INTEGRATING LINKED DATA SEARCH RESULTS USING STATISTICAL RELATIONAL LEARNING APPROACHES

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2016

By
Duhai Alshukaili
School of Computer Science

Contents

Word Count: 36,126

List of Tables

List of Figures

Listings

List of Algorithms

Abstract

Integrating Linked Data Search Results
Using Statistical Relational Learning Approaches
Duhai Alshukaili
A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy, 2016

Linked Data (LD) follows the web in providing low barriers to publication, and in deploying web-scale keyword search as a central way of identifying relevant data. As in the web, searches initially identify results in broadly the form in which they were published, and the published form may be provided to the user as the result of a search. This will be satisfactory in some cases, but the diversity of publishers means that the results of the search may be obtained from many different sources, and described in many different ways. As such, there seems to be an opportunity to add value to search results by providing users with an integrated representation that brings together features from different sources. This involves an on-the-fly and automated data integration process being applied to search results, which raises the question as to what technologies might be most suitable for supporting the integration of LD search results.

In this thesis we take the view that the problem of integrating LD search results is best approached by assimilating different forms of evidence that support the integration process. In particular, this dissertation shows how Statistical Relational Learning (SRL) formalisms (viz., Markov Logic Networks (MLN) and Probabilistic Soft Logic (PSL)) can be exploited to assimilate different sources of evidence in a principled way and to beneficial effect for users. Specifically, in this dissertation we consider syntactic evidence derived from LD search results and from matching algorithms, semantic evidence derived from LD vocabularies, and user evidence, in the form of feedback.

This dissertation makes the following key contributions: (i) a characterisation of key features of LD search results that are relevant to their integration, and a description of some initial experiences in the use of MLN for interpreting search results; (ii) a PSL rule-base that models the uniform assimilation of diverse kinds of evidence; (iii) an empirical evaluation of how the contributed MLN and PSL approaches perform in terms of their ability to infer a structure for integrating LD search results; and (iv) concrete examples of how populating such inferred structures for presentation to the end user is beneficial, as well as guiding the collection of feedback whose assimilation further improves search results presentation.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

This dissertation is a culmination of a journey that would not have been possible without the support of many people to whom I would like to express my sincere thanks and gratitude. I am fortunate to have two of the most distinguished researchers as my supervisors, Dr Alvaro A. A. Fernandes and Prof. Norman W. Paton, who showed me through their example how to be a committed candidate, with defined goals, while maintaining the clarity in seeking the path to achieving my goals. Without their invaluable support, guidance, patience and advice, my graduate student experience would not have been as fulfilling and enriching.

I would to also thank my fellow graduate students with whom I shared discussions, distractions and food. Many thanks to Fernando Osorno, Klitos Christodoulou, Alan Stokes, Julio Cesar, René Sánchez, Iliada Eleftheriou, Ivelize Rocha Bernardo, Mariam Alqasab, Nurzety Ahmed and many more.

Sincere thanks to my brothers and sisters for their love and continued support while I was away from Oman. To my Mom and Dad for believing in me and for your love and support during all the years, I spent away from home. I am forever grateful for everything you provided for me, and I would like to dedicate this thesis to you.

Last but not least, I would like to express my heartfelt appreciation and gratitude to the love my life, to the two most beautiful girls, Sara and Alaa. Thank you for being here with and for me in this journey. I most certain that I could not have done this without you.

Manchester, September 2016

Duhai Alshukaili

To my Mom and Dad.

Glossary and Acronyms

RDF Resource Description Framework.

WoD Web of Data.

SW Semantic Web.

LD Linked Data.

SRL Statistical Relational Learning.

PSL Probabilistic Soft Logic.

MLNs Markov Logic Networks.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

HTTP Hypertext Transfer Protocol.

AUC PR Area Under the Precision-Recall Curve.

Chapter 1

Introduction

The Web of Data (WoD) is an extension of the Web of documents. While the Web of documents consists of a set of hyperlinked documents, the WoD is composed of a set of interlinked datasets. The aim of the WoD is to give a well-defined meaning to information items so that they can be processed by machines [Berners-Lee et al., 2001]. Semantic Web (SW) standards such as the Resource Description Framework (RDF) [Cyganiak et al., 2014] and the Web Ontology Language (OWL) [McGuinness and Van Harmelen, 2004] allow independent publishers to create machine-processable data sources. The foundations for the WoD have been laid out by these technologies that characterize the Web: HyperText Transfer Protocol (HTTP) and Uniform Resource Identifiers (URIs). HTTP provides a decentralized mechanism by which resources are shared over the Web. Instead of HTML pages, the resources — anything with an identity — in the WoD are descriptions encoded as *subject-predicate-object* triples using the RDF model. Resources are identified by URIs, which enable a the assignment of unique names to resources. For example, the URI `http://www.wikidata.org/entity/Q41163` identifies the American stage actor and director whose common name is `Al Pacino`. Using URIs to name resources in the WoD also provides a uniform way to access the information that describes the identified resource.

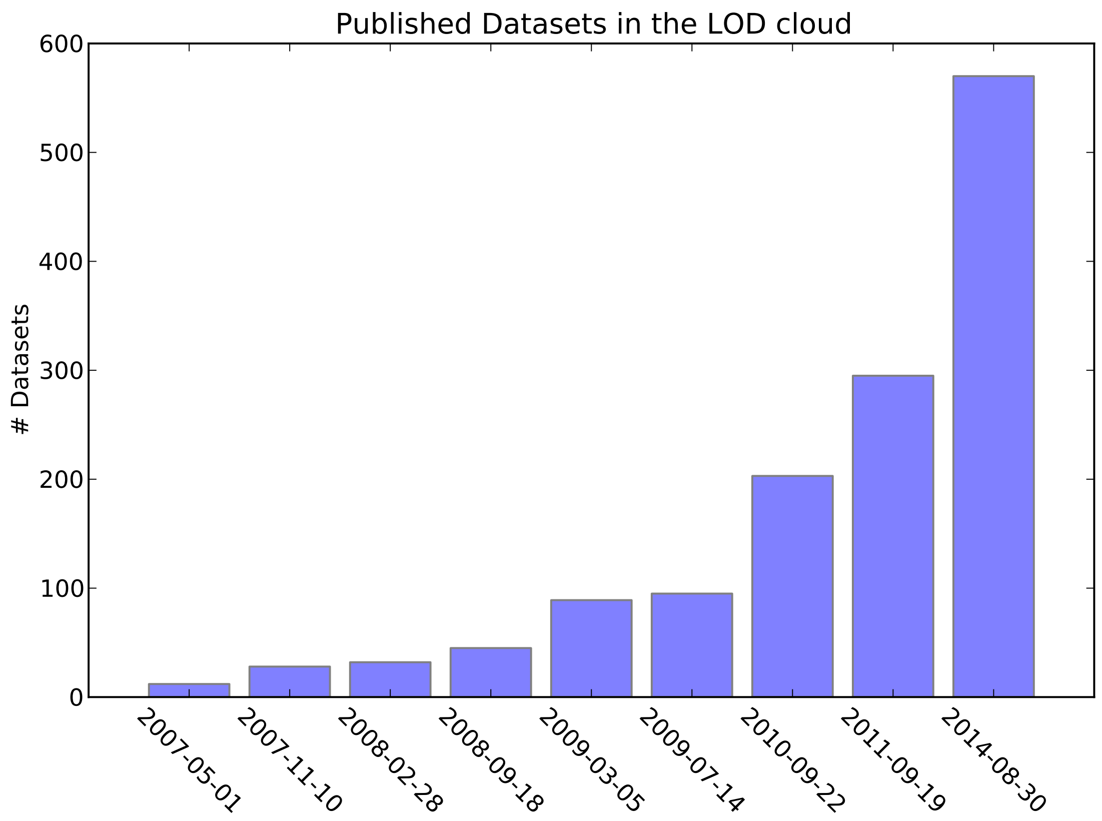


Figure 1.1: Growth in the number of datasets published on the WoD under the LOD project[2]

With community efforts such as W3C's Linking Open Data (LOD) project[1], the amount of Linked Data (LD) on the Web has been growing steadily. This W3C project began in 2007, with the aim to seed the WoD by converting existing datasets to RDF and make them available on the Web. Figure 1.1 shows how the number of datasets published on the Web has grown since the LOD project has began. A May 2016 estimate of the amount of LD on the Web suggests that there are more than 2800 datasets containing about 190 billion triples of data, of which about 5% are accessible by remote querying[3]. In addition to publishing datasets, there is an increasing trend towards the adoption of embedded metadata within web documents [Guha et al., 2016]. Such a trend is expected to further increase as more

[1]`esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`
[2]Based on data from the state of the LOD cloud website (`lod-cloud.net/`)
[3]`stats.lod2.eu/`

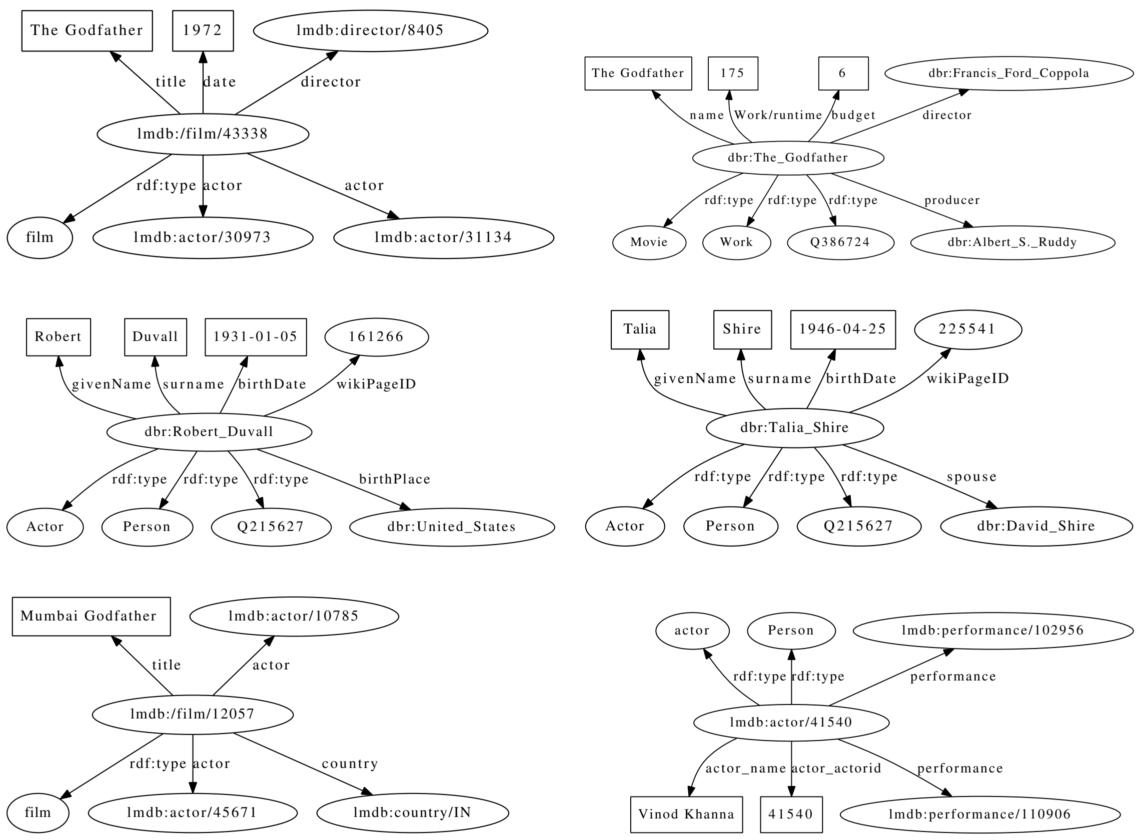


Figure 1.2: Example LD search results for the term `"Godfather actors"`

tools become available that support semantic annotations of Web content.

As LD publishing grew in popularity, tools that facilitate access to the WoD began to emerge. Among these tools are LD search engines. LD search engines, such as Sindice [Oren et al., 2008] and Falcons [Cheng and Qu, 2009], crawl and index RDF data on the Web in order to provide keyword-search capabilities to end users (including software agents). LD search engines are useful, but their results, which take the form of a collection of RDF resources, present to the user the data *as published*. For a human user, such data can be time-consuming and cumbersome to explore.

In essence, LD search engines tackle the question *What resources are out there that match the search?*, and not so much the question of *What data out there match the search?*. For example, a search for `Godfather actors` returns results (as shown in Figure 1.2), among others, that are about two distinct *films* whose name contain the string *Godfather*, as well as about *actors* that have appeared in those films. Assuming for the moment that the user is looking for data about actors in the 1972 US film named The Godfather[4], integrating the results in different tables, as shown in Figure 1.3, would be desirable since it distinguishes between films and actors and provides a structure to the presentation of the data from the underlying resources. This dissertation, explores the opportunity to complement the work to date on search engines for LD by devising techniques that enable the integration of search results through structure inference, thereby enabling the generation of tabular reports of the results. The reason we target tabular report is that they provide an intuitive view of data for users who are used to tables such as spreadsheets and Web tables.

We know of one previous proposal that address this problem, namely Sig.ma [Tummarello et al., 2010]. Sig.ma aimed to generate a report that integrated the top search results from the Sindice search engine. In Sig.ma, several steps were followed to integrate the data by using syntactic matchers, as discussed more fully in Chapter 2. The overall approach assumed that the results describe a single entity. Relying on syntactic matchers to integrate data results effects the resulting integration. For example, if a user is interested in information about *Manchester University*, Sig.ma combines data about a university, a grammar school, and a railway station. This is because the term *Manchester* gives a rise for descriptions about entities which belong to heterogeneous data types. Relying on syntactic matchers only causes Sig.ma to include data values from heterogeneous descriptions. Sig.ma allows the users to interact with the resulting integration with a view to choosing the resources that best describe the entity being searched. However, such feedback does not affect future searches.

Our approach is distinct from Sig.ma in a number ways. As we will describe in Chapters 3 and 4, we use a knowledge based uncertainty reasoning rather than

[4]`en.wikipedia.org/wiki/The_Godfather`

heuristics to represent the integration of LD search results. In doing so, we acknowledge that the results from LD search span different types and potentially describe different real world entities. Thus, it is not sufficient to present the user with a single entity description. In our approach, we represent user feedback as a first class citizen in our model, thus facilitating the capture and reuse of feedback.

Movie				
name	**date**	**director**	**budget**	**runtime**
The Godfather	1972	Francis Ford Coppola	6	172

Actor				
givenName	**surname**	**birthDate**	**birthPlace**	**spouse**
Robert	Duvall	1931-01-05	United States	null
Talia	Shire	1946-04-25	null	David Shire

Figure 1.3: Results in Fig. 1.2 integrated and reported as tables

1.1 LD Integration Challenges

The Semantic Web standards provide a basis for publishing interoperable data on the Web. The widely agreed-upon LD principles proposed by Berners-Lee [2006] advocate the use of a uniform model for publishing data in the form of subject-predicate-object triples. Under this model, real-world entities are given unique URIs to enable global identification of these entities. Dereferencing these URIs allows one to discover more information about the classes or properties (i.e. schema level information) of the underlying entities. Ideally, using consistent naming across all datasets paves the way for more accurate alignment and integration of resources found in different LD datasets. However, achieving consensus on how things are named and identified in the WoD is infeasible in practice, therefore the goal of integrating LD search results can only be reached if the challenges that arise from this lack of consensus are successfully addressed.

Although the problem in hand can be viewed as a data integration problem, it is quite unlike classical data integration, in which typically there is a known target (or global) schema to which the source data should be mapped [Doan et al., 2012]. The absence of such a target rules out the use of standard mapping generation algorithms for this task. Another difficulty stems from the fact that LD exhibits two types of heterogeneities: *instance-level* and *terminological-level* heterogeneities [Christodoulou, 2015].

Instance-level heterogeneities are manifested through disagreement as to what identifies resources in the WoD. This disagreement sometimes leads to fragmented and contradictory descriptions of the underlying entities. Instance-level heterogeneities complicate the task of integrating fragments of relevant data from multiple resources to populate a single row in the tabular representation we are using as our desired target formalism. To see this, consider Figure 1.4 which depicts the resources `http://dbpedia.org/resource/Berlin` and `http://sws.geonames.org/2950159/` returned for the search term `Berlin`, both of which describe the same entity (*i.e., Berlin, Germany*). To align these resources, one needs to recognise that both resources describe entities which that are instances of the same real-world entity type (*i.e., city*). However, the information provided by `http://sws.geonames.org/2950159/` does not explicitly allow one to do so. Second, it is necessary to align predicates in both resources which is not straightforward because of the inconsistencies that occur at the syntactic level (e.g. `dbp:populationTotal` and `geonames:population`).

In addition to instance-level heterogeneities, LD consumers need to deal with terminological heterogeneities as well. Heterogeneities at the terminology level arise because publishers either reuse different existing ontologies or invent their own. In the running example, the predicates `dbo:country` and `geonames:parentCountry` are both being used to assert the name of the country where an entity (*e.g., Berlin*) is located. This becomes an impediment for applications that aim to consume LD, as they have to match against various ontologies before they can obtain a better integrated view.

Another challenge in integrating LD search results is discerning user intent from

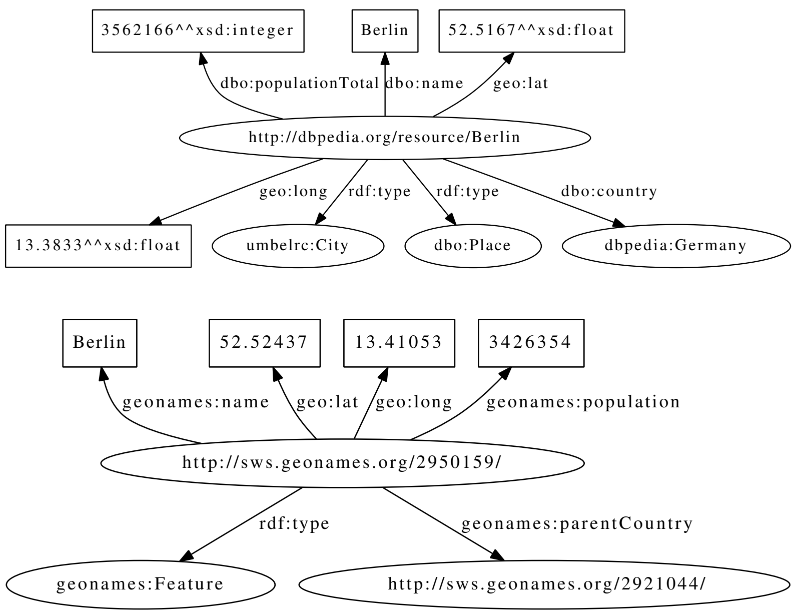


Figure 1.4: Some of the search results returned for the term `"Berlin"`

keyword searches. It is inherently difficult to determine user intention due to ambiguity in search terms [Baeza-Yates et al., 2006]. For example, a user might type the search term `Casablanca` when looking for information about the 1942 US film[5], whereas the system returns resources about the city of *Casablanca*[6] instead. While there are automated methods for classifying user intent in keyword search, these often require access to search logs [Jansen et al., 2008], which is an assumption that cannot be made in our setting because we aim not to be tightly coupled to any particular search engine.

