}{h_{\text{child}}(i)}\right) h_{\text{parent}}(i) \right| \tag{11.1}$$

Even after pruning, a large number of potential subpopulations to further expand with additional constraints and to apply outlier detection on might exist. Hence, we prioritize nodes that have a higher divergence to their parents in later expansion steps in cases where too many nodes would have to be expanded otherwise. This is in line with our supposition that a higher divergence shows a more important change in the distribution of values and thus is more interesting for the overall process. If a node is not pruned during the expansion of its parent node, it is added to the expansion queue and processed in a later step of our breadth-first processing.

11.2.4 Outlier Detection and Outlier Scores

After the lattice has been determined, we perform outlier detection on all unpruned nodes of the lattice and store the resulting outlier scores together with the set of constraints which led to the corresponding instance set. We use the Local Outlier Factor method to determine outlier scores as described in Section 10.2. Compared to other approaches of finding outliers, this method allows us to be independent from any specific probability distribution of the values and thus does not limit our approach to certain types of numerical values. Nevertheless, our overall approach is not restricted to this choice of an outlier detection method since we do not rely on any specifics of LOF apart from having a scoring instead of a binary classification into outliers and non-outliers.

As soon as the outlier detection run is completed on the property, we have a list of instance value combinations with a set of pairs, consisting of a constraint set and an outlier score. Thus, the scores show the degrees of outlierness of the different instance value combinations regarding various subpopulations. To get a single assessment of a value's correctness, we can apply different weighting approaches to this list of scores. One possibility is to use the highest outlier score for each instance. However, during our experiments, this weighting scheme showed to often lead to wrong detections since in many cases an outlier score was chosen which was

generated before reaching a reasonably limited subpopulation. Also weighting approaches based on determining the average or the average outlier score weighted by the number of constraints used for generating corresponding subpopulation showed similar problems.

During our pre-studies, using a possibly existing hierarchy for the classes and properties turned out to be more promising. By determining the number of superclasses or superproperties for each class and property, respectively, in the constraint set, we can get an assessment for its specificity. For each instance value pair, we use the outlier score achieved for the subpopulation restricted by the most specific constraint set. This approach favors more specific classes like `City` over classes higher in the hierarchy like `PopulatedPlace`. To determine the specificness of an entity, we use property paths as introduced by SPARQL 1.1 like in the following query for a class by its IRI `CLS`

```
SELECT COUNT(DISTINCT ?i) AS ?cnt
WHERE {
        <CLS> rdfs:subClassOf+ ?i
}
```

The more direct and indirect superclasses a specific class has, the higher we assume its specificity. In cases where no hierarchy is defined, this leads to all classes having the same specificity, making it still possible to use a different weighting scheme as fallback.

Finally, the instance value pairs are ranked according to the outlier scores resulting from the application of the weighting scheme.

11.2.5 Cross-checking for Natural Outliers

Given the ranked pairs, we have a list of instances and values ordered by their deviation from typical behavior of other instance value pairs. However, as we described before, outliers do not necessarily point to erroneous values but can be also caused by values naturally showing deviation from the typical behavior. An example for such an outlier is the island Honshu which is one of the main islands of Japan. Being the main island of Japan, a large part of Japan's population is living there leading to a population count of $103,000,000$.[4] The three additional main islands each have a population of around $5,000,000$ to $14,000,000$. However, Japan is an island nation and in total consists of $6,852$ islands, the most of which only have have population counts of several hundred to several thousands. Thus, the population of an extremely high fraction of Japan's islands is several magnitudes smaller than the population of the main islands. Given the high number of lowly populated islands, an outlier detection approach applied to the island data would assign a high outlier score to the main islands and, in particular, to Honshu. Hence, when considering the raw outlier scores at this stage potentially leads to many natural outliers falsely recognized as erroneous.

[4] According to `http://en.wikipedia.org/w/index.php?title=Honshu&oldid=618654743`

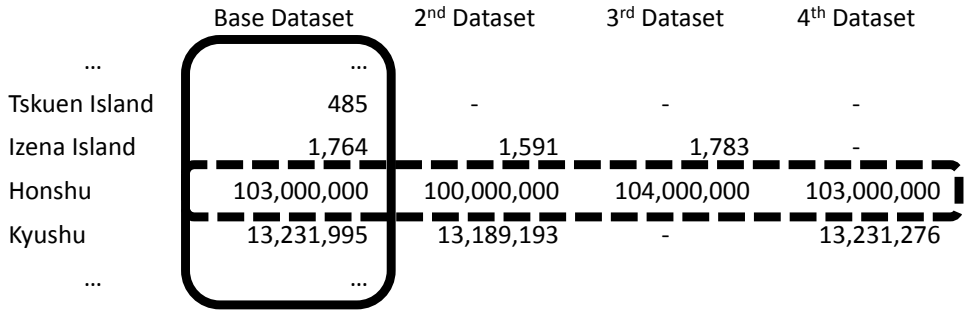


Figure 11.3: Using two independent outlier approaches for the DBpedia property `populationTotal` and the instance "Honshu" to improve the detection result. Only considering the base dataset (vertical), the actually correct value is detected as an outlier. The detection run on the values from different sources (horizontal) confirms the value and thus prevents to mark the value as a wrong value.

To prevent this wrong detection, we introduce an additional cross-checking step. This step exploits one of the core features of Linked Data: the possibility to state links between the instances in different repositories. This is often used to state equivalence of instances across different datasets by means of `owl:sameAs` statements. Since both the Linked Data and the Semantic Web community encourage the reuse and interlinking of schema vocabulary, these statements enable us to retrieve additional property values for the same instance. Even in cases where additional datasets do not use the same vocabulary as the primary dataset, ontology matching techniques as summarized by Euzenat and Shvaiko [35] can be used to find properties equivalent to the currently considered property across the datasets. Based on this idea, we follow the `owl:sameAs` links for the different instances and gather additional values for the current property. Assuming that the original property value for a certain instance is correct in the primary dataset and also in all supplemental datasets, all values should be the same. Hence, we consider all instance value pairs as natural outliers for which all datasets agree in the actual value. Obviously, an exact match between the gathered values is hard to achieve for many properties which fluctuate slightly based on the exact point of time they were determined and also based on the source of the information. Thus, filtering based on an exact match of the values is not necessarily sufficient to filter out natural outliers. To handle this problem, we perform another outlier detection run. Instead of checking whether the considered value is an outlier against the set of values assigned to other instances in the primary dataset, we check whether the value is an outlier with respect to the gathered values for the same instance (or those assigned to be equivalent) in the supplemental datasets. This introduces a second dimension of outlier detection as illustrated in Figure 11.3.

Since we only intent to recognize slight deviations, a full-blown outlier detection approach is not required in this context. Rather, the deviations from rounding, smaller dimension errors and imprecisions can be expected to be normally dis-

tributed around the correct value. Thus, we perform this outlier detection run using the normal distribution-based approach described in Section 10.1. By adapting the size of the confidence interval using the constant c, we are able to influence the sensitivity of this cross-checking step. If a value from the base dataset is not detected to be an outlier regarding the value from the supplemental dataset, we consider it to be a natural outlier and remove it from the final list of erroneous value candidates. Otherwise, we leave the value in as being a potential outlier.

It is evident, that this cross-checking step heavily depends on the availability of additional data for the same instance in other datasets and only works as long as this data is discoverable by means of equivalence links between the instances. For the special case of the English DBpedia dataset and the corresponding versions in other languages, it has be shown by Bryl and Bizer [21] that the number of instances described in multiple datasets is relatively low. However, even if the additional data is sparse, the overall approach still profits from it. Our main interest is finding additional information about natural outliers so that we can filter those out from the list of potentially erroneous values. Filtering out values which are no natural outliers or finding additional evidence that a value is an outlier is not the first priority. We often can assume natural outlier objects to be more interesting to humans than arbitrary "non-special" objects. Thus, we can reasonably expect the natural outlier objects to be described in more different datasets than the non-natural outlying objects which is to the best advantage of our cross-checking approach.

11.3 Experiments

We carried out two experiments to assess the performance of our approach. First, we applied the full approach to selected properties of the DBpedia dataset and manually evaluated randomly chosen samples of the regarding their correctness. Afterwards, we performed a second experiment particularly for evaluating the availability of additional data for the cross-checking step based on the Linked Data dataset NELL.