1.2 Hypothesis

The challenges described in Section 1.1 are not unique to our problem as they have been addressed by the Semantic Web and the database communities for years, but in different settings than ours. One approach to integrate datasets in the absence of a global schema is to utilize existing domain vocabularies as a mediator between

[5]`en.wikipedia.org/wiki/Casablanca_(film)`
[6]`en.wikipedia.org/wiki/Casablanca`

the underlying datasets (e.g. Giunchiglia et al. [2005]; Huhns et al. [1993]). Also, there has been an acknowledgment by the LD community of the challenges raised by heterogeneities in WoD resources. Based on envisioning the WoD as a web-scale database [Heath and Bizer, 2011], an ongoing incremental effort is under-way to reduce the effects of heterogeneity at different levels. At the instance level, a number of approaches [Isele and Bizer, 2013; Hu et al., 2011; Ngomo and Auer, 2011] have been proposed to interlink co-referent resources, i.e., resources that represent the same real world entity. This often requires the use of de-duplication techniques, which in the WoD often results in new `owl:sameAs` relationships between resources. Similarly, there have been numerous efforts to infer semantic correspondences between the vocabularies. This is evident from the plethora of approaches to ontology matching (see [Ferrara et al., 2011] for a comprehensive survey and [Euzenat and Shvaiko, 2013] for a recent book on the problem). Although the heterogeneity issues are far from being resolved due to the continued evolution of the WoD, existing approaches provide useful, albeit partial, evidence about relationships between resources. This evidence can be useful to support the decisions made by automated algorithms that target the integration of LD search results.

Furthermore, one can capitalize on feedback obtained from the user, as is often done *pay-as-you-go* systems [Hedeler et al., 2010] in order to address one or more of the challenges described in Section 1.1. For example, user feedback is utilized by ontology alignment systems such as CrowdMap Sarasua et al. [2012] and ZenCrowd Demartini et al. [2012] in order achieve better results over automatic alignment methods. Also, in LD context, Sig.ma [Tummarello et al., 2010], which also addressed the problem of integrating LD search results (see Chapter2), uses feedback from the user to identify sources of relevant data to the search term provided by the user. Additionally, in traditional information retrieval (IR) approaches, eye-tracking (e.g., [Umemoto et al., 2012]) and click-through (e.g., [Dou et al., 2008]) data are examples of feedback evidence used to discern the user intent behind the provided search term.

Based on the above observations of how the challenges of integrating LD search results are addressed in other settings, our hypothesis is as follows.

> LD search results can be integrated to yield a higher-level structured representation by
>
> 1. combining different sources of evidence that inform the integration process;
> 2. systematically managing the uncertainty associated with these sources; and
> 3. making use of feedback.

The central research question that is addressed in this dissertation is: *What kind of approaches can usually be adopted for systematically incorporating different kinds of evidence in the integration of LD search results?*

To address this question, this thesis investigates the application of a Statistical Relational Learning (SRL) approach which we now briefly introduce.

1.3 Statistical Relational Learning: A framework for managing evidence

In many machine learning domains, the assumption that data is composed of identically structured objects cannot be made [Getoor and Taskar, 2007; Domingos and Lowd, 2009a]. Examples of such domains include social networks, the Web, natural language, and so on. To model such domains, it is necessary to capture not only the structure (in the form of attributes) of individual objects, but also the relationships between them. For example, to accurately model a social network, one needs to capture the dependencies induced by relationships (e.g., friendship) between the individuals in the network. Using relational data in such domains leads to more accurate results from traditional machine learning tasks such as classification and predication [Sen et al., 2008]. For example, individuals in a social network are likely to have similar interests depending on whether they are friends or not. Statistical Relational Learning (SRL) is a subfield of machine learning that seeks to build probabilistic models of relational data, i.e., data that capture not just objects but also relationships between objects [Getoor and Taskar, 2007].

SRL approaches can be categorized by their choice of representation, and by the probabilistic semantics for dealing with uncertainty. Representation is often done through either logic (e.g., first-order-logic) or frame-based (e.g., entity-relationship models). Most SRL approaches are based on probabilistic graphical models (PGMs). Two common types of graphical models used in SRL are Bayesian networks and Markov Networks [Pearl, 1988].

SRL approaches that combine first-order-logic (FOL) and PGMs are suitable for reasoning with uncertainty in relational domains using multiple sources of evidence. In such approaches, weighted FOL rules are defined that model a domain of interest to the modeler. The model is expressed in terms of first-order predicates and functions. Given some input data, which have been pre-processed as ground*evidence predicates*, the weighted FOL rules then are grounded into a PGM. This grounded PGM provides the probabilistic semantics of the domain and can be used to perform inference over so called *query predicates*. In this thesis, we show how SRL can be used to assimilate different sources of evidence in a principled way. Different forms of evidence can be expressed as logical predicates defined over the input data. The dependencies induced by the different kinds of evidence are encoded as FOL rules. The uncertainty associated with the combination of different forms of evidence is captured as parameters of the PGM that results from grounding the FOL rules. The combination of FOL and PGMs allows for the joint inference, i.e., probabilistic inference in the presence of different sorts of evidence, of claims about objects in the domain.

SRL approaches such as Markov Logic Networks (MLN) [Domingos and Lowd, 2009a] and Probabilistic Soft Logic (PSL) [Bach et al., 2015] have been used to model domains that (*i*) exhibit relational dependencies, (*ii*) require the integration of facts from multiple sources of evidence, and (*iii*) require principled treatment of the uncertainty that arises from the use of multiple sources of evidence. For example the authors of Elementary [Niu et al., 2012] and DeepDive [Shin et al., 2015] use MLN to build structured knowledge bases from text in Web pages. In their approach, they supplement domain specific rules (e.g., about marriage relationship) with facts derived from syntactic matches and existing knowledge bases such as Freebase [Bollacker et al., 2008] in order to extract relationships about entity mentions in the textual data sources. Similarly, Pujara et al. [2013] used PSL to infer a knowledge graph [Singhal, 2012] from triples of data produced by web extractors such as NELL [Mitchell et al., 2015]. To achieve this, they utilized PSL to model three canonical SRL tasks: *entity resolution*, *collective classification* and *link prediction*. They used evidence extracted from ontologies to enforce global constraints on the inferred knowledge graph. Although a direct comparison to these approaches is not applicable to our proposed methodology, given that our emphasis is on the integration of LD search, these approaches demonstrate how SRL representations are useful in problem domains where reasoning with multiple sources of evidence is required.

1.4 Aim, Objectives and Research Contributions

In this dissertation, we explore the application of SRL techniques to infer a structure that enable us to construct tabular reports (as shown in Figure 1.3) from LD search results. We develop a meta-model that characterizes the tabular reports shown in Figure 1.3. The semantics of instantiating this meta-model from LD search results is captured by a set of weighted rules which infer constituents of such tabular reports. We evaluate an MLN based approach that utilizes only syntactic evidence from triples. Then we explore the application of PSL to define a rule-base that utilizes three types of evidence: syntactic matches, domain ontologies, and user feedback.

In combining different sources of evidence, we aim at addressing the ensuing challenges of the task at hand as follows.

Challenge I: Lack of global schema Our approach assimilates exiting domain ontologies as a source of evidence. We use ontologies to provide explicit description of the resources in the search results. We adopt a *multiple ontologies* approach where data in the matched resources are potentially described by one or more ontologies. The reason for adopting such an approach is because a single domain ontologies is unlikely to provide sufficient coverage, as discussed in Section 4.2.3, for the results of a LD search.

Challenge II: Heterogeneities To address the heterogeneities of LD search results, we use SRL approaches to include rules that incorporate different matching functions to align between the matched resources. For example, as illustrated in Chapter 4, we defined rules, among others, that match between two RDF class labels based on the similarity of string the labels.

Challenge III: Determining User Intent To address the ambiguity of the search terms, our approach assimilate domain ontologies which provides background knowledge on the intended domain. Additionally, our approach includes user provides feedback as evidence for inferring the target tabular structure, which helps in resolving the ambiguity of the results.

The aim of the research presented in this dissertation is to devise and evaluate approaches and techniques to integrate LD search results. Our aim is to integrate the results stemming from a LD search by structuring RDF data in a tabular representation. The reason we target tabular reports is that they provide an intuitive view for users, who are used to tables., e.g., Web tables [Wang et al., 2012] and spreadsheets [Dix et al., 2016]. We also aimed at exploring the hypothesis that inferring a tabular structure that can be populated using LD search can be addressed by assimilating different sources of evidence. Specifically, we propose a model based on SRL approaches that assimilate syntactic evidence derived from matching algorithms, semantic evidence from ontologies, and evidence from users in the form of feedback.

To achieve the above aim, the following objectives are pursued:

O1 To identify, describe and evaluate approaches that allow us to infer, with uncertainty, a structure for LD search results.

The research contribution resulting from this objective is the characterization of the LD search integration task as a SRL problem. More specifically, we instantiate this characterization using MLN and PSL.

O2 To extend the approaches in **O1** to take advantage of the knowledge encoded by the domain ontologies that are used to describe LD sources with a view to improving the structure inferred in **O1**.

O3 To explore the impact of incorporating user feedback as an additional source of evidence to the approaches explored in **O1** and **O2**.

The contributions stemming from objectives **O2** and **O3** are:

- A methodology for incorporating evidence extracted from domain ontologies, in the case of **O2**, and obtained through interaction with the end user, in the case of **O3**.
- An empirical evaluation of this evidence-based approach, in which it is

shown how the principled, uniform use of different types of evidence improves the integration quality for the end user.

O4 To investigate how the results from **O1** to **O3** can be used to underpin a user interface to LD search that helps users identify the data that is most relevant to them.

The main contribution resulting from this objective is a proposal for a user interface that is driven by inference results from our PSL model. The interface provides the means by which a user can provide feedback that improves integration quality in a *pay-as-you-go* style.

The reason for choosing MLN and PSL for the sought objectives is twofold. First, MLN and PSL are applied, among others, to data integration problems such as entity resolution (e.g., [Singla and Domingos, 2006; Xu et al., 2013]), ontology matching (e.g, [Niepert et al., 2011; Bröcheler et al., 2010]) data extraction (e.g., [Satpal et al., 2011; Niu et al., 2012]) and data cleaning (e.g., [Pujara et al., 2013]). Second, compared to other SRL approaches, e.g., Relational Markov Networks [Getoor and Taskar, 2007], MLN and PSL are supported by mature implementations[7] [8] which makes them readily accessible when compared to other SRL approaches.

1.5 Published Work

The work presented in this thesis is supported by two workshop and conference publications:

Chapter 3: An MLN for interpreting LD search results.

- [Alshukaili et al., 2015] Alshukaili, D., Fernandes, A. A. A., and Paton, N. W. (2015). Interpreting linked data search results using markov logic. In Fischer, P. M., Alonso, G., Arenas, M., and Geerts, F., editors, *Proceedings of*

[7]MLN is implemented by Alchemy system: `alchemy.cs.washington.edu/`

[8]PSL implementation is found at `psl.umiacs.umd.edu/`

the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015., volume 1330 of CEUR Workshop Proceedings, pages 221228. CEUR-WS.org.

Chapters 4-5: Structuring LD search results using PSL.

- [Alshukaili et al., 2016] Alshukaili, D., Fernandes, A. A. A., and Paton, N. W. (2016). Structuring linked data search results using probabilistic soft logic. In Groth, P. T., Simperl, E., Gray, A. J. G., Sabou, M., Krötzsch, M., Lécué, F., Flöck, F., and Gil, Y., editors, *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I, volume 9981 of Lecture Notes in Computer Science*, pages 319.

1.6 Thesis Structure

The remainder of this thesis is structured as follows. In Chapter 2 we introduce preliminary concepts that pertain to Semantic Web technologies and PSL. In this chapter, we also survey a number of LD search engines that have been developed by the research community. The aim of this chapter is to provide a general background for the research described in this thesis. In Chapter 3, we introduce our MLN approach for inferring a structure from LD search results. We present a baseline model that infer elements of the sought tabular reports using syntactic evidence alone. Additionally, we present a set of experiments that show how the parameters of the model are learned and evaluate the model on a dataset that were obtained from LD searches. In Chapter 4, we extend our baseline model from Chapter 3 and adopt PSL as an SRL approach to incorporate evidence from ontologies and from user feedback. We then present an experimental evaluation where the results show that a principled approach to assimilating different types of evidence leads to improved integration quality. In Chapter 5, we present a prototype user interface that uses the inference results of the PSL model to provide useful results for the user. In this chapter, we also discuss various interaction scenarios as to how the different kinds of evidence are obtained and they are used to produce data tables without the redundancies one would normally expect in search results. Finally, in Chapter 6, we review the contributions of the research and discuss research issues that might be addressed in future work.

Chapter 2

Technical Context

In this chapter, we present the technical context and background required to understand the contributions presented in the thesis. First, Section 2.1 introduces terms and concepts relating to some of the core Semantic Web (SW) technologies that underpin the WoD. In Section 2.2, we survey approaches to LD search that return a list of ranked results where a given search term occurs. The work presented in this thesis aims to create tabular reports where content comes from results returned by LD search engines. We leave the introduction of the SRL approaches that we use in the inference of the target tabular structures to subsequent chapters.

2.1 Semantic Web Concepts

The Web was envisaged, from the start, as a decentralized platform for publishing and consuming information. The Web as proposed by Berners-Lee and Cailliau [1990] provided the infrastructure that allowed for sharing and linking of Web documents over the Internet. A document in the Web is assigned a unique identifier, a Uniform Resource Locator (URL), that provides the means of locating it. Accessing a document in the Web is done via the HyperText Transfer Protocol (HTTP). Traditionally, a document in the Web is encoded in the HyperText Markup Language (HTML). An HTML document consists of natural language, embedded images, and instructions for rendering it by a browser. One important feature of Web documents is that they contain hyperlinks to other documents that enable navigation to and discovery of additional related Web content. The Web allows for anyone to contribute content, which includes, e.g., data generated from relational databases through the use of server-side scripting languages (e.g., PHP). However, taking advantage of such data to provide answers to complex user questions can't be done easily in the Web.

To understand this shortcoming of the Web, consider finding an answer to the question *Who are the actors in The Godfather who were born in New York?* To answer this question on the Web, one would need to find sites with data about *The Godfather* actors. The most commonly used tool for this purpose would be a search engine. A typical search engine allows users to provide keyword terms in their underlying question and, in response, returns a list of links to documents that contain these terms. Even if we assume that the resulting links contain consistently relevant information, the data is likely to lack a consistent structure. Manual cleaning and structuring of the data is likely to be needed by the requester if the obtained data is to be consumed by down stream applications. Fundamentally, addressing such a question is likely to require significant effort by the requester, either a human user or an application.

Automating this data integration process is a hard problem. Much of the content on the Web is meant to be consumed by humans rather than software applications. This is because HTML was designed to provide rendering instructions but not semantic annotations of the content. Despite advances in natural language processing, machine learning and information retrieval (IR) techniques, and in spite of the volume of data available, it remains difficult for applications to make effective use of heterogeneous knowledge sources in the Web.

This is, in part, because most of the existing Web content is unstructured and/or heterogeneous in terms of format. In order to enable applications to integrate (i.e., semantically reconcile) heterogeneous data sources, two basic requirements need to be met: (i) data should be represented in a format that is amenable to processing, (ii) means should exist for resolving resource identity, so that referencing and dereferencing allows relationships between data to be represented and explored.

The Semantic Web envisions an ecosystem where data rather than documents, are considered as first-class citizens [Berners-Lee et al., 2001]. To facilitate the realisation of this vision, the *World Wide Web Consortium* (W3C) has published a number of standards. Among these standards are the Resource Description Framework (RDF) [Cyganiak et al., 2014], a data model for representing structured machine-processable data in the Web, RDF Schema (RDFS) [Guha and Brickley, 2014], and the Web Ontology Language (OWL) [McGuinness and Van Harmelen, 2004; Herman et al., 2004], a formal knowledge representation languages that can be used to provide a conceptual description of data. The sections that follow introduce these standards.

2.1.1 Resource Description Framework

The Resource Description Framework (RDF) is a standard for modelling and sharing data on the Web [Manola et al., 2004], i.e., a general framework for representing information in and about distributed data resources. RDF is one of the foundational technologies underpinning the WoD. One of its main goals was to be flexible enough to represent data originally represented in other data models (e.g. relational). As such, RDF fosters interoperability among applications with respect to data model diversity. In theory, RDF makes it easier to integrate data from multiple sources. In this section, we introduce RDF, and in Section 2.1.2 we discuss how RDF Schema and OWL are used to extend RDF with ontological modelling constructs. To facilitate the presentation, we use the running example in Figure 2.1. The example show a few RDF resources found in the results of a search for the terms `Godfather` and `actors`. The rounded boxes with a continuous border illustrate RDF *instance-level*

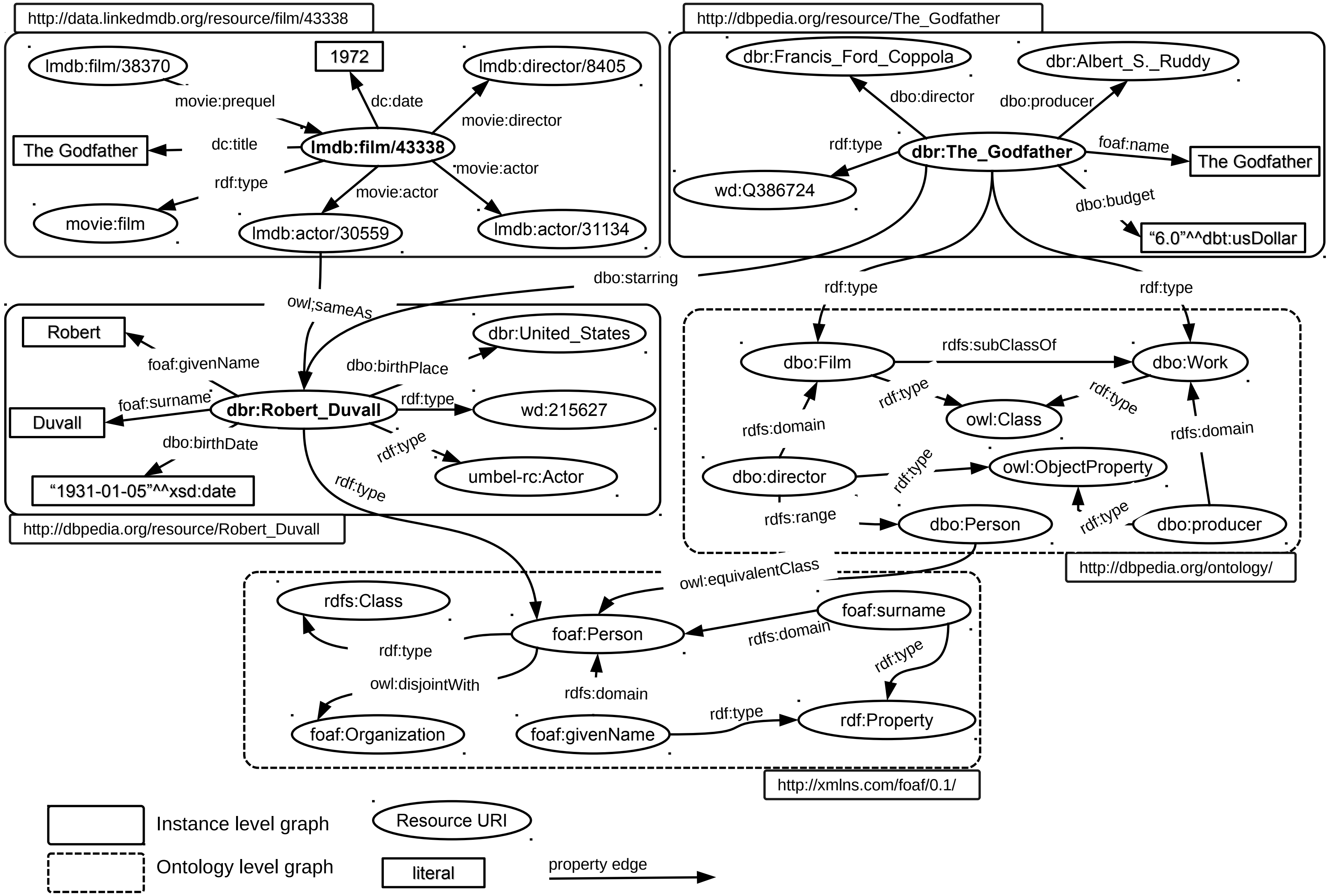


Figure 2.1: A RDF Sub-graph for some of the results returned for the search terms `Godfather actors`

data, which is obtained by dereferencing the URIs returned in the search results. The rounded boxes with a dotted border are *terminological-level* resources that define the terms used at the instance-level. In this example, we only show the subset of these resources required for the following discussion.

RDF Terms

In RDF, resources are described using *RDF terms*, as formalized in Definition 1.

Definition 1. The set of RDF terms is the union of the set of all URIs (denoted by $\mathcal{U}$), the set of all literals (denoted by $\mathcal{L}$), and the set of all blank nodes (denoted by $\mathcal{B}$), where $\mathcal{U} \cap \mathcal{L} \cap \mathcal{B} = \emptyset$.

URIs. A URI is a unique global identifier to a resource in the WoD. For example, the URI `http://data.linkedmdb.org/resource/film/43338` identifies the 1972 American film called *The Godfather* in *LMDB*[1] (an online RDF database extracted from IMDB[2]). For convenience, URIs are sometimes abbreviated using Compact URIs syntax (CURIE)[3]. For example, if `PREFIX lmdb: <http://data.linkedmdb.org/resource>`, is a prefix for resources in the LMDB dataset, then the above URI can be written as `lmdb:/film/43338`, as shown in Figure 2.1. In addition to serving as identifiers for resources in the WoD, it is considered good practice to respond with RDF data when a URI is dereferenced using the HTTP protocol, so that links to related concepts or entities can be looked up. For example, dereferencing `lmdb:/film/43338` returns references to resources describing the type, the director, the cast of and the sequel to this film. Although it is recommended, as per the LD principles [Berners-Lee, 2006], that URIs should be dereferenceable using the HTTP protocol, URIs are not always dereferenceable. One study [Hogan et al., 2010] has shown that over 8% of URIs cannot be dereferenced,

[1]data.linkedmdb.org/
[2]http://www.imdb.com/
[3]www.w3.org/TR/curie/

whereas, according to the same study, 34.8% of URIs do not return RDF data when dereferenced. Also, as common in Web environments [Madhavan et al., 2007], there are no grantees over the content of the RDF graph provided by Web servers. So a Web server may not return the same RDF graph in response to a requested resource.

Literals. The set $\mathcal{L}$, is the set of literals that are RDF terms. Literals are terms that are represented as a sequence of characters (i.e. strings). Literals can be either plain or typed. Plain literals are strings, such as `"The Godfather"`, potentially annotated with a language tag such as `"The Godfather"@en`. Typed literals are strings annotated with a datatype. For example, in Figure 2.1, the actor identified by the URI `dbr:Robert_Duvall` was born on `"1931-01-05"^^xsd:date`. Datatypes themselves are identified by URIs. In this case, `xsd:date` is a data type whose elements denote dates, as defined in XML Schema W3C standard.

Blank Nodes. The set $\mathcal{B}$, denotes *blank nodes* (or *bnodes*). A Blank node is an identifier with local scope, i.e., one restricted to the RDF document in which it occurs. Thus, a blank node cannot be referenced from outside that document. In practice, blank nodes are used to model composite attributes of a resource, or for defining a new resource without creating a global identifier for that same purpose. However, as argued by Heath and Bizer [2011], the use of blank nodes is not recommended in the context of LD. In fact, in the search results that we used in our experiments in Chapters 3 and 4, less than 7% of the resources are blank nodes.

RDF Triples and Graphs

An RDF model contains assertions about the properties of resources. A resource is anything we want to express some knowledge about (e.g., a Web resource, a real-world entity, an abstract concept, etc.). In RDF, data about resources is modelled in relation with other resources. These relations form *triples.* A triple is a statement consisting of: *a subject*, *a predicate*, and *an object*, as formalized in Definition 2. An

```
# prefix declaration
@prefix dbr: <http://dbpedia.org/resource/>.
@prefix dbo: <http://dbpedia.org/ontology/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix wd: <http://www.wikidata.org/entity/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

# triples
dbr:The_Godfather rdf:type dbo:Film,
                           dbo:Work, wd:Q386724 .
dbr:The_Godfather foaf:name "The Godfather" .
dbr:The_Godfather dbo:budget "6.0"^^dbt:usDollar .
dbr:The_Godfather dbo:producer dbr:Albert_S._Ruddy
dbr:The_Godfather dbo:director dbr:Francis_Ford_Coppola .
```

Listing 2.1: Triples describing the resource `dbr:The_Godfather`. These are depicted as a directed graph in Fig. 2.1

RDF triple takes on value from three disjoint sets $\mathcal{U}$, $\mathcal{B}$ and $\mathcal{L}$.

Definition 2. An RDF triple t is given by a 3-tuple relation $t = (s, p, o)$ where $s \in \mathcal{U} \cup \mathcal{B}$ is called the *subject*, $p \in \mathcal{U}$ is called the *predicate*, and $o \in \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ is called the *object*.

Generally, a subject denotes the entity of interest that is being described, a predicate denotes a property chosen to describe that entity, with the value of that property being denoted by the object. For example, the statement "The Godfather director is Francis Ford Coppola" is represented with the last triple in Listing 2.1.

A set of RDF triples $\mathcal{G}$ is known formally as an RDF graph such that $\mathcal{G} \subset (\mathcal{UB} \times \mathcal{U} \times \mathcal{UBL})$. By convention, URIs are depicted as ellipses and literals are as rectangles. It is common to conceptualize a set of triples as a direct labeled graph where the vertices represent subjects and objects, and the labeled edges represent predicates. Listing 2.1 shows a set of RDF statements that describe the resource `dbr:The_Godfather`. These statements are depicted as the graph in Figure 2.1. There are several serialization formats that support the publication of RDF graphs, including, RDF/XML [Gandon and Schreiber, 2014], Turtle [Beckett et al., 2014], and JSON-LD [Sporny et al., 2014]. In this dissertation, listings of RDF graphs are encoded using the Turtle notation.

2.1.2 Vocabularies

The notions of RDF triples and RDF graphs are core for the creation of instance-level Web resources, through the assertion of facts about URI-identified resources. In addition, the core RDF *namespace*[4] provides a set of vocabulary terms with standards-defined semantics that are used to describe predicates and assign resources to domain-specific classes.

The most popular term in the standard RDF vocabulary is `rdf:type`, which can be used to assert the membership of a resource to one or more classes. For example, in Figure 2.1, the resource `dbr:The_Godfather` is asserted to be a member of the of the classes `dbo:Film` and `dbo:Work`, i.e., *film* and *creative work* as defined in the *DBpedia ontology*[5]. At the vocabulary level, both `dbo:Film` and `dbo:Work` are defined to be instances of `owl:Class`, the class of all OWL classes.

Furthermore, the standard RDF vocabulary includes `rdf:Property`, an RDF class that is used to represent the set of all properties (i.e., URI terms that appear in the predicate position). To provide an example, as shown in Figure 2.1, the predicate `foaf:surname` is defined as a property in the FOAF[6] vocabulary.

The semantics of classes and properties can be defined in *ontologies.* Ontologies are computational artefacts that characterize partial knowledge about the world. An ontology describes a domain in terms of *classes* (denoting concepts) and *properties* of classes (denoting relations) in the domain. Ontologies are useful, among other things, in data integration systems [Haase et al., 2013], improving Web search [Sieg et al., 2007] and enhancing linking and navigation of the Web [Bechhofer et al., 2008]. In the WoD, ontologies are used to define shared vocabularies that describe the various domains for which data is published. For example, the FOAF ontology describes persons and their activities in relation to other people and objects; the

[4]www.w3.org/1999/02/22-rdf-syntax-ns
[5]dbpedia.org/ontology/
[6]xmlns.com/foaf/0.1/

SWRC[7] ontology models relationships between entities found in research communities, such as people, organizations and publications.

Given that the standard RDF vocabulary is not expressive enough to define the semantics of classes and properties, two separate standards are used to add semantically annotated RDF terms: RDF Schema (RDFS) [Cyganiak et al., 2014] and Web Ontology Language (OWL) [Herman et al., 2004; McGuinness and Van Harmelen, 2004]. The following sub-sections provide a brief overview of these standards, for which Definitions 3 to 5, below, are needed.

Definition 3 (RDF Class)**.** Given a triple $t = (s, p, o)$, we refer to an *RDF class* as a term that appears either in the o position of t when p is `rdf:type`, or in the s position of t when p is `rdf:type` and o is `rdfs:Class` or `owl:Class`.

Definition 4 (RDF Property)**.** Given a triple $t = (s, p, o)$, we refer to an *RDF property* as a term that appears either in the p position of t, or in the in the s position of t when p is `rdf:type` and o is either `rdf:Property` or one of its subproperties (e.g. `owl:ObjectProperty`, or `owl:DatatypeProperty`).

Definition 5 (Meta Class)**.** A meta-class is an RDF class whose members are themselves are either classes or properties. These classes are defined in the RDFS and OWL standards. Example of such classes include `rdfs:Class`, `owl:Class`, `rdf:Property`, `rdf:ObjectProperty`, etc.

RDFS

As mentioned earlier, the standard RDF vocabulary can express class membership of data resources. However, it lacks the means for expressing the semantics of the terms used in describing the resources. RDFS allows the creation of new classes through the instantiation of the `rdfs:Class` *meta-class*. Also, RDFS extends RDF with terms that allow the specification of relationships between classes and properties. The constructs `rdfs:subClassOf` and `rdfs:subPropertyOf` are used to

[7]ontoware.org/swrc/

model class and property hierarchies. Furthermore, RDFS provides the means to associate a class to a property via `rdfs:domain` and `rdfs:range` constructs. With `rdfs:domain`, one can state that the subject of the relation for a given property is a member of a specific class. Analogously, `rdfs:range` states that the object of the relation for a given property is a member of a specific class. To exemplify, the domain of `foaf:surname` is the RDF class `foaf:Person` (see Figure 2.1). Thus, according to the RDF semantics [J. Hayes and F. Patel-Schneider, 2014], this entails that the subject of a triple that contains `foaf:surname` as predicate is a member of the class `foaf:Person`. This is an example of the notion of entailments that follow from an RDF graph. In particular, entailments are conclusions that can be deduced as logical consequences based on assertions in a given RDF graph. In the Semantic Web, further enable applications to make use of knowledge in the Web by provide the foundation of machine readable knowledge.

OWL

The expressiveness of RDFS is deliberately limited to defining classes and asserting basic relations between classes and properties. OWL extends RDFS with more expressive means to allow for the assertion of relations. with more complex semantics. For example, OWL allows for expressing semantic equivalence. At the instance level, semantic equivalence between resources is modelled by the `owl:sameAs` construct; at the vocabulary level is it is modelled by `owl:equivalentClass` construct for classes and by the `owl:equivalentProperty` construct for properties. OWL further allows for asserting that two classes have no individuals in common using the `owl:disjointWith` construct (e.g. `foaf:Person` and `foaf:Organization` in Fig. 2.1), and for the specialization of RDF properties using the `owl:ObjectProperty` and `owl:DatatypeProperty` meta-classes. An instance of `owl:ObjectProperty` is a property that relates an individual to another individual where both individuals are resources. On the other hand, an instance of `owl:DatatypeProperty` is a property that relates an individual to a literal. OWL comes with a set of profiles that differ in terms of expressiveness. The differences in expressiveness give the domain modeller the choice of balancing between usability and the computational efficiency with which reasoning tasks can be carried out over ontologies. A full account of OWL, OWL profiles and features is beyond the scope of this thesis. The interested reader is refereed to [Herman et al., 2004; McGuinness and Van Harmelen, 2004].

2.1.3 RDF Publishing and Linked Data

The previous sections in this chapter have described the W3C standards that enable stake holders to represent data and knowledge in the Web. These standards neither mandate nor induce any principles to govern the publication of data. In this respect we note cases such as those of OpenCyc[8], UniProt[9] and WordNet RDF[10] in which large amounts of RDF data gave a rise to data silos in spite of their adoption of these standards [Hogan, 2014]. This is was, in part, due to the lack of interlinking between datasets and to the sometimes unwieldy nature of the resource formats used (e.g., compressed dumps). Such features in such initial efforts hindered the desired cost-effectiveness of access to early RDF data resources.

Berners-Lee [2006] proposed a set of principles for publishing data in the Web. Specifically, these principles advocate the use of dereferenceable HTTP URIs for naming things, and encourage the inclusion of external URIs in descriptions of data in order to foster interoperability with existing data on the Web. These principles were formulated as follows [Berners-Lee, 2006]:

- Use URIs to identify things in the Web.

- Use dereferenceable URIs so things can be looked up using HTTP.

[8]sw.opencyc.org/

[9]www.uniprot.org/

[10]http://xmlns.com/2001/08/wordnet/

- When a URI is looked up, return useful information using standards, and RDF in particular.
- Include links to other resources so that new information can be obtained by further dereferencing.

Adoption of these principles gave rise to a community initiative to seed the Web of Data called the Linking Open Data (LOD) project. The goals of the project were (*i*) to identify open licensed data sets, and (*ii*) to publish these datasets using the above principles. This effort was mainly led by researchers and developers in university labs. Soon after, media corporations (e.g. BBC[11]), governments (e.g. US[12] and UK[13]) and other organizations across diverse domains started to publish LD.

The adoption of LD by major Web service providers such as Google and Facebook has given importance to the initiative. Google, for example, imports LD embedded in authoritative[14] web sites into its Knowledge Graph[15], and uses the data to provide exact answers to user searches. The Facebook *like* button[16], which allows a Facebook user to express interest on an item is powered by RDFa[17]. RDFa is an RDF serialization format that allows for embedding structured data on Web pages. RDFa, HMTL microdata[18] and microformats are some of the technologies available with which websites with can be enriched with LD. The practice of embedding LD in HTML Web pages has been growing, as a recent study suggests [Guha et al., 2016]. The key drivers for such adoption are: firstly, the increasing interest from major Web search tools in exploiting embedded LD to improve indexing and retrieval algorithms [Baeza-Yates and Raghavan, 2010], and secondly, the increased

[11]www.bbc.co.uk/programmes

[12]www.data.gov

[13]www.data.gov.uk

[14]Authoritative in the sense that it can be trusted on providing information about a particular topic. Each search engine has different criteria for classifying a site as Authoritative

[15]http://www.google.com/intl/es419/insidesearch/features/search/knowledge.html

[16]https://developers.facebook.com/docs/plugins/like-button

[17]www.w3.org/TR/xhtml-rdfa-primer/

[18]www.w3.org/TR/microdata/

support by Web authoring tools (such as Wordpress[19] and Durpal[20]) for the use of semantic annotations in HTML. In essence, LD lowers barriers to the publication of data in the Web. It allows publishers to release data that live in proprietary containers, such as relational databases and spreadsheets. However, LD, in itself, does not provide an answer to the data quality issues faced by distributed data management systems. The fact that the LD initiative fosters a bottom-up approach for building a Web-scale database makes LD more prone to suffering from data quality issues such as accuracy, heterogeneity, and lack of schema-level definitions. An example of accuracy issues exhibited in the WoD can be seen in the descriptions of the 1942 US-produced film *Casablanca* shown in Figure 2.2. In here, accuracy

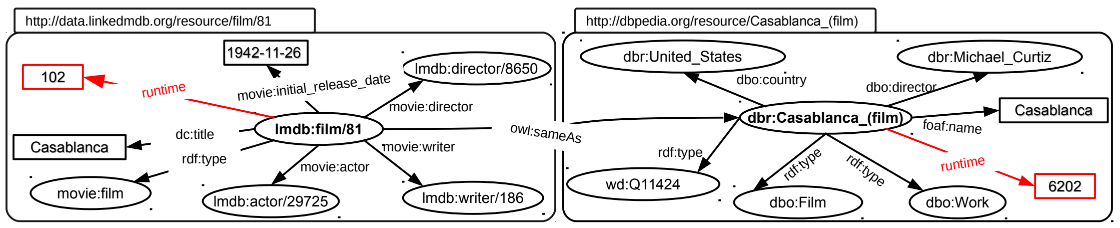


Figure 2.2: Two descriptions of the 1942 US-produced film Casablanca

conflicts are evident in the differences in the values of the `runtime` attributes of the two resources. In one resource (`lmdb:film/81`), the value of `runtime` is given in minutes (i.e., 102), whereas, in the other, i.e., `dbr:Casablanca_(film)`, it is given in seconds (i.e., 6202). Aligning these resources based on the values runtime attribute requires the reconciliation of the difference in expressing the unit of values. LD heterogeneity arises because of the diversity of publishers and the lack of coordination in choosing (a) identifiers for objects, and (b) schema-level concepts for describing the data [Hogan et al., 2010]. Furthermore, while it is recommended for data publishers to publish the ontologies that define the terms used in their RDF datasets, neither the LD principles nor the standards make it mandatory. One study has revealed that 15% of triples use undefined properties, and about 9% of

[19]wordpress.com/
[20]www.drupal.org/

`rdf:type` triples mention undefined classes [Hogan et al., 2010]. For example, in Figure 2.1, the class `movie:film` and the property `movie:actor` are not explicitly defined in any ontology. In addition to that, LD datasets need not to conform to the constraints in an ontology. For example, while `dbo:producer` is defined as a property of `dbo:Work`, as shown in Figure 2.1, there is nothing to prevent a data publisher from publishing a resource of the `dbo:Work` type that does not use the `dbo:producer` property. Indeed, publishers have the flexibility to choose terms from any set of vocabularies for their modelling task. Chapters 3 and 4 describe our approach to mitigate heterogeneity and the schema-less nature of LD resources through incremental systematic assimilation of evidence targeted at the integration of LD search results.