11.3.1 Evaluation of Full Approach

To test the proposed approach, we performed experiments on the DBpedia 3.9 datasets. As described in Section 2.2.3, DBpedia is extracted from the data provided by the Wikipedia project. In this process, each sufficiently large language version of Wikipedia is used to create a separate DBpedia dataset. This results in a total number of 119 language version of DBpedia available in version 3.9 which in combination contain 2.46 billion triples describing 12.6 million unique things. The largest share of these is contributed by the English language version which describes about 4 million instances using 470 million triples. The single language versions are connected to each other by means of `owl:sameAs` links relating the

instances that describe the same thing in the different languages. These links are also called inter-language links in the DBpedia terminology.

The division into language editions and the availability of `owl:sameAs` links between the single instances provides a good scenario for testing our error detection approach. We implemented the approach as described in Section 11.2. For the implementation of the actual outlier detection by means of the Local Outlier Factor method, we resorted to the implementation made available for the RapidMiner data mining toolkit[5] by the RapidMiner Extension for Anomaly Detection.[6] The smoothing parameter k was set to 10 or the number of values in the currently considered subpopulation if it contained less than 10 instance value pairs. During the cross-checking step, we set the parameter $c = 2$.

For the number of bins used in the value histograms of the lattice, the usage of 100 bins turned out to be a good compromise between runtime and precision of the pruning approach. Higher number of bins lead to KL divergence values closer to the actual distribution of the values due to the histogram's higher resolution. At the same time, higher values increase the runtime of the KL divergence computation itself and thus contradict to some degree the runtime saved later-on in the outlier detection step. Values slightly higher than 100 increased the runtime without leading to adequate improvement in the runtime of the actual outlier detection and without clear increase in the final error detection.

The DBpedia instances are not only categorized by classes of the DBpedia ontology but also according to classes of the YAGO ontology.[7] In contrast to the DBpedia classes, the YAGO classes are partly based on Wikipedia categories and thus are much more fine-grained. For example, some instances are assigned to the YAGO class `CitiesAndTownsInAbruzzo` while only being assigned to the class `City` in the DBpedia ontology due to the lack of more detailed class structure. Thus, using the YAGO class for generating subpopulations, we are able to achieve a fine-grainedness similar to the one provided by property value constraints by just relying on class constraints.

For the cross-checking step, we make use of the inter-language links from English DBpedia instances to those in other languages editions. Notably, the overlap between the language editions is not high. In DBpedia 3.9, out of 2.7 million instances in the 17 most populated DBpedia ontology classes, 60% are described only in one language (predominantly English) and only 23% are described in three or more languages. Furthermore, we only considered those 24 language editions whose infobox properties were manually mapped to the DBpedia ontology by the DBpedia community. In these datasets, the same property URIs are used across the different languages, e.g., the DBpedia ontology property `populationTotal` is used for the population property of a populated place in German or French editions even if the original Wikipedia infoboxes use language-specific attribute names.

[5] `http://rapidminer.com`

[6] `https://code.google.com/p/rapidminer-anomalydetection/`

[7] `http://www.mpi-inf.mpg.de/yago`

To assess the performance of our approach for detecting erroneous values, we evaluated its performance on three DBpedia ontology properties: `height`, `elevation` and `populationTotal`. For each of these properties, we performed a run of the error detection approach and then randomly sampled 100 instance value pairs from the final ranked list where we introduced a bias towards higher ranked pairs. To create this bias, we chose a pair with a probability proportional to its outlier score, i.e., the probability of choosing a pair with a score of 2 was twice as high as choosing a pair with score 1. We performed the sampling process based on the ranked but not yet cross-checked list so that we had the possibility to assess both the performance with and without cross-checking. The resulting pairs were independently reviewed by three human annotators regarding the correctness of the values. For determining the correctness of a value, a typical process of the annotators was to first have a look at the current Wikipedia page describing the instance. Additionally, the Wikipedia page in its version as of the time of the extraction run was inspected. Using these two sources, it was possible in most cases to recognize errors in the values which stemmed from parsing errors or vandalism. If these inspections did not yet lead to the detection of an error, the most promising non-English Wikipedia articles about the instance were consulted, e.g., the article in the language most related to the instance. Finally, cited external sources were consulted or the annotators tried to find reliable information on the Web using search engines. If no proof for an error in the data was found, the instance-value combination was marked as correct, otherwise as wrong.

We computed the inter-annotator agreement (IAA) between the three annotators on the evaluated lists by means of Fleiss' kappa. The results of the IAA analysis are shown in Table 11.1. Though being chance-corrected, these values show a very high agreement for all three properties. We conducted an analysis of the few disagreements which showed that the major part of the disagreements was caused by an annotator not finding relevant external information to assess the correctness of the value. The table also provides the numbers of correct instance value pairs for each property. It is important to note that these values do not provide an unbiased insight into the correctness of DBpedia but are overstating its incorrectness due to the way we sampled the evaluated examples.

Table 11.1: Inter-annotator agreement observed for property samples and number of correct instance-value combinations according to majority of annotators.

	`elevation`	`height`	`populationTotal`
Observed agreement	0.987	0.960	0.960
Fleiss' κ	0.968	0.916	0.917
# correct	69	60	57

Based on the results of this manual evaluation, we plotted the distribution of the wrong instance-value combinations and the actual value distribution not only over the sampled values but over all values in the dataset. The resulting plots are

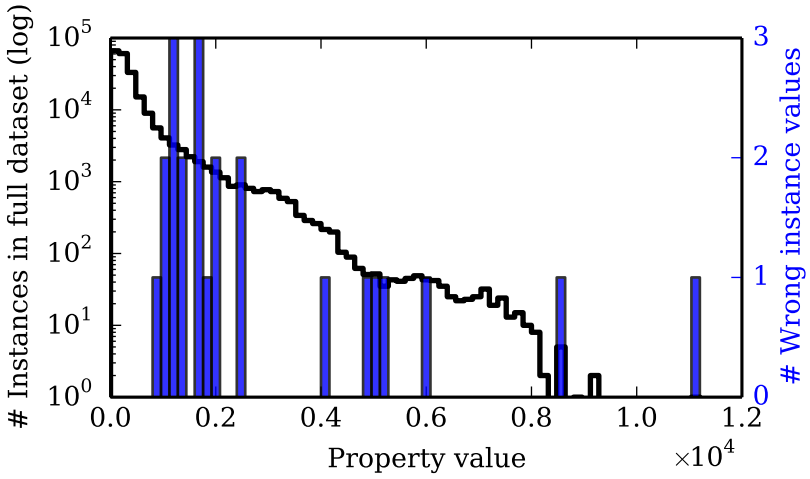


Figure 11.4: Distribution of all values in the instance value set for property `elevation` in log-scale with values of erroneous values discovered during the manual evaluation marked. Property value and instance count scale is restricted to the given ranges.

shown in Figures 11.4, 11.5 and 11.6.

These plots allow us a further analysis of the erroneous values. For example, in Figure 11.5, we recognize two accumulations of errors. The first one is located at the lower bound of the value range. This one is caused by errors for entities of the class `Person` using the wrong unit, e.g., instead of a correct value 1.98 m a value of 1.98 cm was assigned. In addition to these, this spike also contains errors that are not directly recognizable as errors since they fit the typical height of people. The second spike is located around a value of 200 and also stems from the problem of using wrong units, this time using meter instead of centimeter like 198 m instead of 198 cm. In particular, this finding stresses the need for performing the outlier detection on subpopulations instead of on the full datasets since values close to 200 in the full dataset not necessarily point to data errors. For other instance types like buildings, a height value of 200 is totally possible and valid. The other two properties show their erroneous values to be almost homogeneously distributed over the value range and to be not only corner cases in the given ranges. Thus, these outliers would not be detectable without considering subpopulations.