2.2 LD Search

As was mentioned in Chapter 1, this thesis investigates the application of SRL techniques for integrating results returned by LD searches. To provide the reader with an overview of the various LD search approaches, this section surveys LD search engines. First, we describe a classification framework. We then describe some of the existing search engines as instances of this framework resulting in the characterization displayed in Table 2.1.

2.2.1 Classification Framework

As the number of LD sources published in the Web grew in size and number, a clear need arose for developing LD search engines that crawl and index LD sources in order to provide search capabilities over the LD cloud. Figure 2.3 shows the general architecture of LD search engines. As with their Web counterparts, LD search engines have an indexing component and a query component. The indexing process builds the structures that enable the search, and the query process uses those

Dimension	**Swoogle [Ding et al., 2004]**	**Watson [D'Aquin and Motta, 2011]**	**Falcons [Cheng and Qu, 2009]**	**Sindice [Oren et al., 2008]**	**SWSE [Harth et al., 2012]**
Data Acuisition					
Seed list input	Manual Web search	Manual User Submitted Web and WoD search	Manual User Submitted Web and WoD search	Manual User Submitted	Manual
Source type	RDF	RDF	RDF	RDF SPARQL endpoints embedded semantic markup	RDF
Transformation					
type of terms	class URIs literal words	class URIs literal words instance URIs property labels	class URIs literal words instance URIs	class URIs literal words instance URIs property labels property-value pairs	literal words triples
term mapping	term-document	term-document	term-entity	term-document	term-entity
flow	direct	direct	indirect	direct	
reasoning	-	-	class inclusion	inverse function property based on OWL 'ter Horst' fragment [Kiryakov et al., 2005]	rule based approach over OWL fragment [Hogan et al., 2009]
Indexing					
structure	inverted index	inverted index	inverted index	an inverted index for each type of terms	inverted index sparse index
methods	proprietary	third party (Apache Lucene)	third party (Apache Lucene)	third party (Apache Lucene)	third party (Apache Lucene)
User Interaction					
query interface	keywords	keywords URIs	keyword URIs	keywords URIs structured	keywords structured
query refinment	-	filters on literals, class labels, and property terms	class inclusion hierarchy	filters on class labels, properties, type of source, and PLD	class inclusion hierarchy
user type	human applications via APIs	human applications via APIs	human	human applications via APIs	human
Ranking					
metrics	OntoRank	relevance based	combined relevance and popularity metric	PageRank for LD	source and resource ranks

Table 2.1: A classification of LD search approaches based on LD search engines' functional components

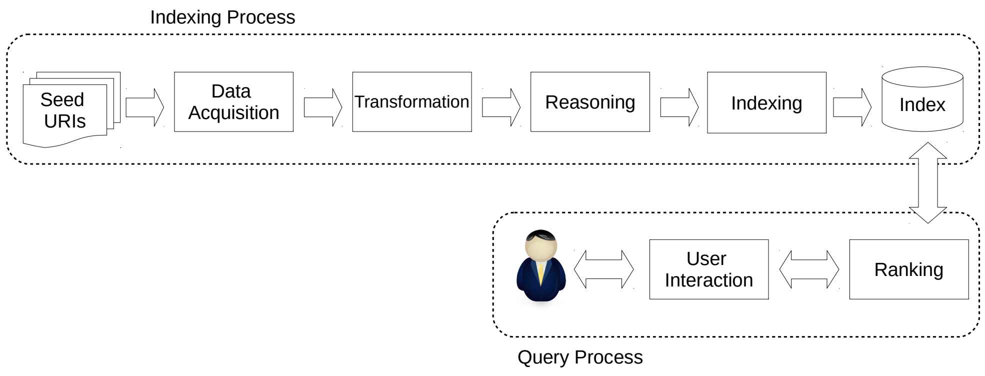
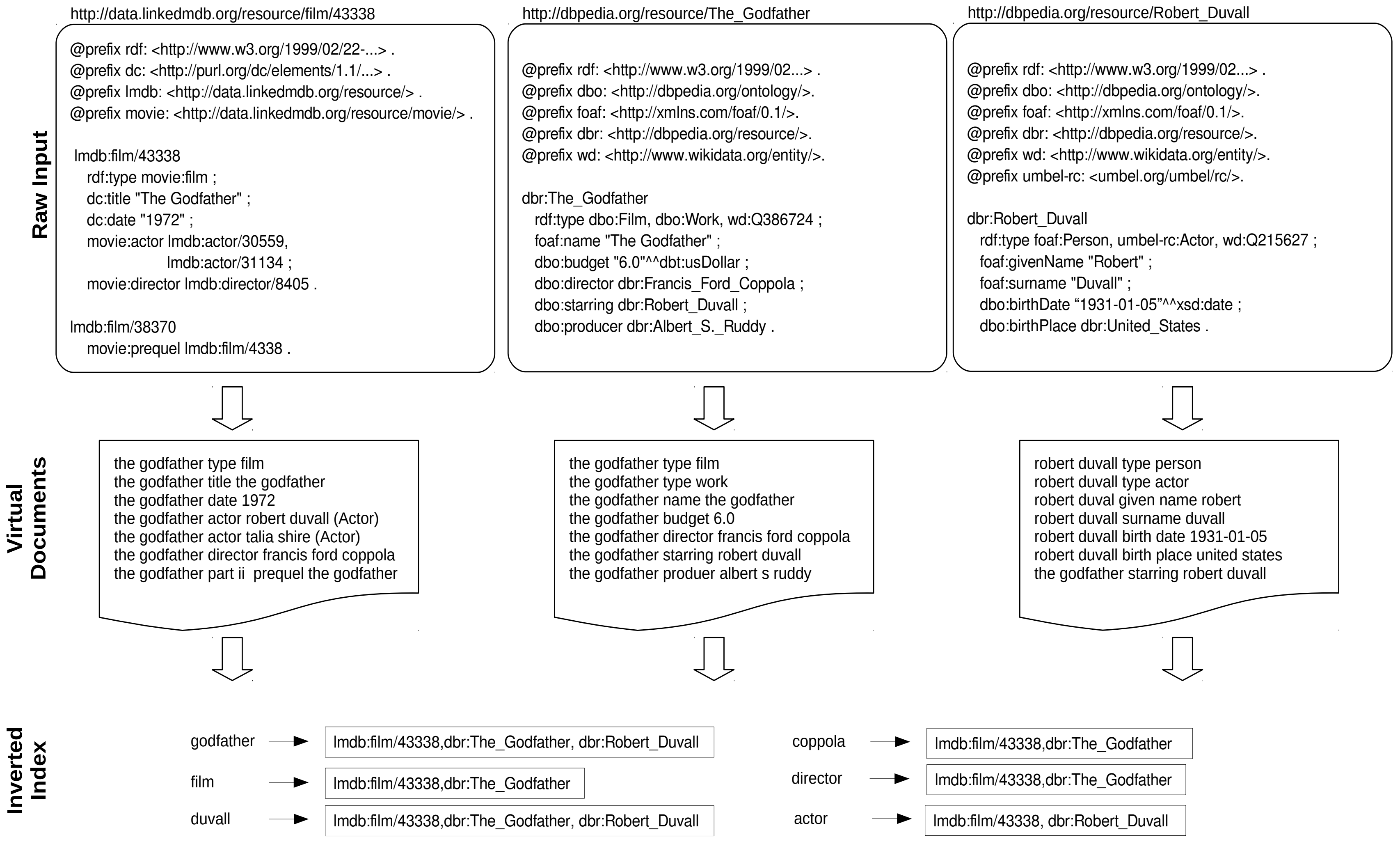


Figure 2.3: A general architecture for LD search engines (adopted from Croft et al. [2009])

structures to provide the user with a ranked list of search results. The classification framework presented here is based on the process illustrated in Figure 2.3.

Data Acquisition LD search engines implement various source identification methods to seed the crawling process by means of which RDF documents are retrieved for the purpose of building an index. The initial set of sources to be indexed can be obtained from different inputs, e.g., by manual curation of seed URIs, by using the results of existing LD and Web search engines, or using user-submitted URIs. There are also various sources from which data can be acquired, e.g., RDF dumps of datasets, or SPARQL endpoints, or semantic markup (e.g., JSON-LD, Microdata, and RDFa) embedded in HTML.

Transformation The transformation component (see Figure 2.4 for examples of common transformations used by LD search engines) derives search terms from sets of triples. The simplest index terms used are words mentioned in literal values. Literal values are often preprocessed using techniques such as case normalization, stemming and stop-words removal [Ding et al., 2004; Cheng and Qu, 2009; Harth et al., 2012]. Vocabulary terms (i.e., *RDF predicates and classes*) are often replaced by their labels in ontologies if one exists (e.g., as done is SWSE [Harth


Raw Input
Virtual Documents
Inverted Index
http://data.linkedmdb.org/resource/film/43338
http://dbpedia.org/resource/The_Godfather
http://dbpedia.org/resource/Robert_Duvall

```
@prefix rdf: <http://www.w3.org/1999/02/22-...> .
@prefix dc: <http://purl.org/dc/elements/1.1/...> .
@prefix lmdb: <http://data.linkedmdb.org/resource/> .
@prefix movie: <http://data.linkedmdb.org/resource/movie/> .

 lmdb:film/43338
   rdf:type movie:film ;
   dc:title "The Godfather" ;
   dc:date "1972" ;
   movie:actor lmdb:actor/30559,
                  lmdb:actor/31134 ;
   movie:director lmdb:director/8405 .

lmdb:film/38370
   movie:prequel lmdb:film/4338 .
```

```
@prefix rdf: <http://www.w3.org/1999/02...> .
@prefix dbo: <http://dbpedia.org/ontology/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix dbr: <http://dbpedia.org/resource/>.
@prefix wd: <http://www.wikidata.org/entity/>.

dbr:The_Godfather
   rdf:type dbo:Film, dbo:Work, wd:Q386724 ;
   foaf:name "The Godfather" ;
   dbo:budget "6.0"^^dbt:usDollar ;
   dbo:director dbr:Francis_Ford_Coppola ;
   dbo:starring dbr:Robert_Duvall ;
   dbo:producer dbr:Albert_S._Ruddy .
```

```
@prefix rdf: <http://www.w3.org/1999/02...> .
@prefix dbo: <http://dbpedia.org/ontology/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix dbr: <http://dbpedia.org/resource/>.
@prefix wd: <http://www.wikidata.org/entity/>.
@prefix umbel-rc: <umbel.org/umbel/rc/>.

dbr:Robert_Duvall
   rdf:type foaf:Person, umbel-rc:Actor, wd:Q215627 ;
   foaf:givenName "Robert" ;
   foaf:surname "Duvall" ;
   dbo:birthDate "1931-01-05"^^xsd:date ;
   dbo:birthPlace dbr:United_States .
```

the godfather type film
the godfather title the godfather
the godfather date 1972
the godfather actor robert duvall (Actor)
the godfather actor talia shire (Actor)
the godfather director francis ford coppola
the godfather part ii prequel the godfather

the godfather type film
the godfather type work
the godfather name the godfather
the godfather budget 6.0
the godfather director francis ford coppola
the godfather starring robert duvall
the godfather produer albert s ruddy

robert duvall type person
robert duvall type actor
robert duval given name robert
robert duvall surname duvall
robert duvall birth date 1931-01-05
robert duvall birth place united states
the godfather starring robert duvall

godfather →	lmdb:film/43338,dbr:The_Godfather, dbr:Robert_Duvall
film →	lmdb:film/43338,dbr:The_Godfather
duvall →	lmdb:film/43338,dbr:The_Godfather, dbr:Robert_Duvall
coppola →	lmdb:film/43338,dbr:The_Godfather
director →	lmdb:film/43338,dbr:The_Godfather
actor →	lmdb:film/43338, dbr:Robert_Duvall

Figure 2.4: Transforming LD documents to index terms and the inverted index of selected terms.

et al., 2012]). The terms that are indexed vary depending on the type of index to be created. Recall that a resource URI is an identifier for a real-world entity that is described using RDF, whereas, a document URL is the address where the RDF triples are found. The most common types of indexes used are *literal indexes* and *URI indexes*. A literal index contains an entry for each term and lists the document URLs where the term occurs. A URI index contains an entry for each resource URI and lists the document URLs where the resource URI occurs. For example, in Sindice Oren et al. [2008], both URI and literal indexes, among other indexes, are constructed to support the search tasks. For example, in Figure 2.1, the document URL `http://data.linkedmdb.org/resource/film/43338` contains RDF triples where the resource URIs `lmdb:film/38370` and `lmdb:film/4338` occur in the subject position. Thus a URI index of this document will contain the entries `lmdb:film/38370` and `lmdb:film/4338`, each of which list the document URL `http://data.linkedmdb.org/resource/film/43338`. *Entity-centric* search engines, such as SWSE [Harth et al., 2012] and Falcons [Cheng and Qu, 2009], map a term to a resource URI. On the other hand, *document-centric* search engines, such as Swoogle [Ding et al., 2004] and Sindice [Oren et al., 2008], map a term to a document URL. The transformation of a document into index terms can be either *direct* or *indirect*. In direct transformation, index terms are extracted from the retrieved documents and passed on to the indexing component. In indirect transformation, an intermediate representation is created from which terms are extracted. This indeterminate representation (e.g., a virtual document, as illustrated in Figure 2.4) often includes terms that do not directly occur in the source resource. Rather, they are obtained by dereferencing resources that occur in the object position of triples occurring in the indexed resource.

```
(:x rdf:type :C)                 ← (:P rdfs:domain :C) ∧ (:x :P :y)
(:y rdf:type :C)                 ← (:P rdfs:range :C) ∧ (:x :P :y)
(:y owl:sameAs :z)               ← (:x :P :y) ∧ (:x :P :z)
                                  ∧ (:P rdf:type owl:FunctionalProperty)
(:y owl:sameAs :x)               ← (:x owl:sameAs :y)
(:C owl:equivalentClass :D) ← (:C rdfs:subClassOf :D)
                                  ∧ (:D rdfs:subClassOf :C)
```

Listing 2.2: A example of ter-Horst rules using within reasoning component of some LD search engines

Reasoning Some LD search engines apply an additional reasoning step before indexing a given resource. The reasoning component allows for the consolidation of triples from external resources as well as the discovery of implicit knowledge about the indexed resource. OWL semantics can be used to automatically *union* triples found in different resources using, for example, `owl:sameAs` and the `owl:InverseFunctionalProperty` to identify URIs that potentially describe the same underlying real-world entity. For example, the SWSE [Harth et al., 2012] search engine uses `owl:sameAs` to consolidate instance-level resources. In this case, the resources `dbr:Robert_Duvall` and `lmdb:actor/30559` in our running example would both be assigned a canonical identifier before they are indexed. The `owl:InverseFunctionalProperty` (IFP) construct is capable of uniquely identifying resources. For example, if two resources `:x1` and `:x2` have the same value for a property `:P`, and `:P` is defined as an inverse functional property, then `:x1` and `:x2` are the same. More generally, RDFS and OWL descriptions can be used to infer implicit knowledge of LD resources, by the means of *inference rules* that match parts of the given RDF graph and imply conclusions that are logical consequences of the explicitly annotated knowledge. The set of rules most often used by LD search engines that implement a reasoning component are presented in [ter Horst, 2005a,b], which are based on RDFS and a fragment of the OWL semantics. Listing 2.2 shows a few examples of such rules.

Indexing The underlying index of the search engine can vary in terms of the structure used (e.g. inverted index, sparse index) and the methods and algorithms used to create the index (e.g. proprietary, third-party tools). Inverted indexes are used to support lookups based on RDF literals or URIs. On the other hand, sparse index structures are used to provide pattern based lookup such as *property-value* pairs and *triple* lookups [Harth et al., 2007].

User Interaction LD search tools often employ different query interfaces that allow the user to pose different types of queries. Different search engines allow the user to provide keyword, URI, or structured search terms. URI search terms allow the user to search for resources containing specific identifiers. For example, a user might express an interest in resources related to `The Godfather` film by using the URI `dbr:The_Godfather` in search engines that use URI indexing and lookup, such as Sindice [Oren et al., 2008]. Structured search terms allow the user to search using patterns. For example, the structured search term `(predicate=foaf:name, value="Godfather")` can be used to look up resources containing the keyword `Godfather` in the value of the RDF predicate `foaf:name`. LD search engines also provide different methods to allow the end user to refine the search results, e.g., filtering for more specific search results, or class hierarchies that enable the exploration based on specific RDF types. In addition to providing interfaces for human users, some LD search engines, such as Swoogle, also provide APIs through which LD applications can discover RDF documents that reference a certain URI or contain a certain keyword. These interfaces often provide the results in a machine-processable format (e.g., RDF).

Ranking The ranking component of an LD search engine provides a ranked list of candidate results for the user query. There are different metrics used by LD search engines to rank the results, such as relevance-based and popularity-based. Often, the relevance of a resource is computed from the similarity between the search term and the indexed resource. A commonly-used information retrieval (IR) similarity metric is the cosine similarity, which is often computed based on the term frequency–inverse document frequency (TF-IDF) weighting scheme [Manning et al., 2008]. Popularity-based metrics use variants of the PageRank [Page et al., 1999] algorithm. The general idea of these metrics is to consider incoming and outgoing links to compute the probability of someone stumbling on a given RDF resource. The higher the probability, the more popular the resource. Some LD search engines which rely on third-party tools (i.e., *Apache Lucene*[21]) for index creation often use the ranking metrics already employed by such tools.

2.2.2 LD Search Engines

We describe five LD search engines based on the framework introduced in Section 2.2.1. These were included because they are sufficiently well described and widely cited in the literature.

Swoogle [Ding et al., 2004] is one of the first LD search engines. It support keyword-based search over RDF-represented resources by means of inverted indexes. Swoogle is primarily focused on searching ontologies. Given search terms, Swoogle returns ontologies that mention the given terms. Swoogle ranks documents using OntoRank, which iteratively computes the ranking of a document based on references to class and property terms defined in other documents. As such, Swoogle adopts a document-centric approach for indexing RDFS and OWL vocabularies.

Unlike Swoogle, Falcons [Cheng and Qu, 2009] focuses on searching instance-level RDF data using an entity-centric search approach. Falcons uses crawling, parsing, organizing, ranking, and sorting as prior steps to providing its search capabilities. In order to extract terms from RDF documents, Falcons employs the notion of virtual documents. It essentially converts the triples in a given resource into a document comprising literals before indexing the virtual document to give rise to an inverted

[21]lucene.apache.org/

index. Falcons also includes a reasoning component focused on class inclusion and instance checking. The inferred class hierarchies are provided to users for filtering the search results based on type information. Falcons uses two metrics to rank matched documents. The first is a relevance-based metric, viz., the cosine similarity between the vector representing the search term and the vector representing the virtual document derived from the resource. The second is a popularity-based metric that ranks the importance of resource based on the logarithm of the number of RDF documents in which the resource is mentioned.

Watson [D'Aquin et al., 2007; Sabou et al., 2007; D'Aquin and Motta, 2011] is a document-centric search engine that provides search facilities over instance-level and ontology-level RDF documents. It is mainly focused on providing APIs to expose search services to software applications. Its capabilities include keyword search over RDF, searching for entities (i.e., classes, properties, and individuals), and searching for RDF documents containing a matching keyword in the literals. Watson employs a relevance-based metric that enables the ranking of documents containing the search terms.

Sindice is a LD search engine that acquires data not only through crawling the WoD, but also through SPARQL endpoints (i.e., a remote query interface to RDF datasets). Sindice has similar capabilities to Watson, with a focus on providing APIs for finding instance-level RDF documents based on the lookup of literals, URIs, and IFP-value pairs. Each type of lookup is supported by a separate inverted index. For building the IFP-value pair index, Sindice uses both explicitly-defined IFPs as well as RDFS and OWL reasoning over the ter Horst rules for discovering implicit IFPs based on sub-property relations and cardinality restrictions.

SWSE takes an entity-centric approach to LD search. It reconciles resources that describe the same real-world entity using `owl:sameAs` and IFP reasoning. In doing so, it creates a canonical URI to represent the reconciled resource. The canonical URI is chosen by ranking the merged resources using an approach that takes into consideration the popularity of the dataset where the resource resides. To enable the discovery of implicit information, it uses rule-based reasoning based on the ter-Horst rules. The SWSE user interface offers a number of facets that enable the user to filter the search results by type, as well as browsing capabilities over the results of the search.

In addition to LD search engines, Web search engines, such as Google and Bing, also index RDF data. The earliest type of semi-structured data to be exploited by Web search engines was HTML metatags, which allowed Web authors to provide natural language statements to specify descriptions, keywords, authorship, and other metadata about an HTML page. Since metatags failed to standardize the provision of metadata, other methods were developed to augment Web pages with structured data, such as microdata, microformats, and RDFa. These methods were used to publish metadata about people, products, events, organisations, recipes, events, etc. Initially, such metadata was used for ranking search results. Later, Web search engines used to augment the search results with mini-summaries. To give Web authors a wider selection of vocabularies, Microsoft, Google, Yahoo!, and Yandex championed the *schema.org*[22], which provides a collection of vocabularies that are supported by their search engines.

Although most of the LD search engines were developed as prototypes to address the intricacies of providing keyword searches over LD, the adoption of LD by mainstream search engines is testimony to its importance as a source of data that can potentially address complex information retrieval needs.

2.3 Summary and Discussion

This chapter has introduced some of the basic concepts related to the Semantic Web technologies used in publishing data on the Web. We have highlighted the LD principles, which advocate the adoption of Semantic Web standards as well as

[22]schema.org/

making the published data reusable by both human and software applications. LD advocacy has successfully elicited large amounts of data expressed in RDF formats from by public sector as well as private sources. We have also surveyed a number of LD search engines that were developed to address the need to provide users with access to this sort of data.

LD principles seek to do for data what the Web did for documents. In essence, they provide a guideline for publishing and consuming data in the Web using the RDF data model. However, LD does not address long-standing data management problems such as heterogeneity. Worse, it introduces new problems, such as the need to assign URIs to data in formats that are not easily consumable by humans.

While LD search engines relieve the user from interacting with LD sources through complex queries by providing a keyword search interface to the WoD, their results do not insulate the user from the inherent heterogeneities of the WoD. LD search engines allow the user to find RDF resources by matching search terms against indexes. In doing so, they only provide the user with a list of potentially relevant resources. A more appealing alternative would be to retrieve relevant data about the resources in the result, and provide the user with an integrated view of the underlying resources. Sig.ma [Tummarello et al., 2010] recognized this opportunity by providing a layer of processes that, given LD search results, constructs a report assuming that the results can be used to describe a single entity (see Chapter 3 for more details on Sig.ma). However, such an assumption does not always hold because a search for entity may well return other related entities too or a search may be for a collection of things, such as `London Hotels`. We saw in Chapter 1 that a search for the film called `The Godfather` also returns results about the cast involved in the film. Thus, the generated report must grapple with the fact that many entities of many types maybe described in a resource and hence can be can be returned as legitimate search results.

In Chapters 3 and 4 we describe our approach to integrating search results by inferring a structure that recognises the diversity and heterogeneity of LD search results using SRL techniques.

Chapter 3

Interpreting LD search results using MLNs

The adoption of LD principles by the Web community has resulted in the generation of large volumes of semantically-annotated RDF data. One of the challenges posed by such an amount of data is how to find and explore relevant information in the WoD. This an important requirement for facilitating the uptake of LD by applications that support both non-experts and expert users [Heath and Bizer, 2011]. LD search engines play an important role in providing access to data in this WoD. As briefly described in Chapter 2, Falcons [Cheng and Qu, 2009] and Sindice [Oren et al., 2008] are examples of search engines that use information retrieval (IR) techniques to provide users with a ranked list of results given search terms. However, to a greater extent than is the case for the Web, consuming LD can be time-consuming and cumbersome for ordinary users. While documents in the Web are often designed for human eyes, searching the WoD returns URIs for RDF resources and this makes them more difficult for humans explore. The difficulty stems from the fact that RDF was designed to represent the Web meta-data for algorithmic treatment inside software applications. In addition to this, as is the case in the Web, LD search engines only provide a best effort match to a search term. This often leads to results

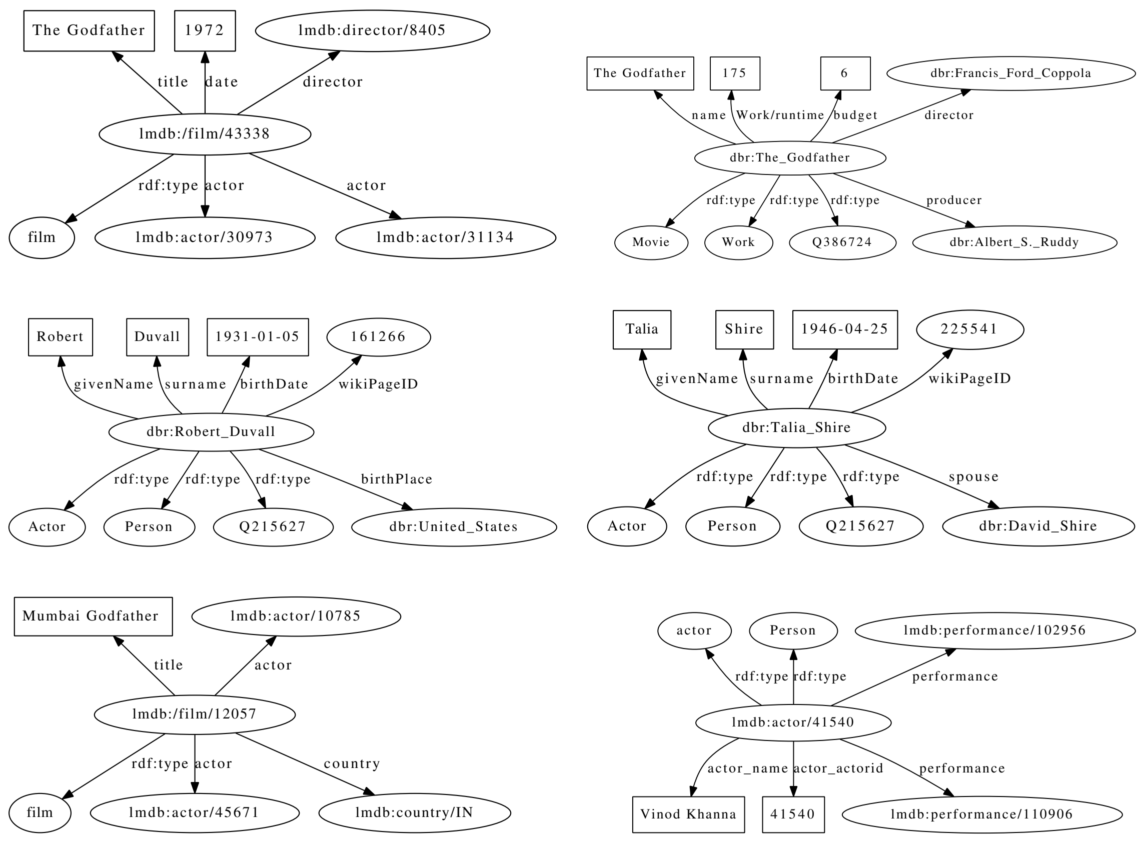


Figure 3.1: Example LD search results for the term "`Godfather actors`"

Movie				
name	**date**	**director**	**budget**	**runtime**
The Godfather	1972	Francis Ford Coppola	6	172

Actor				
givenName	**surname**	**birthDate**	**birthPlace**	**spouse**
Robert	Duvall	1931-01-05	United States	null
Talia	Shire	1946-04-25	null	David Shire

Figure 3.2: Results in Figure 3.1 integrated and reported as tables

that include many heterogeneous types. Furthermore, the results of a search may involve value of different types, which are interleaved in the results. Consider again the example we presented in Chapter 1. A search for `Godfather actors` using the Sindice search engine [Oren et al., 2008] returns results (as shown in Figure 3.1) that represent several distinct films as well as well as actors in those films. A manually created report (e.g., the one shown in Figure 3.2) over such a search might pull together the properties of the film name *The Godfather* from several resources into a heading and a list of properties, then might provide separate tables for the collections of the actors in this film about which information was retrieved.

Could such a report be generated automatically? The creation of such a report requires, at a minimum, identifying (i) the (real-world) entity types that occur in the results, (ii) the instances of the latter that occur in the results, (iii) the properties of each entity type, and (iv) the values assigned to the properties for every entity that occurs in the results.

Although this is a data integration problem, it is quite unlike classical data integration, in which typically there is a known target (or global) schema to which source data should be mapped, and there is some level of human engagement in the data integration. This raises the question as to approaches that might be suitable for interpreting and integrating search results.

In this Chapter we discuss an approach based on Markov Logic Networks (MLNs) in which we: (i) postulate a preliminary set of rules that capture relationships that exist within search results, which are expressed using logic; (ii) learn weights for these rules that represent their strengths as constraints; and (iii) use the resulting weighted rules over search results to infer (with uncertainty) the entity types and instances that are represented in the results. We are motivated to use Markov Logic as it provides a well founded approach to integrating evidence of different types to support conclusions that can inform data integration.

Markov logic has been applied to a range of tasks of relevance to data integration, including classification, entity resolution and knowledge-base construction, as we discuss in Section 3.1. In Section 3.2 we briefly introduce the relevant background on MLNs. Section 3.3 describes our approach to applying MLNs for the inference of tabular structures from LD search results, and presents an experimental evaluation of the approach. In Section 3.4, we draw some conclusions based on the results obtained in our experiment.

3.1 Related Work

In this section we discuss work that is relevant to the integration of LD search results, and specifically, by reviewing results on LD search, on data integration for LD and on Markov logic for data integration.

3.1.1 Linked Data Search

As previously noted, the increase in the amount of RDF data published in the Web has given rise to a number of proposals for performing keyword search over LD. Some of these, e.g., Swoogle [Ding et al., 2004], are designed to search at the *vocabulary-level* of the WoD. Others, e.g., Sindice [Oren et al., 2008] and Falcons [Cheng and Qu, 2009], focus on crawling, indexing, and ranking the *instance-level* of the WoD. Our approach is complementary to LD search engines insofar as we aim to build a coherent view of returned results.

We saw in Chapter 2 that instance-level LD search engines can be classified, in terms of how resources are indexed, as either *entity-centric* or *document-centric*. The difference between the two approaches is that a term is mapped to resource URIs in the former and to document URLs containing the RDF triples in the latter. In practice, irrespective of the adopted approach, performing a search essentially leads to results that often return heterogeneous resources of different types. Our proposed method is therefore agnostic with respect to the underlying indexing approach used by LD search engines, because our focus is on the inference of a tabular structure that integrates the results of the search.

Chapter 2 noted how limited forms of data consolidation are employed by LD search engines. For example, Falcons consolidates data about a resource by mapping it to a virtual document. This consolidation is based on a resource being mentioned in different data sources. For example, the resources URI `dbr:United_States` is mentioned as a property value of the resources `dbr:Robert_Duvall` and `dbr:Talia_Shire`. When the latter resources are converted into virtual document and an inverted index is created based on their content, the terms `united` and `states` both will map to these resources. Thus a search using the above terms is likely to return the resources `dbr:Robert_Duvall` and `dbr:Talia_Shire`. Some search engines, such as SWSE [Harth et al., 2012] and Sindice, consolidate resources using reasoning over `owl:sameAs` and `owl:InverseFunctionalProperty` properties, as these assert distinct URIs that potentially describe the same underlying real-world entity. However, such forms of consolidation fall short of a comprehensive approach to result integration. This is because consolidation approaches are often tailored towards improving the index construction as opposed to producing an integrated set of results.

Sig.ma [Tummarello et al., 2010] does address result integration in a more comprehensive manner. Sig.ma uses results returned by Sindice to build an aggregated view of the results in the form of an entity profile. Sig.ma uses a recursive approach to collect RDF data which refer to resource identifiers that match a search term. The first step collects the document URLs that contain the search terms. Subsequent steps search for sources containing resource URIs found in the results of the proceeding step. The collected RDF data is decomposed into chunks (called resource descriptions) that describe distinct entities and the former are ranked against the search term. Sig.ma collects additional data when it encounters an `owl:sameAs` predicate. The resulting resource descriptions are consolidated by combining the values of lexically-similar attributes. Hand-crafted rules, such as the removal of "has" from `hasTitle` or replacement of attributes that may share similar values such as "web page" and "homepage" with the term "Web page", are applied in the consolidation step. In addition, Sig.ma allows users to interact with the resulting entity profiles, either through navigating to other sources of information, or by refining the results in a coarse-grained manner: capabilities only allow users to reject or accept the sources that contribute data to the generated entity profiles. Sig.ma does not provide any means for resolving semantic heterogeneities in the data before attempting to construct the integrated view. For example, if a user is interested in information about *Manchester University*, Sig.ma combines data on the university, on Manchester Grammar School, and on a railway station. While it brings together lots of correct information, a great deal of incorrect data is often included. In this dissertation, we have developed a more principled approach with a view to resolving such heterogeneities through the identification of entities and their types using Statistical Relational Learning approaches.

3.1.2 Data Integration for Linked Data

Low barriers to publication, as well as a diversity of publishers without central coordination, can lead to LD being published with inconsistencies both at the conceptual and at the instance levels. At the conceptual level, this comes in the form of different conceptualizations of the same domain, inconsistencies in the structural representation of concepts in LD terminologies, etc. At the instance level, there may be many different resources that describe the same real-world entities, redundant information, and contradictory attribute values. There are many approaches to the problem of matching RDF sources. These can be divided into two broad categories: ontology matching and instance matching [Castano et al., 2011]. The goal of ontology matching is to align schema-level elements of RDF sources using information from the schema level, the instance level, or both [Euzenat and Shvaiko, 2013]. On the other hand, instance-level approaches try to reconcile the multiple of data resources that may describe a single real-world entity [Ferrara et al., 2011]. RDF instance matching tools such as Silk [Volz et al., 2009; Isele and Bizer, 2013], ObjectCoref [Hu et al., 2011], and LIMES [Ngomo and Auer, 2011] aim to generate links (e.g., `owl:sameAs` links) between instance level LD resources. In both ontology and instance matching every resource is considered to be a valid entity. This assumption cannot be made in our setting because of the underlying uncertainty as to whether a resource might contribute relevant data to the target tabular structure, and if so, precisely how. Ontology matching approaches do not take into account this uncertainty, in a systematic and principled way, therefore, one of our motivations for using an SRL approach is that SRL methods inherently do so through probabilistic reasoning.

Our work aims at creating a populated schema from LD search results. To achieve this, we have assumed a meta-structure to underlie the search results. In this regard, there have been various proposal for structure inference of RDF sources [Christodoulou et al., 2015b; Zhu et al., 2013; Zong et al., 2012; Völker and Niepert, 2011]. Such approaches take as an input a data graph and produce a structural summary that is homomorphic to the original data graph using techniques such as hierarchical clustering [Christodoulou et al., 2015b; Zong et al., 2012], association rule mining [Völker and Niepert, 2011], and inference using Bayesian Networks [Zhu et al., 2013]. However, these approaches are often evaluated on a specific dataset (e.g., Magnatune or DBpedia). This is different from inferring a structure for search results because the relevant sources in the results often vary in terms of the datasets they originate from. For example, searching for a film titled `Godfather` on Sindice would return results from at least three datasets, viz., DBpedia[1], YAGO[2], and LMDB[3]. This makes the structure inference problem harder as mappings between the datasets need to be inferred or incorporated as evidence.

In addition to research on linking RDF at the conceptual and instance levels, there has been research on mapping properties across RDF sources [Fu et al., 2012; Zhang et al., 2013; Gunaratna et al., 2013]. The aim of such approaches is to find similar [Fu et al., 2012] or equivalent [Zhang et al., 2013; Gunaratna et al., 2013] properties using statistical measures based on subject and object overlap. These approaches, as well as approaches that map between ontology concepts and instances

[1]dbpedia.org
[2]yago-knowledge.org
[3]data.linkedmdb.org/

across datasets, can be used as sources of evidence in our approach (see Chapter 4).