For actually evaluating the error detection, we determined its performance on the manually annotated value lists before and after cross-checking. These evaluation results were compared to two baseline approaches both also generating scores quantifying the assessment how erroneous they are. The first baseline approach, which we identify by “Baseline”, is computed by determining the median of all values and then computing the absolute difference between this median and the current instance’s value. The resulting value was used as a score regarding how wrong

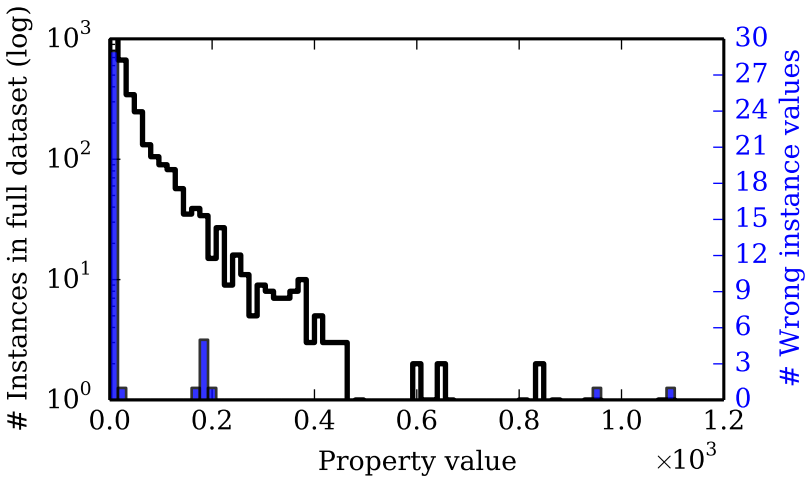


Figure 11.5: Distribution of all values in the instance value set for property `height` in log-scale with values of erroneous values discovered during the manual evaluation marked. Property value and instance count scale is restricted to the given ranges.

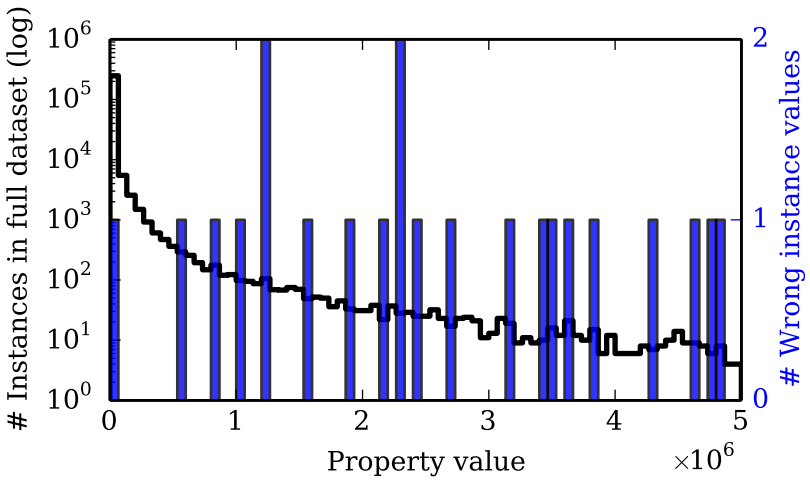


Figure 11.6: Distribution of all values in the instance value set for property `populationTotal` in log-scale with values of erroneous values discovered during the manual evaluation marked. Property value and instance count scale is restricted to the given ranges.

the value is. The second baseline, referred to as "Multi-lingual baseline" in the following, uses the multi-lingual data that is also employed for the cross-checking step. For getting a score for a value v, we compute the expression $|v - \mu|/\sigma$ where μ is the mean of the non-English values and σ their standard deviation. This is similar to the normal distribution-based outlier detection approach. Thus, assuming a normal distribution, 95% of all values should have a score less or equal to 2. In cases, where the multi-lingual data only provides one or even zero non-English value, we assign a score of 2 directly. This fall-back score models that values for which we do not find any information are more likely to be erroneous than values that got validated by similar values from other language's values.

Using the scores produced by all four approaches, we ranked the instance value samples according to their scores and starting with the highest. This mimics the usage of the error detection as an information retrieval problem where it provides a human with a ranked list of possibly wrong values based on which further manual checking can be performed. By comparing the manually created gold standard against the ranked lists, we determined the ranking of the erroneous values in the lists. The results of this evaluation are given as receiver operating characteristic (ROC) curves for each property in Figures 11.7, 11.8 and 11.9. The corresponding area under the curve (AUC) values for each property and each approach are given in Table 11.2.

Table 11.2: Area under the curve determined for the given samples and approaches

Approach	`elevation`	`height`	`populationTotal`
Outlier Detection (OD)	**0.872**	0.888	0.876
Cross-Checked OD	0.861	**0.891**	**0.941**
Baseline	0.745	0.847	0.847
Multi-lingual Baseline	0.669	0.509	0.860

First of all, we see that the AUC values of the cross-checked outlier detection approach are better than the baselines for all three properties. Furthermore, filtering out values that supposedly are natural outliers by cross-checking improves the AUC score for `height` and `populationTotal`. Only for `elevation`, the cross-checking leads to a slight decrease on the AUC value. Closer examination of this decrease showed this to be caused by a wrong value for `elevation` contained not only in the English dataset but also in the multi-lingual data. This duplication of wrong values could be caused by people copying values from one Wikipedia language version over to a different language version without validating the actual value using external sources. However, during our evaluation, this was not a frequent problem. Depending on the impact of such problems in larger scale, a possible solution would be to employ copy-detection approaches as those presented by Dong et al. [32].

For the property `height`, the difference in AUC between the baseline methods and our methods is considerably smaller than for the property `populationTotal`.

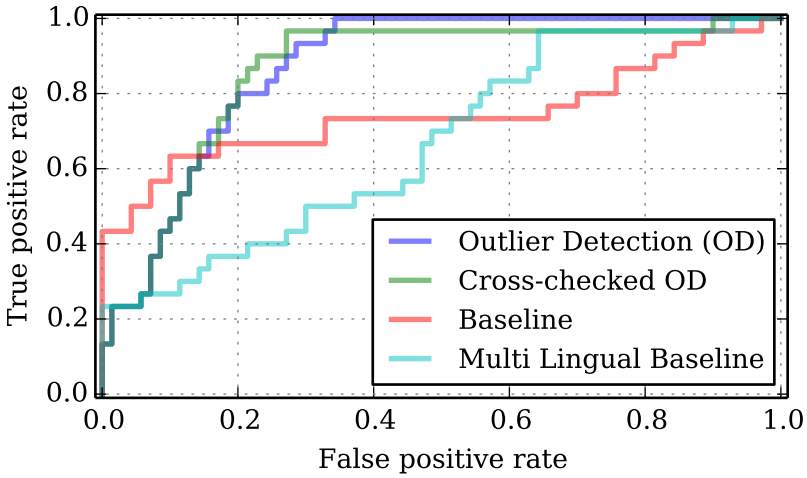


Figure 11.7: Receiver operating characteristic (ROC) for values assigned by property `elevation`

This is caused by a large number of `Person` instances in the dataset. In this specific case, the median value as used in the baseline approach is the height of a (correct) person instance. Since the `height` property for persons follows a normal distribution as also reported by Wienand and Paulheim [99], the median deviation method works particularly well and returns low scores for person instances. Though this leads to high scores for non-person instances, it gives a strong baseline for our particular dataset. Another interesting detail is that the multi-lingual baseline does not perform well. This is due to 86 instances not having enough multi-lingual data to assess their correctness. The greatest part of these instances is made up by instance of the `Person` class, in particular by athletes of sports mostly famous in English-speaking countries like rugby and baseball. Those instances are seemingly not exhaustively described in other languages. The same fact explains why the cross-checking step hardly improved the already good results of the base approach.

Finally, for the `populationTotal` property, the baseline performs well in the first parts of the examples, where it even outperforms the basic outlier detection approach. However, since the baseline does not perform constantly well on the data, the final AUC value for the outlier detection-based approach is higher. Both the very good ROC curve as well as the AUC value for the multi-lingual baseline show that we have more multi-lingual data available for `populationTotal` than for the other properties. Nonetheless, for 60 values there is not enough information for assessing the correctness. The high availability of data for the cross-checking step is also visible from the high increase for the cross-checked approach which makes it the clearly best performing approach on this dataset. This also demonstrated the advantages of combining two orthogonal directions to reach a final de-

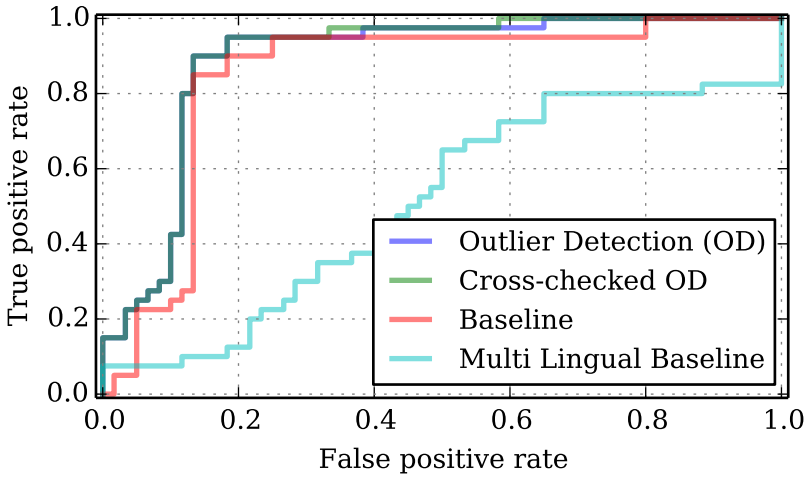


Figure 11.8: Receiver operating characteristic (ROC) for values assigned by property `height`

cision concerning the correctness of a value.