3.1.3 Markov Logic Networks for Data Integration

MLNs have been applied to data integration tasks such as entity resolution [Singla and Domingos, 2006], knowledge base construction [Niu et al., 2012] and ontology matching [Niepert et al., 2011]. Our approach complements these approaches by applying MLNs to the problem of inferring a structure from LD search results. Singla and Domingos [2006] described a domain specific MLN that performs entity resolutions on bibliographic entries. They used an MLN to encode rules that infer the similarity of publications based on the similarity of venues, authors and titles. In contrast with this approach, the rules we use in our model encode a domain-independent model.

One of the motivations for using MLNs for the task in hand is that we can use MLNs to encode different sources of evidence to fuse data coming from multiple (and possibly heterogeneous) sources, as done in Elementary [Niu et al., 2012]. Elementary builds a knowledge base about entities from semi-structured Web sources. It uses MLNs to discover co-referent entities using different kinds of evidence, such as evidence from named entity recognition (NER) tools, from domain knowledge, and from syntactic matching tools. In the context of LD, Niepert et al. [2011] exploited MLNs to match between concepts encoded in domain ontologies. Their approach combined evidence from input ontologies and lexical similarities in order to map between concepts in the ontologies. The input ontologies were encoded as hard constraints. Prior similarities between concept features, e.g., labels, were computed and used as evidence for concept similarities. The values of the final similarities between concepts were returned though an optimization procedure that corresponds to the MLN inference.

3.2 Background: Markov Logic Networks

The goal of machine learning is to develop algorithms that allow systems to acquire knowledge and improve their performance automatically from experience [Mitchell, 1997]. In classical machine learning settings, a typical assumption made is that entities in the data represent *independent and identically distributed (i.i.d.)* set of instances. In such settings, data is represented by a single table of feature vectors, one vector for each entity in the data.

In contrast, SRL focuses on inference and learning algorithms for modelling and reasoning in uncertain multi-relational domains. Multi-relational domains can be seen as consisting of tables that describe feature vectors of entities in the domain and tables that assert relationships among the entities. Thus, the i.i.d assumption is dropped in SRL approaches. Figure 3.3 shows a small example of multi-relational data describing voting behaviour. A typical feature of multi-relational domains is that they often exhibit a degree of uncertainty as to the existence of a relationship among entities.

SRL approaches address the challenge of learning from multi-relational data by combining ideas from two areas. First, some SRL approaches use expressive formalisms, such as *first-order logic* (FOL), to represent the domain structure, thereby capturing the dependencies among entities in the domain. Second, most SRL approaches adopt ideas from probabilistic graphical models, such as Bayesian or Markov networks, to deal with uncertainty

MLNs are an SRL approach that combines FOL and Markov networks to give rise to a unified representation for defining of probabilistic models. In what follows, we briefly introduce MLNs. We refer the reader to [Domingos and Lowd, 2009a] for further details.

Person			
id	**gunControl**	**gayRights**	**votesFor**
p1	no	yes	I
p2	no	no	R
p3	yes	yes	D
p4	yes	yes	D
p5	no	no	I

Spouse	
id1	**id2**
p1	p4
p5	p3

Friend	
id1	**id2**
p3	p4
p3	p1
p2	p1
p5	p2

Figure 3.3: Example database describing voting behaviour, with a table for persons and two relationships connecting persons, where I, D, and R stands for, rep., Independent, Democrat, and Republican

3.2.1 Representation

Definition 6. A Markov Logic Network is a set of pairs (F_i, w_i), where F_i is a first-order logic formula and w_i is a real value representing its weight. Given a set of constants in some domain, an MLN induces a *ground* Markov network where the nodes correspond to the ground atoms in formulae and an edge exists between two ground atoms if they appear together in at least one ground formula.

In an MLN, dependencies among relations are expressed using FOL formulae. An MLN formula expresses a rule of thumb that guides the inference process but it does not have to be true in all possible worlds. The weight attached to each rule is relative to the weights of other rules and correlates to the number of possible worlds in which a rule is satisfied. In other words, the weight determines the relative importance of the rule in the overall model. In an MLN, a formula with a negative weight w can be replaced with its negated formula with a weight of $-w$. A formula can also be assigned an infinite weight to indicate a constraint that should not be violated. Listing 3.1 shows an example MLN (adopted from [Kimmig et al., 2012]) that models the database shown in Figure 3.3. This MLN states that, given any individuals a, b and p, instantiating (respectively) the variables A, B and P, a claim is made that if b is either a friend or a spouse of a, and a votes for party p, then, with some likelihood, b votes for p. The respective weights assert that the influence of spouses on what party b votes for is larger than that of friends.

```
0.3: votesFor(B,P)   ← friend(B,A) ∧ votesFor(A,P)
0.8: votesFor(B,P)   ← spouse(B,A) ∧ votesFor(A,P)
```

Listing 3.1: A weighted MLN describing voting behaviour

An MLN is defined over a set of predicates. The predicates are categorized into *evidence* and *query* predicates. An evidence predicate is a predicate whose ground atoms have known truth values by observation. A query predicate is a predicate where one or more of its ground atoms have unknown truth values. In the above example, `friend` and `spouse` are evidence predicates whereas `votesFor` is the query predicate.

An MLN $\mathcal{F}$ that models some domain and with a set of finite constants $\mathcal{C}$ defines a Markov network $M_{\mathcal{F},\mathcal{C}}$ that models the joint distribution of a set of random variables $X = (X_1, X_2, .., X_n) \in \mathcal{X}$. Each variable of X is a ground atom and $\mathcal{X}$ is the set of possible worlds, that is the set of all possible truth value assignments of n binary variables.

The network $M_{\mathcal{F},\mathcal{C}}$ is constructed by adding a binary node for each ground atom given the set of rules $\mathcal{F}$ and constants $\mathcal{C}$. An edge is created between two nodes if the corresponding ground atoms appear together in at least one rule in $\mathcal{F}$. Figure 3.4 shows the structure of the Markov network induced by the MLN in Listing 3.1 and the constants a, b and p, where a and b represent persons, and p represent a party. Note, that the reason for showing the graph is to illustrate how the Markov network is induced from a rule-base given some constants. We intend to illustrate te outcome of the learning and inference algorithms not through this graph, but though subsequent examples.

Let $f_i \in \mathcal{F}$ be formula in the MLN associated with the weight w_i. Let $\mathcal{G}$ be the set of all ground rules created by grounding formulas in $\mathcal{F}$ using constants $\mathcal{C}$. Also, let $\mathcal{G}_{f_i}$ be the set of all possible grounding for formula $f_i \in \mathcal{F}$ with constants in the

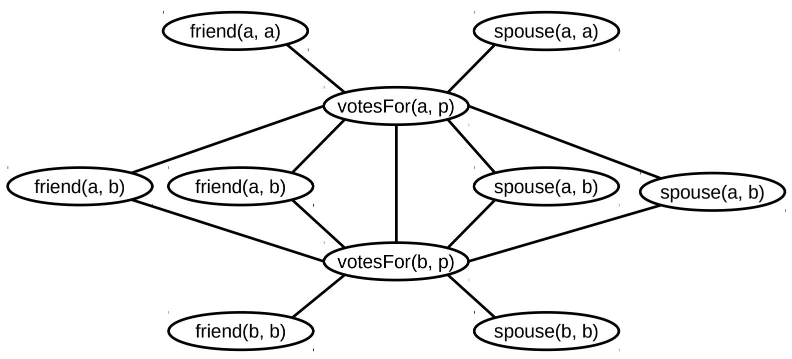


Figure 3.4: Example of Markov network. The network is obtained by grounding the formulas in our running example with constants a, b and p

domain. The probability of X taking value $x \in \mathcal{X}$ is given by:

$$P(X = x) = \frac{1}{Z} \exp\Big(\sum_{i}^{\mathcal{G}} w_i n_i(x)\Big) \tag{3.1}$$

where $n_i(x)$ is the number of true groundings of a formula f_i in a state x, w_i is the formula's corresponding weight, and Z is a normalizing factor that sums over all possible states.

3.2.2 Tasks with MLNs

Given a domain of interest, the modeller must define or learn the structure of the MLN and the weights for each formula. Once this is done, inference can be performed. One can use learning to obtain the structure or one can assert it. Likewise, weights can be learned or asserted. An overview of the learning and inference tasks now follows.

Structure Learning

Given a set of predicates and example data for the domain of interest, the MLN structure learning process learns first-order logic formulae that define the relationships between the given predicates using a sample of data in the domain. The structure learning process uses either a beam search or a shortest-path search strategy to find the best clauses to add to the MLN [Domingos and Lowd, 2009b]. In theory, structure learning avoids reliance on domain experts to write down the rules that capture the semantics of the domain. However, the structure learning process is known to be unscalable for large datasets [Khosravi and Bina, 2010]. Given this, we have not performed structure learning and hence do not report any results in this respect.

Weight Learning

Given a set of rules $\mathcal{F}$ and a database of evidence from the domain of interest, the weights of the rules can be learned. In this process, one or more predicates whose truth values are unknown are designated as query predicates. Predicates of which atoms are assumed to be known are designated as evidence predicate. Let $Y = (Y_1, Y_2, ..., Y_n) \in \mathcal{Y}$ where each variable in Y is a ground atom encoded by a query predicate. Let $\mathcal{Y}$ be the set of all possible truth value assignments of n binary query predicates. Similarly, let $X = (X_1, X_2, .., X_m) \in \mathcal{X}$. Each variable in X is a ground atom encoded by an evidence predicate and $\mathcal{X}$ is the set of all possible truth value assignments of m binary evidence predicates. The conditional likelihood of query predicates given the evidence is computed by

$$P(Y = y \mid X = x) = \frac{1}{Z_x} \exp\Big(\sum_i^{\mathcal{G}} w_i n_i(y, x)\Big) \tag{3.2}$$

where $\mathcal{G}$ is the set of ground formulas in the Markov network. Note that the normalizing constant Z_x is computed over possible worlds consistent with x.

The goal of learning procedure requires fixing the assignments of both Y and X (i.e., both evidence and query predicate need to be supplied in the training data). It finds weight which maximizes the above likelihood of query predicates assignment Y given the evidence predicate assignment X. There are number of algorithms which can be used for learning the weight of an MLN given a dataset (e.g., Voted-perceptron-like [Singla and Domingos, 2005] and MC-SAT [Lowd and Domingos, 2007]). The details of such algorithms is beyond the scope of this thesis since we use default learning algorithms that comes with the Alchemy toolkit[4].

The weights produced for each rule can be positive, negative or zero. A positive weight is an indication that the corresponding rule is supported in the domain given the evidence. On the other hand, a negative weight is an indication that the corresponding rule is not supported, and, in fact, its negation is supported with the corresponding negative weight. Finally, a weight of 0 indicates that there is no evidence (either for or against) in the domain for the corresponding rule.

Example 1 (Weight learning in action)**.** Assume that we have the rules in Listing 3.2 which capture one of the propositional features (i.e., support for gun control) of the *person* table in the dataset shown by Figure 3.3. The rules state that if a person

```
w1: votesFor(A,"D")   ← SupGunControl(A)
w2: votesFor(A,"R")   ← SupGunControl(A)
w3: votesFor(A,"I")   ← SupGunControl(A)
```

Listing 3.2: Un-weighted that capture propositional features of Figure 3.3

p supports a gun control policy, then there is some likelihood that the person will vote for a certain party. Running a weight learning algorithm on the above set of rules using the dataset in Figure 3.3 produces the weighted rules in Listing 3.3.

So the hypothesis that a person who supports gun control tends to vote for the *D* party is supported by the dataset used, and the rule that encodes this hypothesis

[4]Alchemy(`alchemy.cs.washington.edu/`) is an implementation of MLNs. Its default learning method is the MC-SAT algorithm.

```
1.85:  votesFor(A,"D")   ← SupGunControl(A)
-1.15: votesFor(A,"R")   ← SupGunControl(A)
-1.14: votesFor(A,"I")   ← SupGunControl(A)
```

Listing 3.3: Weights learned for the rules that capture propositional features of Figure 3.3

receives a relative weight of 1.85. On the other hand, the hypotheses encoded by the alternative rules are not supported by the given dataset.

Inference

A basic inference task that is performed in MLNs is finding the most probable assignments of truth values Y given the evidence X. This done my maximizing the conditional likelihood given by equation 3.3. Thus this inference can be defined by the following formula:

$$\operatorname*{argmax}_y P(y \mid x) = \operatorname*{argmax}_y \frac{1}{Z_x} \exp\Big(\sum_i^{\mathcal{G}} w_i n_i(y,x)\Big) \tag{3.3}$$

As shown by Domingos and Lowd [2009c], equation 3.3 can be reduced to

$$\operatorname*{argmax}_y P(y \mid x) = \operatorname*{argmax}_y \sum_i^{\mathcal{G}} w_i n_i(y,x)) \tag{3.4}$$

This entails that inference can be computed by finding the assignment of truth values y that maximizes the sum of the satisfied ground rules. As is the case for most graphical models, full inference in MLNs is intractable[Domingos and Lowd, 2009c]. The basic reason behind this is that enumerating all possible assignments of truth value for large MLN is infeasible. In response, one technique that can be used to perform inference over a ground MLN is to used a sampling based approach. The sampling approach begins by assigning random truth values to a ground query atoms. It then computes the cost of unsatisfied clauses by summing the weights of such clauses. It then proceeds to iteratively re-sample the values of query atoms with the goal of minimizing the cost of unsatisfied clauses.

Example 2 (Inference in action)**.** The MLN inference process takes as input a weighted MLN and set of constants. Given the rules in Listing 3.1 and the datasets in Figure 3.3, the values in Listing 3.4 are produced by MAP inference. The values

```
VotesFor(P3,I) 0.609989
VotesFor(P3,R) 0.592991
VotesFor(P5,R) 0.568993
VotesFor(P2,I) 0.564994
VotesFor(P5,D) 0.542996
VotesFor(P2,D) 0.489001
VotesFor(P1,R) 0.471003
VotesFor(P4,I) 0.469003
VotesFor(P4,R) 0.467003
VotesFor(P1,D) 0.438006
```

Listing 3.4: The conditional probabilities produced by inference using the rules in Listing 3.1

produced are interpreted as the conditional probability of the corresponding atom being true given the evidence and the MLN. In our case, we use such the resulting probabilities to rank the results produced by the inference process in a way that informs user interaction with the results (see Chapter 5).

3.3 Interpreting LD search results using Markov Logic

The key idea of our approach is to use probabilistic inference over the search results to populate the meta-model in Figure 3.5. This meta-model characterizes the tabular structure that we aim to use to present the search results. Such an instantiation of the meta-model is a tabular representation of the search results and we refer to it as a *report*. Thus, given the search result, our goal is to infer that some resources are entity types and some are entities (i.e., elements in the extent of the inferred entity types), some other resources are properties of some inferred entity type and some are property values (i.e., elements in the domain of the inferred property).

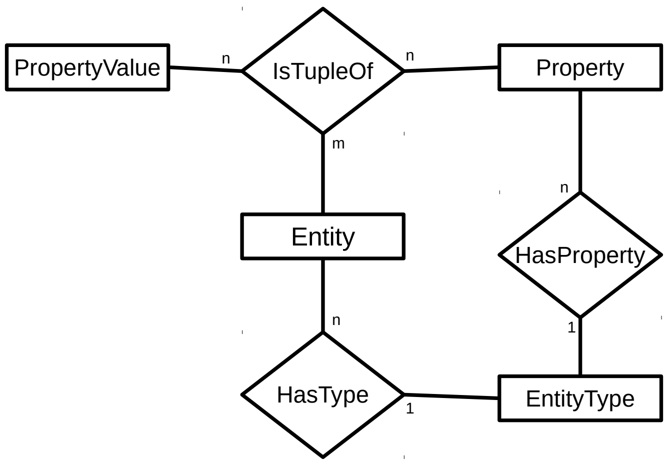


Figure 3.5: The meta-model for population with LD search results.

Identifying which entities, entity types, properties and property values are relevant from the user's point of view is a task with uncertain outcomes. One source of uncertainty is the search itself (i.e., its associated precision and recall). For example, the RDF individual `dbr:Raja_Casablanca` is returned in the search results for the search term `Casablanca` (see Figure 3.6) but should not be listed in a table describing the 1942 US-produced film Casablanca. Another source of uncertainty is that not every RDF predicate or RDF type can be considered a suitable representation for a user-level property or type. For example, the RDF type `wd:Q11424` may not be useful to denote an entity type from a users point of view, and is therefore not suitable to be reported as a table to the end user.

A key advantage of an SRL approach, such as MLNs, for this task is that it provides an opportunity for incorporating different forms of evidence to strengthen the inference decisions that populate the target meta-model. In Chapter 4, we extend the work presented in this chapter to include evidence from domain ontologies and user provided feedback in the construction of the tabular reports. We first describe the MLN for probabilistic inference of instantiations of our meta-model from LD search results. We also describe the learning and inference processes. Finally, we present and discuss the experimental results of our initial investigation. The discussion will motivate the investigation reported in Chapter 4.