All in all, we see that the cross-checked method performs consistently well for all three properties. It always produces better results than the baseline approaches. In most cases, it is also better than the non-cross-checked approach showing that it indeed prevents natural outliers from being detected as errors.

11.3.2 Availability of Cross-Checking Data

The cross-checking step relies on the availability of additional data for the same instance reachable by following `owl:sameAs` links. For the English DBpedia dataset and the other language editions of DBpedia, we already provided the number of instances described in multiple datasets based on the work by Bryl and Bizer. In this section, we describe an additional experiment to assess the availability of the relevant data on a different Linked Data dataset. For this experiment, we chose the NELL dataset [24] in its RDF version [105] as primary dataset. The NELL dataset is produced by crawling the Web and extracting structured data from unstructured data discovered during the crawling process. Due to this creation method, we can assume that parsing errors and other difficulties result in some quality deficiencies in the data. Thus, NELL would qualify as a dataset for applying our error detection methods. We let our approach run on the longitude and latitude values of the NELL dataset. In particular, we retrieved the data to cross-check the values by using the Wikipedia links contained in the NELL data for finding the corresponding DBpedia instance. Besides the DBpedia values for longitude and latitude, we used the `owl:sameAs` links assigned to the DBpedia instances to find further instances in the Linked Data cloud which provided the desired values. We included

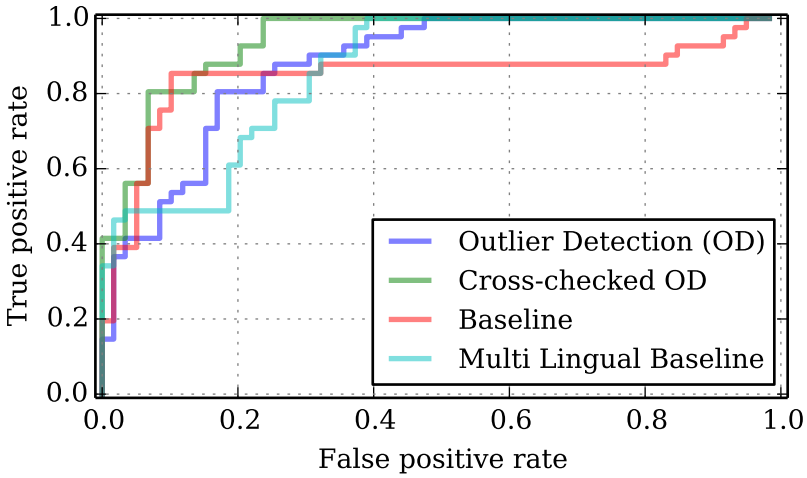


Figure 11.9: Receiver operating characteristic (ROC) for values assigned by property `populationTotal`

Table 11.3: Numbers of values found for different NELL instances

Number of values	1 (only NELL)	2	3	4	5	Total
Number of instances	6,187	5,043	3,144	6,471	13,100	33,946

the values we could retrieve from Freebase, GeoNames, YAGO and DBpedia. The corresponding statistics are shown in Table 11.3.

We did not perform a larger scale evaluation on the results of the outlier detection. However, at first sight, only few values with a sufficiently high outlier score showed up. An inspection of the some data values showed that there is close to no deviation throughout the datasets. Almost all of the inspected values were correct possibly because of the highly standardized value format for latitude and longitude which leads to only few parsing errors. The small deviations of the values seem to be caused by subjective decisions, e.g., where to exactly position the longitude-latitude marker for the area of a county. Nevertheless, the latitude value with the highest outlier score which was not filtered by the cross-checking showed to be a data error. Being assigned to the NELL instance `nell:county_grey_county`,[8] the latitude value was detected to be wrong also based on its outlierness for the population of the class `County`. An inspection of the Wikipedia page assigned by NELL showed that it should actually represent Grey County, Ontario, Canada[9] whereas the coordinates provided by NELL are in the area of Greymouth, New

[8]The prefix `nell:` is used for the full namespace `http://nell-ld.telecom-st-etienne.fr/`

[9]`http://en.wikipedia.org/wiki/Grey_County`

Zealand which belongs to the Grey District.[10] This hints to disambiguation problems. This result is in line with the findings of Paulheim [76] who also discovered that NELL has problems with homonyms when linking data. In this special case, the confusion could have been amplified by the near synonymy of district and county. All in all, this use case demonstrates the availability of data from different repositories and thus the applicability of cross-checking for improving wrong value detection.

11.4 Conclusion

In this chapter, we showed an approach enabling us to detect erroneous numerical values in Linked Data. This approach complements our previously presented schema-based methods by also allowing to find errors on a non-schema level. In contrast to previous work, the method presented here does not rely on manually crafted data correctness rules but uses the statistical methods developed in the area of outlier detection for finding suspicious values in a potentially large amount of data. This reduces the initial effort of the error detection drastically. Furthermore, our work extends previous works on using outlier detection for finding wrong numerical values in Linked Data in two ways. First, we pay more attention on efficiently selecting subpopulations of the original set of values allowing us to detect errors otherwise hidden. As a second addition, we propose a way of handling natural outliers. By following `owl:sameAs` links pointing to instances that describe the same "thing" in other datasets, we are able to aggregate additional values for the same property. Checking these additional values allows to filter out error candidates that are most probably natural outliers.

In our evaluation, we showed that our proposed approach constantly performs better than the baseline approaches though those already deliver a strong performance in ranking the values according to their probability of being erroneous. Providing such ranked lists to humans for manual approval of the error reduces the effort for finding problems in Linked Data sets. If the error is approved during the manual evaluation, it can either be fixed directly in the Linked Data source, in the original data the dataset is extracted from or the extraction method can be fixed so that similar errors are prevented in future extractions.

A major limitation of performing error detection based on outlier detection lies in the fact that only those values can be detected as errors that are different from the typical behavior of similar values. Thus, for example, detecting that an instance of the class `Person` has assigned a `height` of 1.75 m instead of 1.85 m is virtually impossible using our current approach since the assigned value is no unusual value for the height of a person. Furthermore, it is not possible to detect systematic errors in the dataset because to actually detect errors as outlier there has to be a sufficiently high number of correct values so that the typical value pattern can be determined reliably. If there are too many wrong values, as to be expected for

[10] `http://en.wikipedia.org/wiki/Grey_District`

systematic errors, these errors are as the correct behavior and only values deviating from it are marked as erroneous. Since there is no intermediate materialization of the detected, considered to be correct patterns, manual intervention into the outlier detection-based error detection is not as straightforward. This again highlights the advantage of the intermediate step of learning an ontology as done for the detection of logical errors. Possibly, both kinds of errors are detectable by considering additional data from other datasets. However, as we also saw in our experiments, the low number of instances described in multiple datasets might limit the applicability of such an approach purely working on this parallel data. Another possibility could be the inclusion of external data into the outlier detection process. Textual data might be the most obvious choice given that it represents the by far largest amount of data available. A major hindrance for including additional external data into the error detection process is the need for extracting the actually relevant data from the source data, e.g., the relevant numerical values from textual information. To provide a large enough amount of data, this extraction should be done automatically which again gives possibilities to introduce systematic errors that could interfere with the outlier detection step later-on.

One direction for further research lies in the inclusion of human annotators in the process. By assessing potential erroneous numerical values, training data can be gained that in turn can be used to more accurately determine additional suspicious values. To integrate this manually obtained data, two areas seem to be suitable. First, the application order of constraints could be determined based on previous results. For example, a subpopulation that performed well for another property might also perform well for the current one. Secondly, it could help to increase the accuracy of detecting wrong values when applying multiple outlier detection approaches instead of only one. Then, the results of a previous manual evaluation could be used to learn a proper weighting of the different single scores to produce a more appropriate total result. In particular, this could help to improve repeated runs.