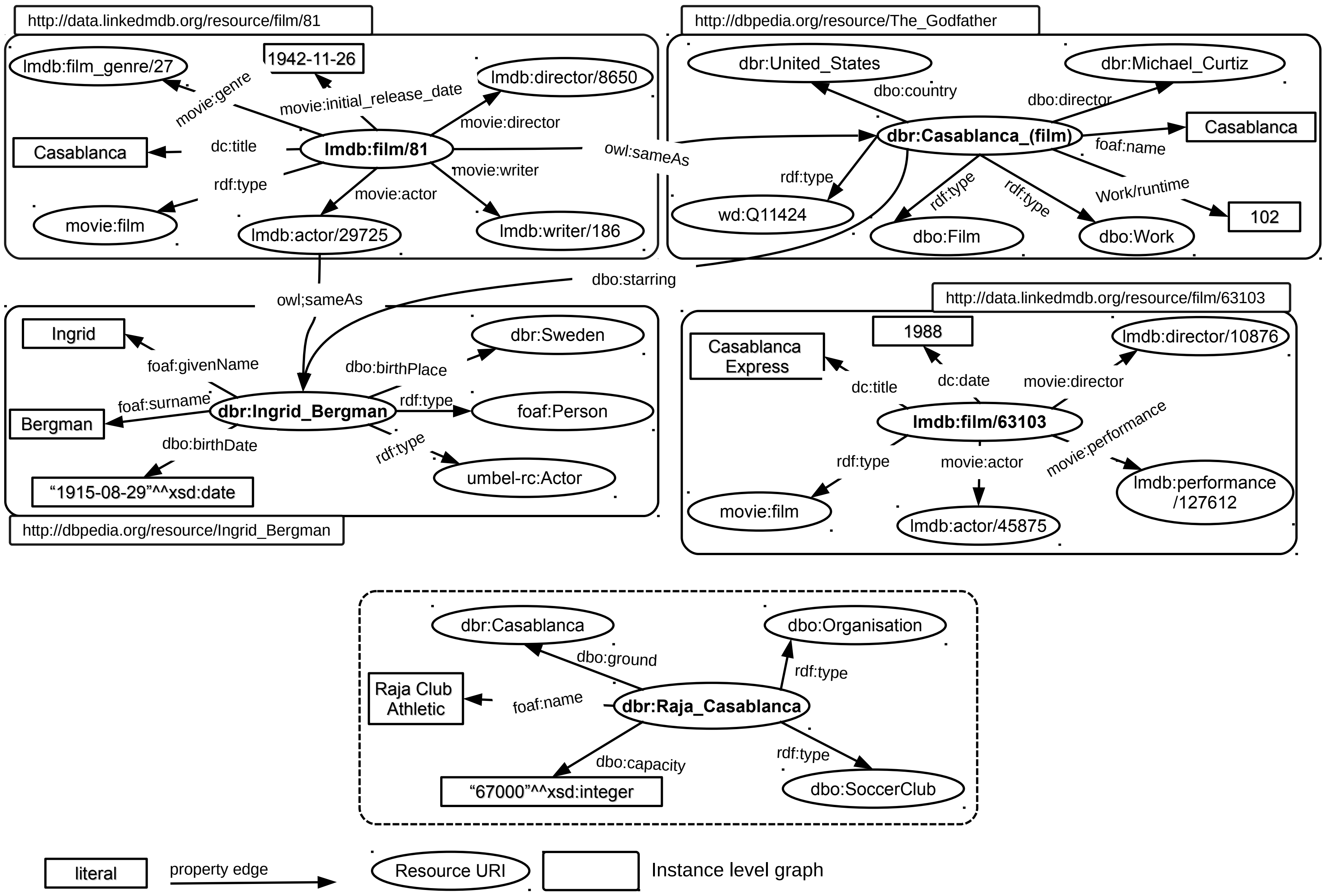


Figure 3.6: A RDF Sub-graph for some of the results for the term `Casablanca`

3.3.1 An MLN for LD search results

Our approach follows the direction of other applications of MLNs to specific problems, (e.g., [Niu et al., 2012] and [Singla and Domingos, 2006]): we modeled the structure of the MLN by defining rules that relates RDF annotations of resources to instantiations of a meta-model that characterizes our target report. Our goal is to capture the uncertainty of mapping from RDF to our meta-model constructs using the MLN rules. We used the weight learning process provided by an MLN implementation[5] to learn the uncertainty of rule these rules. The approach is comparable to that often used in probabilistic graphical models literature, where dependencies among variables are manually specified and then the parameters that specify the probability distributions are learned from data. One advantage of using the MLNs approach is that it comes with general-purpose learning and inference algorithms. Thus one does need to implement specialized techniques for each problem.

We now describe the MLN in Listings 3.5 and 3.6, which we use to interpret RDF search results. Recall that evidence predicates characterize ground atoms with known truth values. Here, the evidence predicates represent observed triple patterns in the search results. These predicates are defined over the `uri` and `literal` domains. To enable a simpler and more direct way of writing rule bodies, we partition the space of triples. As such, we use `Triple1` to represent RDF triples with URIs in the object position and `Triple2` to represent RDF triples that have literals in the object position. Without such a partition, rule R1 in our MLN would be expressed as:

```
EntityType(o) ← IsURI(s) ∧
                 Triple(s,"rdf:type",o)
```

Besides notational convenience, we note that the additional `IsURI` predicate would result in additional nodes in the ground Markov network, and hence in a less efficient

[5] `alchemy.cs.washington.edu/`

```
// Evidence Predicates
Triple1(uri, uri, uri)
Triple2(uri, uri, lit)

// Query Predicates
Entity(uri)
EntityType(uri)
Property(uri)
PropertyValue(uri)
LnkPropertyValue(uri)
HasProperty(uri,uri)
HasType(uri,uri)
```

Listing 3.5: The predicates used to define the rules in Listing 3.6

MLN.

Figure 3.5 show the meta-model which characterizes that structure of the target reports. Each node in the meta-model (expect for `IsTupleOf`) represent a query predicate in our MLN model. Each instances of these query predicates represents an instance of a corresponding meta-model construct. The `Entity`, `EntityType` and `Property` predicates model whether a given URI represents an entity, an entity type or a property in our report. For example, given the results in Figure 3.1 and the report shown in Figure 3.2, the following instantiations of query predicate are true:

```
Entity(dbr:The_Godfather)

EntityType(dbo:Film)

Property(movie:director)
```

Conversely, the following are false:

```
Entity(lmdb:/film/12057)

EntityType(wd:Q215627)
```

This is because our search was for the "Godfather actors" and the results above are irrelevant to the search term for considering that we are seeking a report about a particular film in this case.

We also define predicates for `HasProperty` and `HasType`. `HasProperty` models a relationship between two URIs where the first is an entity type and the second is an attribute of that type. `HasType` models a relationship between two URIs where

```
R1: EntityType(o)         ← Triple(s,"rdf:type",o)
R2: Entity(s)             ← Triple1(s,"rdf:type",o) ∧
                             EntityType(o)
R3: Attribute(p)          ← Triple2(s,p,o) ∧
                             Entity(s) ∧
                             AttributeValue(o)
R4: Attribute(p)          ← Triple1(s,p,o) ∧
                             Entity(s) ∧
                             LnkAttributeValue(o)
R5: LnkAttributeValue(o) ← Triple1(s,p,o) ∧
                             Entity(s) ∧
                             Attribute(p)
R6: AttributeValue(o)     ← Triple2(s,p,o) ∧
                             Entity(s) ∧
                             Attribute(p)
R7: HasType(s,o)          ← Triple1(s,"rdf:type",o) ∧
                             EntityType(o)
R8: HasProperty(t,p)      ← Triple1(s,"rdf:type",t) ∧
                             Triple2(s,p,o) ∧
                             EntityType(type) ∧
                             Attribute(p) ∧
                             AttributeValue(o)
R9: HasProperty(t,p)      ← Triple1(s,"rdf:type",t) ∧
                             Triple1(s,p,o) ∧
                             EntityType(t) ∧
                             Attribute(p) ∧
                             LnkAttributeValue(o)
```

Listing 3.6: An MLN rule set that uses RDF triples to infer a structure from LD search results

the first is an entity, the second is an entity type, and the first is an instance of the second. Examples from Figure 3.6 are:

```
HasProperty(dbo:Film,dbo:director)
HasType(lmdb:film/81,movie:film)
```

Finally, we use two predicates to interpret values in the object position of RDF triples, viz., `AttributeValue` and `LnkAttributeValue`.

In addition to the described predicates, the MLN contains rules that encode relationships between RDF triples returned in the results and the query predicates. Rules R1, R2 and R7 use the `rdf:type` construct for the inference of `Entity`, `EntityType` and `HasType`. The idea here is that RDF types that occur frequently in the results are likely to be instances of the `EntityType` meta-type. For example, in Figure 3.6, `Film` is more probably an entity type given that it appears as the type for three resources, as opposed to `SoccerClub` which only appears for one. Some of the rule heads may appear in rule bodies of other rules as derived evidence. For example, in rule R2, a returned RDF triple `(s, "rdf:type", o)` where `o` is inferred to be an entity type, counts as evidence that `s` is an entity. This is possible because the MLN inference algorithm iteratively updates the state of satisfied clauses based on the results of previous inferences [Domingos and Lowd, 2009c]. Unlike rules R1-R7, which only use a single triple for making inferences, rules R8 and R9 utilize two triple patterns as evidence for making an inference about the `HasProperty` relation. The intuition here is that if an RDF type and an RDF predicate often share resources, then it is likely that they are related by the `HasProperty` relation. Consider again the example search results in Figure 3.6. Given rules R8 and R9, if the rules have positive weights, `HasProperty(Film, director)` is more probable than `HasProperty(Film, writer)` because writer is a predicate that holds for fewer resources than director does.

3.3.2 Experimental Evaluation

We now present an empirical evaluation of our MLN-based approach to infer a tabular structure from LD search results. The goal of this experiment is to measure how effectively the MLN based approach ranks the instances of meta-model instantiations given the evidence in the search results. Our goal in here is limited to showing the idea of using an MLN approach in representing the uncertainty of mapping RDF triples into instances of our meta-model. The uncertainty captured through the weight learning procedure over training data. Subsequently, we measure the precision and recall of the MLN-based inference over LD search results using the learned weighted MLN model. Note that we are not evaluating the correctness of the rules presented in Section 3.3.1. This is because, as mentioned earlier, an MLN formula expresses a rule of thumb, i.e. it is a claim on the training data with likelihood. The likelihood of the rule translates into a weight. So by definition, a rule in an MLN never 100% correct. Given that this a modeling exercise, there could be other ways to model the problem at hand.

Dataset

To our knowledge, there is no publicly available standard dataset that allows us to evaluate our approach. In order to learn the weights for the MLN described in Section 3.3, we chose four domains and conducted ten searches using the Sindice [Oren et al., 2008] and the Falcons [Cheng and Qu, 2009] search engines. From each search engine, we collected the top 20 results for each search term. Table 3.1 shows the terms we have used. Table 3.2 provides further details about the size and composition of the datasets for each domain. The most common datasets from which triples appear in the results are DBpedia, semanticWeb.org[6], Linked Movie Database (LMDB), and MusicBrainz. While DBpedia is a general knowledge base, the other

[6]data.semanticweb.org

Table 3.1: Search terms used in constructing the evaluation dataset

Domain	**Search Terms**
Cities	Berlin,Manchester
Films	Godfather,Casablanca, Godfather actors
Organizations	Apple Inc., Microsoft, UK universities
People	Tim Berners-Lee, Chris Bizer

sources are domain-specific. SemanticWeb.org captures information about the Semantic Web community, LMDB is an online RDF database extracted from IMDB, and MusicBrainz provides information about artists, releases, tracks, and the relationships between them. Note that while the most intuitive domain for terms `Berlin` and `Manchester` is *cities*, many of the returned results for these terms are not from this domain. This suggests that terms that would not be often thought of as ambiguous, often become so in search results because of the diversity of resources in the WoD. Also, note that the number of typed individuals in each domain is greater than 40, i.e., it is greater than the number of RDF results collected, because, often, a single result contains descriptions about more than one individual. For example, the RDF description of artist *Melissa Manchester* in MusicBrainz, which is returned as a result of search of the term *Manchester*, includes descriptions about albums and singles by the artist. For this reason, the number of instances for some of the common types has turned to be more that the number of hits we obtained from each LD search engine.

Search results in this dataset were preprocessed to remove triples containing RDF types that belong to the `yago` and `dbyago` name-spaces. The reason is that such types (e.g, `yago:GangsterFilms` and `yago:AmericanEpicFilms`) are used in site-specific ways to categorize resources, as opposed to assigning real-world entity types to resources. Also, triples that contain domain-independent RDF predicates and have literal objects were removed: `dct:abstract`, `rdfs:comment`, `rdfs:label`, `skos:prefLabel`, `skos:altLabel`, `skos:note` and `dce:description`.

Domain	**# of Triples**	**Distinct Types**	**Distinct Properties**	**Typed Individuals**	**Common Types**		**Common Data sources**
					Type	*# of Instances*	
People	2488	42	160	70	swrc:InProceedings foaf:Document swrc:Article	30 30 9	semanticweb.org dblp.l3s.de
Organizations	12874	41	142	99	sl:Tag foaf:Person dbo:Organisation	30 21 16	dbpedia.org semanticweb.org
Films	8328	112	213	86	foaf:Person movie:actor dbo:Agent	45 33 15	linkedmdb.org dbpedia.org
Cities	16318	114	310	155	schema:MusicAlbum schema:Person pos:SpatialThing	53 19 18	musicbrainz.org dbpedia.org

Table 3.2: Statistical information of the evaluation dataset

Ground-Truth Definition

To conduct our experimental evaluation on the collected dataset, we annotated it with the ground truth. The annotation of all the domains in the dataset was done by a single person, i.e., the author of this thesis. To exemplify the reasoning behind this procedure, we provide an example of the ground truth for some of the results of the search term `Casablanca`, shown in Figure 3.6. While annotating the dataset, we took into account the relevant domain of the term. For example, in the *films* domain we considered, among others, the types `Film`, `Actor`, `Producer` and `Director` as true instances of `EntityType`. Conversely, the types `SoccerClub` and `Organisation` are not true instances of `EntityType` in the films domain. Similarly, the individuals `dbr:Ingrid_Bergman` and `lmdb:film/63103` (i.e., Casablanca Express) are instances of the meta type `Entity`, unlike `dbr:Raja_Casablanca`. In order to define `EntityType` instances, we used `schema.org` as a reference schema for the ground-truth annotation of `Property` and `HasProperty`. For example, given Figure 3.6, and with reference to `schema.org`, the properties of the `EntityType Film` are `actor`, `director`, `runtime`, `starring`, `country` and `genre`. Similarly, the properties of `Actor` are `givenName`, `surname`, `birthPlace` and `birthDate`. Note that the properties of `Actor` in this case are also properties of the super-type `Person`. In the ground truth, subtypes inherit all the properties of the supertypes. The ground truth for the RDF graph in Figure 3.6 is shown in Figure 3.7.

```
/* HasType GT*/
HasType(lmdb:film/81, Film)
HasType(lmdb:film/63103, Film)
HasType(dbr:Casablanca_(film), Film)
HasType(dbr:Ingrid_Bergman, Actor)
```

```
/* EntityType GT */
EntityType(Work)
EntityType(Film)
EntityType(Person)
EntityType(Actor)

/* Property GT*/
Property(initial_release_date)
Property(actor)
Property(director)
Property(writer)
Property(starring)
Property(country)
Property(runtime)
Property(genre)
Property(performance)
Property(birthPlace)
Property(birthDate)
Property(givenName)
Property(surname)
```

```
/* Entity GT */
Entity(lmdb:film/81)
Entity(lmdb:film/63103)
Entity(dbr:Casablanca_(film))
Entity(dbr:Ingrid_Bergman)

/* HasProperty GT */
HasProperty(Work, runtime)
HasProperty(Work, author)
HasProperty(Film, actor)
HasProperty(Film, starring)
HasProperty(Film, director)
HasProperty(Film, genre)
HasProperty(Film, country)
HasProperty(Film,
   initial_release_date)
HasProperty(Film, performance)
HasProperty(Person, birthDate)
HasProperty(Person, birthPlace)
HasProperty(Person, givenName)
HasProperty(Person, surname)
```

Figure 3.7: The ground truth annotation for the RDF in Fig. 3.6

Methodology and Results

To evaluate our approach, we used a five-fold cross-validation procedure [Refaeilzadeh et al., 2009] on the annotated dataset. In each iteration of the procedure, we split the dataset into training and test sets. We ran the weight learning process of the training subset and the inference process on the test subset. To ensure that the weights are not skewed towards a particular domain of search, we randomized the search results to ensure that every split contained search results from every one of the domains shown in Table 3.1. Given this randomization step, the inference process was ran on data that contained a mix from all domains. Thus our results in Table 3.4 are computed over all the domains at once and individually. Table 3.3 shows the average weights learned for the MLN rules. One observation is that the

Rule ID	**Average Weight**
1	43.161 ± 64.782
2	3.227 ± 0.709
3	4.051 ± 0.435
4	4.161 ± 0.399
5	2.903 ± 0.888
6	2.491 ± 0.537
7	1.807 ± 0.307
8	1.140 ± 1.279
9	1.229 ± 0.890

Table 3.3: Average weights learned for each rule

weight of `R1` is about ten times higher than the weight of the other rules. The reason for this is that `rdf:type` provides the most direct signal for the inference in `R1`, i.e., that a URI in the object position of an `rdf:type` triple denotes an entity type. This is because the majority of the types of in RDF results returned by underlying search engines tend to be relevant to the domain of the search (except for the cities domain). So in this case, the majority of RDF types in the results match the expectation of the true (according to the ground truth) entity types. The strength of the signal for the other query predicates in the MLN is less direct, and hence much weaker.

To evaluate our approach, we have compared the results of the MLN with the results of a random classifier. The random classifier simply assigns a random number in the range $[0, 1]$ to each instance of each query predicate. The results are evaluated by computing the area under the precision-recall curve (AUC) (see Appendix A for an example on AUC computation). To compute the AUC, the precision-recall (PR) curve is generated first. The precision and recall are computed according to the following [Fawcett, 2006]:

$$Precision = \frac{\text{Number of propositions correctly predicted as positives}}{\text{Number of propositions predicted as positives}} \tag{3.5}$$

Predicate	**Random**	**MLN**
Entity	0.03 ± 0.00	0.608 ± 0.19
EntityType	0.02 ± 0.00	0.742 ± 0.07
Property	0.04 ± 0.00	0.190 ± 0.05
HasProperty	0.01 ± 0.00	0.058 ± 0.018
HasType	0.01 ± 0.00	0.760 ± 0.09

Table 3.4: AUC PR for the inference results of the experiment

$$Recall = \frac{\text{Number of propositions correctly predicted as positives}}{\text{Number of positive propositions in the data}} \tag{3.6}$$

A PR curve is produced by plotting a point for the precision and recall obtained at given threshold values. The AUC is a single scalar value that summarises the area under the PR curve. This summary is often used to evaluate the performance of machine learning systems that produce continuous outputs (e.g., probability scores) instead of binary ones [Boyd et al., 2013]. As mentioned earlier, the goal of our experiment is to measure the effectiveness of our MLN approach in ranking instances of the inferred structure. A significant property of the AUC is that it is equivalent to the probability that the classifier will rank randomly selected positive propositions higher than randomly selected negative ones [Fawcett, 2006]. Thus, the higher the AUC for a query predicate the more effective our MLN approach in ranking instance of that query predicates.

Note that in this experiment the F-measure would be inadequate, simply because it combines the precision and recall calculated at a specific threshold value, whereas, our goal is to evaluate the proposed approach without predefining a threshold for the inferred probability values of query predicates. This is because the distributions of the inferred probability values might vary across different query predicates. So, for example, a threshold of 0.5 might be adequate for computing the F-measure of some query predicates (e.g., `Entity`) but not others (e.g., `HasProperty`).

Table 3.4 shows the AUC scores obtained for every query predicate averaged over five folds of the cross-validation procedure.