By not only considering the raw values from a dataset but also allowing to combine multiple values for an instance by means of algebraic operations like substraction, additional types of errors could be recognized. For instance, the application of outlier detection to the difference of birth and death date of a person could help to identify errors in these dates. However, finding pairs for which such an operation should be applied is a non-trivial task. It could get even more complicated if not only considering combinations of values from the same instance but also allowing combinations of values connected by property paths. The latter case could help to identify errors hard to find when only considering a single instance, e.g., person instances described to be married but having birth years several hundred years apart. An alternative to applying algebraic operators to values could lie in two-dimensional outlier detection. However, depending on the specific kind of outlier detection approach employed for this, the amount of data required to get reliable results can be dramatically higher than for the one-dimensional case.

Chapter 12

Conclusion

In this chapter, we are first summarizing the previously presented work. For this purpose, we have a look at the research questions posed in the beginning of this work and answer each of them together with a short overview of the chapter relevant for the questions. Afterwards, we are giving the directions of possible future work.

Research question (RQ1) was concerned with whether we can learn OWL axioms from instance data. In the first part of Chapter 3, we therefore considered the prerequisites for doing so and finally concluded that it is possible for all OWL axioms. However, we also came to the conclusion that the open-world assumption on which OWL is based and the fact that OWL does not implement a unique name assumption might introduce the possibility of learning wrong axioms. Afterwards, we concentrated on axioms which we considered useful for debugging purposes. In Chapter 5, we therefore explored the possibilities of inductively learning class disjointness axioms from instance data available in a Linked Data repository. We compared three inductive approaches and a supervised disjointness learning method which helped us to further substantiate our answer for (R1). It showed that both, inductively learning and the supervised approach, have their strengths but also their weaknesses in certain settings. In particular, learning disjointness axioms from instance data showed generally promising results which did not depend on characteristics of an external dataset. Instead, it only relied on the data contained in the considered dataset which made it more self-contained and more suited for automatic application to arbitrary datasets without manual intervention. For the supervised approach, it turned out that differences in the characteristics of training dataset and the dataset to learn on can have considerable influence on the result unless specifically addressed by additional methods. Furthermore, the extensive analysis of the gold standard creation showed some problems humans had when enriching an ontology with class disjointness and that the task of enriching a given ontology with disjointness is a hard task even for humans. Afterwards, in Chapter 6, we extended the association rule mining-based approach for learning property-centric axioms which showed the possibility of learning axioms from in-

stance data for additional axiom types. An evaluation by experts complemented by an evaluation by means of crowd-sourcing showed the approaches to be practically applicable and have a well enough accuracy in generating the respective axioms from instance data. Furthermore, this evaluation showed that even wrongly learned axioms might help since they depend on data that shows the wrong pattern and thus can help to discover erroneous data.

While enriching ontologies with more expressive axioms, we learned that the resulting ontologies often were incoherent. Repairing these unsatisfiabilities was hindered by the fact that current reasoning tools were not able to reliably generate the set of all explanations leading to the incoherent classes for an ontology like those learned by our approaches. Thus, in Chapter 7, we implemented a rule-based reasoning tool TRex for discovering incoherences in learned ontologies. Instead of supporting the full range of expressivity, this tool focused on axioms supported by our learning approaches. In particular, it did not include reasoning for instances but only supported terminological axioms limited to those that might cause unsatisfiable classes. Experiments using this approach not only experimentally confirmed its correctness but also showed its applicability to ontologies enriched inductively by our aforementioned methods. TRex showed considerably longer runtimes than common implementation of inference tools. However, this turned out to be explainable by the sheer number of explanations that were generated. Based on this tool, we tackled research question (RQ2) on how to debug and repair incoherent learned ontologies in Chapter 8. We described four ways of determining which axioms to remove from the ontology to get it coherent. For all four approaches, the confidence values of the learned axioms were considered for reaching the final decision, thus, exploiting a special feature available for learned ontologies. Besides greedy approaches, we also included Markov Logic-based approaches which showed a superior performance. However, regarding the second part of the research question, greedy approaches showed the best characteristics when including humans into the final decision instead of repairing the ontology automatically. In particular, the greedy approaches allowed for easier understanding of the proposed repair actions due to their more local way of optimization. Thus, the proper way of repairing incoherent ontologies especially depends on whether humans are included into the process or not.

For answering research question (RQ3), we finally used the ontologies learned by means of inductive approaches for detecting errors in the dataset. In Chapter 9, we relied on patterns whose violation indicated a problem in either data, original ontology or learned axioms. In contrast to other works, we not only reported on the number of detected violations but also evaluated them so that we finally were able to give an assessment on the number of actual errors that were detected using this approach. Furthermore, we evaluated on both learned ontologies and on an ontology manually enriched with class disjointness axioms. This allowed to find out how much potential in detection is forfeited by relying on automatic enrichment methods. It showed that the detection performance using learned axioms was lower than using the gold standard, however, the difference was not high enough

to advocate the additional manual effort. By including additional property-centric axioms, the ratio of violations that actually pointed to an error in the data could be increased to almost 70%. Furthermore, it showed that violations that did not point to actual errors were caused by only a low number of different axioms whose violations could be easily filtered out after a first identification in a more interactive workflow. Wrongly learned axioms also helped to discover more widespread problems in the instance data since each learned axiom is grounded in a frequent pattern from the dataset and thus can help to identify commonly made mistakes throughout the dataset. Hence, this further showed the potential of learning axioms for getting new insights into datasets. Since the evaluation also indicated flaws in the original ontology, not only the instance data can be improved by using learned axioms for error detection but also the original ontology's quality can raised by fixing recognized shortcomings. This all helps us to answer research question (RQ3) to such an extent that an ontology automatically enriched with expressive axioms actually helps to debug Linked Data datasets. During the evaluation, we also investigated the influence of incoherence of the ontology on the results by comparing the axioms participating in incoherences to those leading to violations not indicating actual errors. We saw that incoherence only influenced the detection to a small degree and thus answered (RQ4). Nevertheless, we also saw that a prior examination of one of the proposed repair approaches would have directed our attention to a disputable modeling decision. Fixing it would have reduced the number of violations that we considered as not pointing to a data error and thus increased the accuracy of the later error detection step. However, having a coherent ontology does not show to be the first priority with respect to detecting errors in Linked Data. This is particularly entailed by the fact that pattern-based approaches for finding violations, which are best suited for application on the large number of instances in Linked Data datasets, are more resilient to incoherences than full-grown reasoning approaches. Apart from this specific application, an ontology's coherence stays an important aspect since it helps to assure adequacy for more general usage scenarios.

In Chapter 11, we proposed an approach which enabled us to efficiently find errors in numerical values in Linked Data for addressing research question (RQ5). Our approach is based on outlier detection methods and thus identifies those property values as errors that do not share the typical behavior of other values for this property. In contrast to other works relying on outlier detection for finding errors in numerical values, we further extended this basic approach by efficiently discovering subpopulations for which the application of outlier detection seems to be more promising than on the full set of property values. Furthermore, we proposed to use links in Linked Data datasets to find additional property values and employ them to identify natural outliers which otherwise would be identified as errors even though being correct. Our evaluation of this approach showed that it performs well for indicating potentially wrong numerical values in a dataset and that the inclusion of further values from linked datasets indeed reduces the number of values wrongly recognized as erroneous.

12.1 Future Work

Based on the experience we gained during our previously described work, we see several directions for future work which we provide in the following. The overall direction of this future work is to arrive at more pedantic Linked Data which exceeds the pure providing of data in a structured format and gives actual advantages compared to simple, non-semantic programmatic interfaces.

One interesting path for further work seems to be the integration of the different approaches proposed in this work. As we saw in Chapter 9, the application of learned axioms for detecting errors in the data is a promising way of improving the quality of Linked Data. This capability is not limited to the instance data contained in Linked Data but also covers the ontology itself since it allows to get a better understanding of the represented data. Thus, an interactive workflow that at the same time allows to find errors in the dataset, to fix errors in the ontology and to extend the ontology with even more axioms seems to be promising. In particular, many decision that have to be made when checking the detected violations, directly qualify for being expressed as ontology axioms. This way not only errors in Linked Data would be resolved but also the ontology quality would be raised significantly which finally is a step towards more semantic Linked Data and hence a step further into the original idea of the Semantic Web. Such a workflow could be further extended to not only provide its functionality to experts in ontologies and Linked Data but also enabling laymen to participate in improving the actual semantics of Linked Data and its ontologies. First crowd-sourcing experiments as done in Chapter 6 showed that there is some potential in this direction.

Based on the detection of errors in Linked Data, the automatic proposal of fixes for the discovered problems is another potential extension. As already done in some related works, the most nearby idea is to give a list of general applicable fixes when discovering errors caused by shortcomings of the ontology like generalizing the domain or range restriction of certain properties. Including the confidence scores of different generated axioms could give the opportunity to recognize more appropriate fixes and thus help to more quickly find the correct decision. For erroneous numerical values, approaches like examined by Bryl and Bizer [21] could be further extended and applied to generate proposals to fix the errors.

Given more expressive and more correct ontologies, a very interesting further work would be to use these ontologies to produce Linked Data that more stringently adheres to the defined terminological axioms. For example, a commonly discovered problem during the evaluation on DBpedia was a team membership link from an athlete to the city of his team because the actual team was not described in the dataset. More expressive ontologies would specify that teams and cities are disjoint as well as that a team membership should actually point to a team. In contrast to the current ontology whose quality is too low in most facets, a higher quality ontology could provide these information reliably. Considering these ontologies in the actual creation process could help to prevent the generation of wrong data and thus lead to better overall results. Instead of applying fixes at a later stage, this more in-

tegrated approach could help to preserve more data. For example, the introduction of intermediate blank nodes when a mismatch between disjointness axioms and a domain restriction shows up could preserve the data and keep it consistent.

The approaches presented in the course of this work for detecting erroneous Linked Data share one main limitation. They are not able to detect errors in the data where the error does not show up as a discrepancy from the typical data behavior. Regarding numerical errors, values that fit perfectly into the distribution of values do not arise any suspicion of being wrong and would thus not be highlighted by the outlier detection-based detection method. For link errors, the same holds true when a link does point to an instance of an appropriate class, e.g., two athletes are confused with each other. Arguably, these problems are much harder to detect since the information relevant for their detection might not be available in the dataset itself or other linked datasets. For finding such problems, the inclusion of additional external data could be promising, like extracting additional information from textual resources. However, this requires the processing of unstructured data again and needs reliable methods to find mappings between textual entities and the corresponding entities in the dataset. Such an extended error detection method spans many research areas that are currently under much research themselves and thus outreaches the scope of this work considerably.

Part IV

Appendix

Appendix A

MLN Model Based on Entailment Rules

```
// hidden predicates ************************
// TRex rules in ML
csub(cls, cls)
cdis(cls, cls)
psub(prop, prop)
pdis(prop, prop)
dom(prop, cls)
ran(prop, cls)
psubinv(prop, prop)
pdisinv(prop, prop)

*csubConf(cls, cls, float_)
*cdisConf(cls, cls, float_)
*psubConf(prop, prop, float_)
*pdisConf(prop, prop, float_)
*domConf(prop, cls, float_)
*ranConf(prop, cls, float_)
*psubinvConf(prop, prop, float_)
*pdisinvConf(prop, prop, float_)



//*****************************************
conf: csub(c1, c2) v !csubConf(c1, c2, conf)
conf: cdis(c1, c2) v !cdisConf(c1, c2, conf)
conf: psub(p1, p2) v !psubConf(p1, p2, conf)
conf: pdis(p1, p2) v !pdisConf(p1, p2, conf)
conf: dom(p1, c1) v !domConf(p1, c1, conf)
conf: ran(p1, c1) v !ranConf(p1, c1, conf)
conf: psubinv(p1, p2) v !psubinvConf(p1, p2, conf)
conf: pdisinv(p1, p2) v !pdisinvConf(p1, p2, conf)
//*****************************************

!cdis(c1, c1).
!pdis(p1, p1).
```

```
csub(c1, c1).
!cdis(c1, c2) v cdis(c2, c1).
!csub(c1,c2) v !csub(c2,c3) v csub(c1, c3).
!csub(c1,c2) v !cdis(c2,c3) v cdis(c1, c3).

psub(p1,p1).
!pdis(p1, p2) v pdis(p2, p1).
!psub(p1,p2) v !psub(p2,p3) v psub(p1, p3).
!psub(p1,p2) v !pdis(p2,p3) v pdis(p1, p3).

!dom(p,a) v !csub(a,b) v dom(p,b).
!ran(p,a) v !csub(a,b) v ran(p,b).
!psub(p,q) v !dom(q,a) v dom(p,a).
!psub(p,q) v !ran(q,a) v ran(p,a).

// 13
!cdis(a,b) v !dom(p,a) v !dom(p,b) v pdis(p,p).
!cdis(a,b) v !ran(p,a) v !ran(p,b) v pdis(p,p).

//15
!psubinv(p,q) v !dom(q,a) v ran(p,a).
!psubinv(p,q) v !ran(q,a) v dom(p,a).
!psubinv(p,q) v !psubinv(q,r) v psub(p,r).
!psubinv(p,q) v !psub(q,r) v psubinv(p,r).

// 19
!psub(p,q) v !psubinv(q,r) v psubinv(p,r).
!pdisinv(p,q) v !psub(r,q) v pdisinv(p,r).
!psubinv(p,q) v !pdis(q,r) v pdisinv(p,r).

// 22
!psubinv(p,q) v !pdis(q,r) v pdisinv(p,r).
!pdisinv(p,q) v pdisinv(q,p).
!pdisinv(p,p) v pdis(p,p).
```

Bibliography

[1] Ziawasch Abedjan and Felix Naumann. Improving RDF data through association rule mining. *Datenbank-Spektrum*, 13(2):111–120, 2013.

[2] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing Linked Data quality assessment. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013: 12th International Semantic Web Conference*, volume 8219 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2013.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[4] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H.R. Motahari-Nezhad, E. Bertino, and S. Dustdar. Quality control in crowdsourcing systems: Issues and directions. *Internet Computing, IEEE*, 17(2):76–81, 2013.

[5] Maria-Luiza Antonie and Osmar R. Zaïane. Mining positive and negative association rules: An approach for confined rules. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2004)*, volume 3202 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2004.

[6] Ron Artstein and Massimo Poesio. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596, 2008.

[7] Sören Auer and Jens Lehmann. What have Innsbruck and Leipzig in common? extracting semantics from wiki content. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *The Semantic Web: Research and Applications – ESWC 2007: 4th European Semantic Web Conference*, volume 4519 of *Lecture Notes in Computer Science*, pages 503–517. Springer, 2007.

[8] Franz Baader. Description logics. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems*, volume 5689 of *Lecture Notes in Computer Science*, pages 1–39. Springer, 2009.

[9] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, New York, NY, USA, 2nd edition, 2010.

[10] Franz Baader, Bernhard Ganter, Baris Sertkaya, and Ulrike Sattler. Completing description logic knowledge bases using formal concept analysis. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 230–235, 2007.

[11] Franz Baader and Werner Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95, 2003.

[12] Kenneth Baclawski, Mieczyslaw M. Kokar, Richard Waldinger, and Paul A. Kogut. Consistency checking of Semantic Web ontologies. In Ian Horrocks and James Hendler, editors, *The Semantic Web – ISWC 2002: First International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 454–459. Springer, 2002.

[13] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. *RDF 1.1 Turtle – Terse RDF Triple Language*. W3C Recommendation, 2014. Available at `http://www.w3.org/TR/2014/REC-turtle-20140225/`.

[14] Tim Berners-Lee. Linked Data – design issues. Technical report, World Wide Web Consortiom (W3C), 2006. Availbale at `http://www.w3.org/DesignIssues/LinkedData.html`.

[15] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 2001.

[16] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009. The Web of Data.

[17] C. Böhm, F. Naumann, Z. Abedjan, D. Fenz, T. Grütze, D. Hefenbrock, M. Pohl, and D. Sonnabend. Profiling linked open data with ProLOD. In *Workshops Proceedings of the 26th International Conference on Data Engineering (ICDE 2010)*, pages 175–178, 2010.

[18] Christian Borgelt. Frequent item set mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):437–456, 2012.

[19] Christian Borgelt and Rudolf Kruse. Induction of association rules: Apriori implementation. In *Proceedings of the 15th Conference on Computational Statistics (COMPSTAT)*, pages 395–400. Physica Verlag, 2002.

[20] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: Identifying density-based local outliers. *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 29(2):93–104, 2000.

[21] Volha Bryl and Christian Bizer. Learning conflict resolution strategies for cross-language Wikipedia data fusion. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *Proceedings of the WebQuality Workshop at the 23rd International World Wide Web Conference (WWW 2014)*, 2014.

[22] Lorenz Bühmann and Jens Lehmann. Universal OWL axiom enrichment for large knowledge bases. In Annette ten Teije, Johanna Völker, Siegfried Handschuh, Heiner Stuckenschmidt, Mathieu d'Acquin, Andriy Nikolov, Nathalie Aussenac-Gilles, and Nathalie Hernandez, editors, *Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2012)*, volume 7603 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2012.

[23] Lorenz Bühmann and Jens Lehmann. Pattern based knowledge base enrichment. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013: 12th International Semantic Web Conference*, volume 8218 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2013.

[24] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010.

[25] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, 2009.

[26] Didier Cherix, Ricardo Usbeck, Andreas Both, and Jens Lehmann. Lessons learned — the case of CROCUS: Cluster-based ontology data cleansing. In Valentina Presutti, Eva Blomqvist, Raphael Troncy, Harald Sack, Ioannis Papadakis, and Anna Tordai, editors, *The Semantic Web: ESWC 2014 Satellite Events*, Lecture Notes in Computer Science, pages 14–24. Springer, 2014.

[27] Philipp Cimiano. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer, 1st edition, 2010.

[28] Philipp Cimiano and Johanna Völker. Text2Onto. In Andrés Montoyo, Rafael Muńoz, and Elisabeth Métais, editors, *Natural Language Processing and Information Systems*, volume 3513 of *Lecture Notes in Computer Science*, pages 227–238. Springer, 2005.

[29] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 2nd edition, 1988.

[30] Óscar Corcho, Catherine Roussey, Luis Manuel Vilches Blázquez, and Iván Pérez. Pattern-based OWL ontology debugging guidelines. In Eva Blomqvist, Kurt Sandkuhl, François Scharffe, and Vojtech Svátek, editors, *Proceedings of the Workshop on Ontology Patterns (WOP 2009) , collocated with the 8th International Semantic Web Conference (ISWC 2009)*. CEUR-WS, 2009.

[31] Claudia d'Amato, Nicola Fanizzi, and Floriana Esposito. Inductive learning for the Semantic Web: What does it buy? *Semantic Web Journal*, 1(1,2):53–59, April 2010.

[32] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Truth discovery and copying detection in a dynamic world. *Proceedings of the VLDB Endowment*, 2009.

[33] Jianfeng Du, Guilin Qi, and Qiu Ji. Goal-directed module extraction for explaining OWL DL entailments. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *The Semantic Web – ISWC 2009: 8th International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2009.

[34] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad, editors, *Proceedings of the Second International Confierence on Knowledge Discovery and Data Mining*. The AAAI Press, 1996.

[35] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching, Second Edition*. Springer, 2013.

[36] Daniel Fleischhacker, Christian Meilicke, Johanna Völker, and Mathias Niepert. Computing incoherence explanations for learned ontologies. In Wolfgang Faber and Domenico Lembo, editors, *Web Reasoning and Rule Systems*, volume 7994 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2013.

[37] Daniel Fleischhacker, Christian Meilicke, Johanna Völker, and Mathias Niepert. Technical report: Computing incoherence explanations for learned ontologies. Technical report, University of Mannheim, 2013.

[38] Daniel Fleischhacker, Heiko Paulheim, Volha Bryl, Johanna Völker, and Christian Bizer. Detecting errors in numerical Linked Data using cross-checked outlier detection. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble, editors, *The Semantic Web – ISWC 2014: 13th International Semantic Web Conference*, volume 8796 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2014.

[39] Daniel Fleischhacker and Johanna Völker. Inductive learning of disjointness axioms. In Robert Meersman, Tharam Dillon, Pilar Herrero, Akhil Kumar, Manfred Reichert, Li Qing, Beng-Chin Ooi, Ernesto Damiani, DouglasC. Schmidt, Jules White, Manfred Hauswirth, Pascal Hitzler, and Mukesh Mohania, editors, *On the Move to Meaningful Internet Systems: OTM 2011 – Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011*, volume 7045 of *Lecture Notes in Computer Science*, pages 680–697. Springer, 2011.

[40] Daniel Fleischhacker, Johanna Völker, and Heiner Stuckenschmidt. Mining RDF data for property axioms. In Robert Meersman, Hervé Panetto, Tharam Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and Isabel F. Cruz, editors, *On the Move to Meaningful Internet Systems: OTM 2012 – Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012*, volume 7566 of *Lecture Notes in Computer Science*, pages 718–735. Springer, 2012.

[41] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pages 124–133, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[42] Birte Glimm, Aidan Hogan, Markus Krötzsch, and Axel Polleres. OWL: yet to arrive on the Web of Data? In Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas, editors, *WWW 2012 Workshop on Linked Data on the Web*, volume 937 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

[43] Birte Glimm, Ian Horrocks, Boris Motik, and Giorgos Stoilos. Optimising ontology classification. In *The Semantic Web – ISWC 2010: 9th International Semantic Web Conference*, pages 225–240, 2010.

[44] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. HermiT: An OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.

[45] Nicola Guarino and Christopher A. Welty. An overview of OntoClean. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 201–220. Springer, 2009.

[46] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[47] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 29(2):1–12, May 2000.

[48] Steve Harris and Andy Seaborne, editors. *SPARQL 1.1 Query Language*. W3C Recommendation, 2013. Available at `http://www.w3.org/TR/sparql11-query/`.

[49] Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational Linguistics (COLING 1992) - Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[50] Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of OWL class descriptions on very large knowledge bases. *International Journal on Semantic Web and Information Systems*, 5(2):25–48, 2009.

[51] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 2012. Available at `http://www.w3.org/TR/owl2-primer/`.

[52] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, pages 57–67, 2006.

[53] Ian Horrocks and Ulrike Sattler. Decidability of SHIQ with complex role inclusion axioms. *Journal of Artificial Intelligence*, 160(1-2):79–104, 2004.

[54] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning*, volume 1705 of *Lecture Notes in Computer Science*, pages 161–180. Springer, 1999.

[55] Qiu Ji, Peter Haase, Guilin Qi, Pascal Hitzler, and Steffen Stadtmüller. RaDON — repair and diagnosis in ontology networks. In Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen,

Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Simperl, editors, *The Semantic Web: Research and Applications – ESWC 2009: 6th European Semantic Web Conference*, volume 5554 of *Lecture Notes in Computer Science*, pages 863–867. Springer, 2009.

[56] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *The Semantic Web – ISWC 2007 + ASWC 2007: 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, pages 267–280. Springer, 2007.

[57] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A web ontology editing browser. *Journal of Web Semantics*, 4(2):144–153, 2006. Semantic Grid –The Convergence of Technologies.

[58] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics*, 3(4):268–293, 2005.

[59] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of Linked Data quality. In *Proceedings of the 23rd International Conference on World Wide Web (WWW 2014)*, pages 747–758. ACM, 2014.

[60] Dimitris Kontokostas, Amrapali Zaveri, Sören Auer, and Jens Lehmann. Triplecheckmate: A tool for crowdsourcing the quality assessment of Linked Data. In Pavel Klinov and Dmitry Mouromtsev, editors, *Knowledge Engineering and the Semantic Web*, volume 394 of *Communications in Computer and Information Science*, pages 265–272. Springer, 2013.

[61] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. LoOP: Local outlier probabilities. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM 2009)*, pages 1649–1652. ACM, 2009.

[62] Solomon Kullback and Richard. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[63] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159—174, 1977.

[64] Jens Lehmann. *Learning OWL Class Expressions*, volume 6 of *Studies on the Semantic Web*. AKA Heidelberg, 2010.

[65] Jens Lehmann and Lorenz Bühmann. ORE - a tool for repairing and enriching knowledge bases. In Peter F.F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors,

The Semantic Web: ISWC 2010 – 9th International Semantic Web Conference, volume 6497 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2010.

[66] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web Journal*, 6(2):167–195, 2015.

[67] Vladimir Levenshtein. Binary codes capable of correcting deletions and insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[68] Alexander Maedche and Steffen Staab. Discovering conceptual relations from text. In Werner Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000)*, pages 321–325. IOS Press, 2000.

[69] Alexander Maedche and Steffen Staab. Ontology learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79, 2001.

[70] Deborah L. McGuinness and Frank van Harmelen, editors. *OWL Web Ontology Language Overview*. W3C Recommendation, 2004. Available at `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[71] André Melo, Martin Theobald, and Johanna Völker. Correlation-based refinement of rules with numerical attributes. In William Eberle and Chutima Boonthum-Denecke, editors, *Proceedings of the 27th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2014)*. AAAI Press, 2014.

[72] George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[73] Jan Noessner and Mathias Niepert. ELOG: A probabilistic reasoner for OWL EL. In Sebastian Rudolph and Claudio Gutierrez, editors, *Proceedings of the 5th International Conference on Web Reasoning and Rule Systems (RR 2011)*, pages 281–286. Springer, 2011.

[74] Jan Noessner, Mathias Niepert, and Heiner Stuckenschmidt. RockIt: Exploiting parallelism and symmetry for MAP inference in statistical relational models. *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, 2013.

[75] Heiko Paulheim. Browsing Linked Open Data with auto complete. In *10th Semantic Web Challenge at the 11th International Semantic Web Conference (ISWC 2012)*, 2012.

[76] Heiko Paulheim. Identifying wrong links between datasets by multi-dimensional outlier detection. In Patrick Lambrix, Guilin Qi, Matthew Horridge, and Bijan Parsia, editors, *Proceedings of the Third International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM 2014) co-located with 11th Extended Semantic Web Conference (ESWC 2014)*, pages 27–38. CEUR-WS, 2014.

[77] Heiko Paulheim and Christian Bizer. Type inference on noisy RDF data. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013: 12th International Semantic Web Conference*, volume 8218 of *Lecture Notes in Computer Science*, pages 510–525. Springer, 2013.

[78] María Poveda-Villalón, María del Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Validating ontologies with OOPS! In Annette ten Teije, Johanna Völker, Siegfried Handschuh, Heiner Stuckenschmidt, Mathieu d'Aquin, Andriy Nikolov, Nathalie Aussenac-Gilles, and Nathalie Hernandez, editors, *Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2012)*, pages 267–281. Springer, 2012.

[79] Youen Péron, Frédéric Raimbault, Gildas Ménier, and Pierre-François Marteau. On the detection of inconsistencies in RDF data sets and their correction at ontological level. Technical report, VALORIA - Laboratoire de Recherche en Informatique et ses Applications de Vannes et Lorient, 2011.

[80] Guilin Qi, Peter Haase, Zhisheng Huang, Qiu Ji, Jeff Z. Pan, and Johanna Völker. A kernel revision operator for terminologies - algorithms and evaluation. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web – ISWC 2008: 7th International Semantic Web Conference*, pages 419–434. Springer, 2008.

[81] Matthew Richardson and Pedro Domingos. Markov logic networks. *Journal of Machine Learning*, 62(1-2):107–136, 2006.

[82] Catherine Roussey and Ondrej Zamazal. Antipattern detection: how to debug an ontology without a reasoner. In Patrick Lambrix, Guilin Qi, Matthew Horridge, and Bijan Parsia, editors, *Proceedings of the Second International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM 2013)*, pages 45–56. CEUR-WS, 2013.

[83] Stefan Scheglmann, Gerd Gröner, Steffen Staab, and Ralf Lämmel. Incompleteness-aware programming with RDF data. In Evelyne Viegas, Karin Breitman, and Judith Bishop, editors, *Proceedings of the Workshop on*

Data Driven Functional Programming (DDFP 2013), pages 11–14. ACM, 2013.

[84] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications – ESWC 2005: Second European Semantic Web Conference*, volume 3532 of *Lecture Notes in Computer Science*, pages 226–240. Springer, 2005.

[85] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, volume 18, pages 355–362, 2003.

[86] Guus Schreiber and Yves Raimond, editors. *RDF 1.1 Primer*. W3C Recommendation, 2014. Available at `http://www.w3.org/TR/owl2-primer/`.

[87] Zhaohua Sheng, Xin Wang, Hong Shi, and Zhiyong Feng. Checking and handling inconsistency of DBpedia. In FuLee Wang, Jingsheng Lei, Zhiguo Gong, and Xiangfeng Luo, editors, *Proceedings of the International Conference on Web Information Systems and Mining (WISM 2012)*, volume 7529 of *Lecture Notes in Computer Science*, pages 480–488. Springer, 2012.

[88] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007. Software Engineering and the Semantic Web.

[89] Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. A modularization-based approach to finding all justifications for OWL DL entailments. In John Domingue and Chutiporn Anutariya, editors, *The Semantic Web – ASWC 2008: 3rd Asian Semantic Web Conference*, volume 5367 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[90] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L. McGuinness. Integrity constraints in OWL. In Maria Fox and David Poole, editors, *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*. AAAI Press, 2010.

[91] Wei-Guang Teng, Ming-Jyh Hsieh, and Ming-Syan Chen. On the mining of substitution rules for statistically dependent items. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2002)*, pages 442–449. IEEE Computer Society, 2002.

[92] Gerald Töpper, Magnus Knuth, and Harald Sack. DBpedia ontology enrichment for inconsistency detection. In Valentina Presutti and Helena Sofia Pinto, editors, *Proceedings of the 8th International Conference on Semantic Systems (I-SEMANTICS 2012)*, pages 33–40. ACM, 2012.

[93] Paola Velardi, Roberto Navigli, Alessandro Cucchiarelli, and Francesca Neri. Evaluation of OntoLearn, a methodology for automatic learning of domain ontologies. In Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini, editors, *Ontology Learning from Text: Methods, Evaluation and Applications*, pages 92–106. IOS Press, 2005.

[94] Johanna Völker, Daniel Fleischhacker, and Heiner Stuckenschmidt. Automatic acquisition of class disjointness. *Journal of Web Semantics*, 35:124–139, 2015.

[95] Johanna Völker, Pascal Hitzler, and Philipp Cimiano. Acquisition of OWL DL axioms from lexical resources. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *The Semantic Web: Research and Applications – ESWC 2007: 4th European Semantic Web Conference*, volume 4519 of *Lecture Notes in Computer Science*, pages 670–685. Springer, 2007.

[96] Johanna Völker and Mathias Niepert. Statistical schema induction. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter Leenheer, and Jeff Pan, editors, *The Semantic Web: Research and Applications – ESWC 2011: 8th Extended Semantic Web Conference*, volume 6643 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2011.

[97] Johanna Völker and Sebastian Rudolph. Lexico-logical acquisition of OWL DL axioms. In Raoul Medina and Sergei Obiedkov, editors, *Proceedings of the 6th International Conference on Formal Concept Analysis (ICFCA 2008)*, volume 4933 of *Lecture Notes in Computer Science*, pages 62–77. Springer, 2008.

[98] Johanna Völker, Denny Vrandečić, York Sure, and Andreas Hotho. Learning disjointness. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *The Semantic Web: Research and Applications – ESWC 2007: 4th European Semantic Web Conference*, volume 4519 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2007.

[99] Dominik Wienand and Heiko Paulheim. Detecting incorrect numerical data in dbpedia. In Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai, editors, *The Semantic Web: Trends and Challenges – ESWC 2014: 11th Extended Semantic Web Conference*, volume 8465 of *Lecture Notes in Computer Science*, pages 504–518. Springer, 2014.

[100] Gang Wu, Guilin Qi, and Jianfeng Du. Finding all justifications of OWL entailments using TMS and MapReduce. In Craig Macdonald, Iadh Ounis, and Ian Ruthven, editors, *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 1425–1434. ACM, 2011.

[101] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics (ACL 1994)*, pages 133–138. Association for Computational Linguistics, 1994.

[102] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In David Heckerman, Heikki Mannila, and Daryl Pregibon, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD 1997)*, pages 283–286. AAAI Press, 1997.

[103] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for linked data: A survey. *Semantic Web Journal*, 7(1):63–93, 2016.

[104] Chengqi Zhang and Shichao Zhang. *Association Rule Mining: Models and Algorithms*. Springer, 2002.

[105] Antoine Zimmermann, Christophe Gravier, Julien Subercaze, and Quentin Cruzille. Nell2RDF read the Web, and turn it into RDF. In *Proceedings of the 2nd International Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data*, 2013.