In general, the MLN performs better than the random classifier. It performs particularly well on `Entity`, `EntityType`, and `HasType`. Explicit assertions of the types in RDF seems to provide compelling evidence for the inference of the `HasType` relation, thus leading to a relatively high AUC score. Although the MLN outperforms the random classifier in the case of the `Entity` and `EntityType` query predicates, the average AUC for these query predicates is slightly lower than the AUC for the `HasType`. This can be attributed to the ambiguity in some of the search results. The MLN infers that the resource `dbr:Berlin` (which describes the German city) is an `Entity` in our meta-model with a probability of 0.995. However, it also infers that resource `dbr:Brock_Berlin` is an `Entity` with the same probability, but this is a *false positive*. The reason for this is that our model does not include domain-specific rules and, therefore, strong signals for resources from different domains (and hence ambiguous) are treated equally. For `EntityType`, the most probable instances according to our MLN are `EntityType(foaf:Person)`, `EntityType(schema:Organization)` each with a probability score of 0.995. The reason for this is that these two types are common in search results regardless of the domain. We observed that some of the RDF types that are irrelevant to the domain have higher probabilities than relevant ones. For example, in the *films* domain the probability of `EntityType(schema:MusicAlbum)` is 0.89, whereas the probability of `EntityType(movie:actor)` is 0.77. This is because the RDF type `schema:MusicAlbum` is more common than the RDF type `movie:actor`.

In comparison to `Entity`, `EntityType` and `HasType`, the MLN does not preform well on `Property` and `HasProperty`. This can be attributed to two reasons. The first is that RDF properties that are related to the internal structure of the data source and carry no useful information to the user occur frequently in resources, and inferred with relatively high probability (e.g., Prob(`Property(dbo:wikiPageID)`)=0.987). The second reason is attributed to the sampling approach used by the MLN inference procedure. In MLN, by default all ground atoms have probability 0.5. MLN generates the full ground network, then run inference on top of that. So in the absence of any evidence, the default probability for ground atoms is 0.5. We observed that various `Property` instances that involve URIs that do not occur in the predicate position of triples were inferred with an average probability of 0.499. Examples of such results include `Property(dbo:Company)` and `Property(dbr:NeXT_Computer)`. The URI `dbo:Company` is in fact an RDF type, whereas `dbr:NeXT_Computer` is a URI for a resource that occurs in the object position of a triple. Note also that most query predicates are defined over the set of observed URIs. Given that rule bodies for `Property` and `HasProperty` have chains of inference (i.e., bodies are defined using other query predicates), the MLN inference procedure has a larger space to sample from, which reduces the likelihood of finding the correct answer for the query predicate. Note, also, that `Property` and `HasProperty` atoms correspond to schema elements with weak signal from the data (e.g., not a direct one-step, inference from `rdf:type` as with rule R_1). Previous research has shown that schema inference from instance data is challenging [Christodoulou et al., 2015b]. In LD search, this problem is even harder because of the variability in the data resources in terms of the descriptions they use. In Section 3.4 we discuss proposals for improving the performance of the MLN-based approach.

Note, finally, that frequency of occurrence does determine to some extent the inferences made by the model. For example, RDF types which occur frequently in the results are likely to be inferred as instances of `EntityType`. However, the frequency of occurrence is not the only factor. The inferences made by the rules is determined also by their learned weights; and the weight of a rule is determined by the training data. In order not to over-fit the learned weights, i.e., to reduce the effect of frequency of occurrence during the weight learning process, we used a 5 fold cross validation process where we randomized the search results in each fold. So, not only the frequency of occurrence in the data determine inference results, but also the weight learned for each of the rule in our MLN. Given that the approach in this chapter is a preliminary one, the search results may have a much bigger impact on the inferences made, more than one would want to. In Chapter 4, we amend this by introducing rules that bring other types of evidence to subdue the effect of frequency of occurrence in the search results.

3.4 Summary and Conclusions

LD search engines are useful for providing non-experts with access to a great variety of RDF datasets. WoD search results consist of a collection of triples, and contain RDF resources that may or may not describe real-world entities. Typically, the resulting resources may contain duplicates and/or fragments of what a user expects as an answer to the search terms given. In addition, the resources returned in the results of LD search can sometimes be semantically ambiguous, and may have resulted from heterogeneous vocabularies. These features and more mean that LD search results return complex data set that may be difficult to interpret automatically.

In order to deal with this complexity, we have described an approach based on using an MLN to model the uncertainty associated with inferring types, individuals, attributes, and attribute values that would allow one to integrate these constructs into a tabular structure characterizing real-world entities in the search results. This was done by using an MLN to express various hypotheses encoded as rules that capture the relationships between triples in the search results and the roles that the URIs and literals in those triples may play in terms of a target relational structure.

We have evaluated the approach, and while there is evidence of efficacy, this was not universally so. The approach led to good results in inferring entity types and instance-of relationships from the search results. On the other hand, entity instances and attributes were not identified accurately by the MLN. There may be different reasons for this (e.g., the rules for identifying such features could be improved upon), but our preferred interpretation is that the rules capture relevant signal but that the signal was sometimes not strong enough, i.e., that more, and more kinds of, evidence was needed.

In this context, evidence may be lacking for two reasons. First, different publishers publish data in different ways. In general, datasets in the WoD can be classified as either vertical or horizontal in the following senses. Vertical datasets (e.g., LMDB[7]) provide descriptions for entities in specific domain (e.g., films), whereas entities in horizontal datasets (e.g., DBpedia[8]) span many domains. In the dataset collected for our experiment, our observation is that sources in vertical datasets often have with few RDF types (on average, two per resource) and their features are described using a small set of RDF properties (a maximum of nine properties). On the other hand, horizontal datasets use more types (on average five per resource) and properties (a maximum of 16 properties). Thus, inferring a structure from such datasets is more challenging because, for example, noise is more likely in the possible ways in which a property can be assigned to an entity type is high. Also, a fraction of the retrieved data is not directly informative in terms of the structure of the data in the domain. For example, resources are often described with RDF properties (e.g., `dbo:wikiPageRedirects`) that carry no relevant information to the end user. Similarly, an RDF type `SoccerClub` found in the search results may not be useful for a user looking for information about films. Dealing with such properties and types requires the integration of other sources of knowledge (e.g., ontologies) that can help in defining the semantics of the domain.

The problem of inferring domain knowledge from LD search results is a difficult one. Using an SRL approach for this problem is motivated by the perception that this is an evidence-rich problem. This suggests that the search results can be combined with additional information to enable well-founded inferences to be drawn. Some examples of sources of evidence are:

[7]data.linkedmdb.org
[8]dbpedia.org/

- *Additional generic integration rules.* Our rules have focused on the identification of elements in our target meta-model using the data from the search result. However, it would be possible to write additional generic rules. For example, none of the current rules attempt to identify duplicates across the resources retrieved by the search.

- *Incorporating domain knowldege.* In the WoD, there are many domain vocabularies in the form of ontologies that are used to define common understanding of the structure of information, some of which are shared across datasets. These vocabularies characterize semantic relations (e.g., the `HasProperty` relation) among concepts and properties in a domain. Thus, incorporating such ontologies could be useful.

- *Integration of feedback.* In Sig.ma [Tummarello et al., 2010], users are able to refine the returned reports by ruling in/out specific sources of data. In our problem, feedback of different forms could be provided in a *pay-as-you-go* manner, for example, on the correctness or relevance of specific results. Such feedback could be used as evidence by our model to inform the inference of results for different query predicates.

- *Domain-specific integration rules.* Successful applications of MLNs (for example, in entity resolution or knowledge base construction) have often made use of domain-specific rules. As such, although searches are generic, it would be possible to write rules that know about certain domains, and the widely used terminologies in such domains. For example, rules could be written that are informed by common searches in search logs, or that cover terminologies that are widely used in practice [Hogan et al., 2012].

- *Integration of results of other analyses.* The current rules act directly on the search results. However, it would be possible to run additional analyses on these search results, which in turn could be reflected in rules. For example, analyses could carry out ontology alignment between search results, or could cluster triples based on attribute values. The results of such analyses could then be used as evidence predicates, and included in additional generic or domain-specific integration rules.

In this chapter our aim was to model the uncertainty of identifying which entities, entity types, properties, and property values are relevant from the user's point-of-view using Markov logic. The choice of MLNs was due to evidence in the literature that our initial model can be extended later to include other forms of evidence. However, the lessons we learned from using Markov logic, and in particular the one about the lack of direct way of encoding similarities the issues regarding the scalability of the sampling based inference approach, have motivated us to investigate an alternative approach, namely PSL.

In the next chapter, we respond to some of the shortcomings of our MLN approach-based approach and a different, more expressive and more efficient SRL formalism, viz., PSL. This allows us to expand the scope of our instantiations by bringing additional sources of evidence to the problem. We show this by incorporating evidence from ontologies and from user feedback.

Chapter 4

An evidence-based approach for integrating LD search results using PSL

Chapter 3 contributed an MLN approach to address the schema-less nature of LD search results. It described a method for automatically inferring a structural summary (i.e., tabular reports) of LD search results. Having such tabular reports will improve the visual presentation of LD search results. However, we saw that relying on syntactic evidence drawn from triples in the results alone does not yield sufficient precision and recall in some cases. More specifically, inference of `Property` and `HasProperty` relationships is affected by noise in search results that arises because of the heterogeneity of entity types. Also, many of the false positives in the inference of `Entity` and `EntityType` relationships can be attributed to the lack of context. For example, our MLN model infers that the non-relevant RDF type `schema:MusicAlbum` is an instance of `EntityType` with high probability (i.e., 0.89), whereas the relevant RDF type `movie:actor` is inferred to be an instance of `EntityType` with a probability of 0.77. One way to deal with such issues is the inclusion of evidence about the context of the search in the form of ontologies that characterize the domain of the search results. Furthermore, deriving the desired tabular representation requires the resolving the duplicates of data in the resources in the search results. Resolving duplicate instances of data requires the integration of matches of individuals, entity types, properties, and property values as evidence to decide on the similarity between resources.

Solutions to the problem of integrating heterogeneous LD sources benefit from the assimilation of different sorts of evidence. While the majority of integration systems rely on syntactic matches in order to align constructs (e.g., [Volz et al., 2009; Hu et al., 2011; Ngomo and Auer, 2011]), there are also systems that have utilized evidence encoded in linguistic resources (such as WordNet [Miller, 1995]) and LD vocabularies to support the integration process (e.g., [Nikolov et al., 2012; Saïs et al., 2009; Scharffe et al., 2009]). Additionally, there has been an interest in utilizing user feedback to learn models that enable the improvement of integration decisions made by automated techniques (e.g., [Tummarello et al., 2010; Demartini et al., 2013; Isele and Bizer, 2013; Kejriwal and Miranker, 2015]).

Thus, having a uniform, well-founded approach to the assimilation of different sources of evidence to address the challenges of integrating LD search results seems to be a promising direction of research in this area. In this regard, SRL techniques are appealing. First, LD is inherently relational insofar as LD expresses how individuals, RDF types, RDF properties and domain knowledge relate to each other in a variety of ways. For example, two RDF types might be related to each other by virtue of being in the domain of a similar set of RDF properties or by having a similar set of individuals in their extent. SRL techniques allow us to encode rules that express such relations. Second, LD coming form different sources is likely to be noisy. For example, noise might stem from reliance on matching algorithms to reduce the multiplicity in search results. In fact, matching is error prone, which cause decisions to be uncertain [Gal, 2006]. SRL models allow for probabilistic inference which is helpful when reasoning with noisy data.

In this chapter, we exploit an SRL technique, viz., Probabilistic Soft Logic (PSL), to assimilate different kinds of evidence with a view to generating tabular summaries of search results stemming from multiple heterogeneous LD resources. PSL is a recent SRL approach that unifies logic and probability. PSL provides a language for encoding expressive domain knowledge through FOL formulae, while handling the uncertainty that arises from combining different types of evidence through graphical models, viz., Markov networks. Reasons for adopting PSL in our approach are twofold. One, a number of studies [Bach et al., 2013a; Beltagy et al., 2014] suggest that PSL scales better than MLNs in terms of learning and inference, as the structure of the rule base becomes more complex. This allows us to extend our model with more predicates and rules that handle the additional types of evidence. Second, a key distinguishing feature of PSL is that ground atoms have soft, continuous, truth values in the interval $[0, 1]$ rather than the binary truth values used in MLNs. This enables reasoning with evidence derived from similarity values (e.g., syntactic matches), and reasoning with set similarity, which can be useful in the inference of relationships in our meta-model.

In this chapter, we present a PSL rule-base that combines three sources of evidence, viz., syntactic matching, domain ontologies and user feedback. As we discuss later, the techniques we contribute in this chapter are generic and can be applied to search terms in any domain. The remainder of this chapter is structured as follows. In Section 4.1, we introduce PSL in terms its of syntax, semantics, and its accompanying learning and inference procedures. Section 4.2 describes our proposed approach for using PSL to combine different kinds of evidence with the goal of integrating LD search results. In Section 4.3, we provide an empirical evaluation of our approach where we show how principled, uniform use of different types of evidence improves the integration quality for the end user. In Section 4.4, we discuss the related work. Finally, Section 4.5 concludes with an overall discussion of the proposed approach.

4.1 PSL

In this chapter, the formalism used to infer an integrated structure over returned LD search results is PSL [Bach et al., 2015].

As in the MLN approach in Chapter 3, here too we contribute a set of PSL rules, over an extended version of the target meta-model used there. The rules express probabilistic relationships between triples in the search results. After the rules have been converted into a *PSL program*, the PSL implementation uses the returned results to instantiate the target meta-model, thereby yielding an integrated, structured view over the results. As with Markov Logic, PSL combines a first-order logic syntax for rules with Markov networks to yield a unifying formalism where logical predicates are grounded into nodes in an undirected graphical model. In contrast with MLNs, logical predicates in PSL take *soft truth values* from the interval $[0, 1]$, rather than 0 or 1 values only. This enables the introduction of similarity functions as a modelling construct. In what follows, we briefly introduce PSL by describing its syntax, semantics, learning, and inference tasks. We provide examples for making inferences over the database shown in Figure 4.1, which describes a small part of the publication domain.

Document	
DocumentID	*Title*
d1	t1
d2	t2
d3	t3
d4	t4

Author	
DocumentID	*Author*
d1	a1
d1	a2
d2	a3
d3	a4
d4	a1
d4	a2

Figure 4.1: Example database describing few publications, with attribute table for documents (*Document*) and table describing the authorship relation (*Author*)

4.1.1 PSL Syntax

We start with a number of basic definitions.

Definition 7. A **PSL program** is a set of weighted first-order logic formulae of the form $w : A \leftarrow B$, where w is a non-negative weight, B is a conjunction of literals and A is a single literal. If the weight w is a very large number that tends to ∞, then the rule is called a constraint and is denoted by the prefix c.

Definition 8. A **term** is either a constant or a variable. A constant is denoted by a quoted string, e.g., `"child"`, whereas a variable is denoted by an *identifier* which begins with a capital case letter followed by zero or more letters, e.g., `Author`.

Definition 9. A **predicate** is a relation between terms given by a *unique identifier* and terms it accepts as arguments. For example, `SameDocument(A,B)` defines a predicate named `SameDocument` which accepts two arguments given by the variables `A` and `B`.

Definition 10. An **atom** is a predicate with zero or more arguments. If there are no variables in the arguments of the atom, then it is called a *ground atom*.

Definition 11. A **literal** is an atom or its negation.

A PSL program is similar to an MLN in the sense that it uses weighted first-order-logic rules to define models. However, unlike an MLN, only universally quantified predicates are allowed in PSL. Furthermore, in contrast to an MLN, weights in PSL cannot be negative. Also, whereas MLNs defined over Boolean variables, a PSL program is defined over continuous variables that take on values in the interval $[0, 1]$.

To define a program, PSL uses a syntax that has features of classical first-order logic and of object-oriented languages. Consider the example PSL program in Listing 4.1

```
// rules
3.5: SameDocument(A,B)   ⃖̃ SimTitle(A.Title, B.Title) ∧̃
                            {A.Author} SimSetEq[SimAuthor] {B.Author}
c  : SameDocument(A,C)   ⃖̃ SameDocument(A,B) ∧̃
                            SameDocument(B,C)

// prior
0.1: ¬̃ SameDocument(B,C)
```

Listing 4.1: A PSL program describing the publication database in Fig. 4.1

The semantics, including the tilde-capped connectives, is explained below but, intuitively, this PSL program states that, given any individuals `a` and `b` (specific documents in the case of this example) instantiating the variables `A` and `B`, the rules make a claim that `a` and `b` are similar if they have similar titles and similar sets of authors. The above rules assume that the predicates `Title(D, T)` and `Author(D, A)` are defined to capture the relationships shown in Figure 4.1. In Listing 4.1, `A.Title` is a variable that denotes the range of the `Title` relationship for a given `A`. If we wanted to denote the domain of the `Title` relationship, then we would write `A.Title(inv)`. For instance, given the example in Figure 4.1, if `A` = `d1`, then `A.Title` = `t1`, and if `A` = `t1`, then `A.Title(inv)` = `d1`. Variables can also be used to denote sets of individuals. For instance, in Listing 4.1, {`A.Author`} denotes the set of authors related to a document in our running example. Thus, if `A` = `d4`, then {`A.Author`} = {`a1,a2`}.

Atoms in PSL can be user-defined. Thus, a binary predicate `SimTitle` can denote a user-defined similarity function that returns truth values in $[0,1]$. Atoms in PSL can also capture relationships between sets of individuals. For example, the truth value of {`A.Author`} `SimSetEq`[`SimAuthor`] {`B.Author`} is the similarity of the respective sets of individuals {`A.Author`} and {`B.Author`} as a function of, and parametrized by, the user defined predicate `SimAuthor` and computed as follows:

$$SimSetEq[SimAuthor] = \frac{\sum_{i\in\{A.Author\}}\sum_{j\in\{B.Author\}} SimAuthor(i,j)}{|\{A.Author\}| + |\{B.Author\}|} \tag{4.1}$$

The value of `SimAuthor` can either be observed, i.e., given as evidence, or inferred through other rules.

PSL allows to define a *prior* over query predicates. A prior is a weighted rule, which has a negative literal as a head and an empty body, and it is often assigned a small weight. Priors ensure that the most probable value for any given ground query atom is 0 unless there is evidence that suggests otherwise. In Listing 4.1 a prior is defined for the `SameDocument(B,C)` query predicate which ensure that no two documents are the same unless there is evidence, which is assimilated by other rules, that suggests the two documents are probably the same.

Weighted rules in PSL define uncertain relationships that may hold. On the other hand, a constraint defines a rule that must not be violated. The example shown in Listing 4.1 defines a transitivity constraint that must hold for the `SameDocument` relationship.

4.1.2 PSL Semantics

Definition 12. An **evidence predicate** is a predicate that ground atoms of which are completely observed, i.e., the soft truth values of the ground atoms are observed. If the soft truth values of a ground evidence predicate is not observed, then it is assumed to be 0.

Definition 13. A **query predicate** is a predicate that ground atoms of which are unobserved, i.e., the soft truth values are of the ground atoms are unknown.

As in MLNs, a PSL program is grounded in a Markov network given some input. The input to a PSL program (in inference and learning tasks) consists of a specification of both evidence (e.g, `Title` and `Author`) and query (e.g., `SameDocument`) predicates, and input data that is pre-processed in the form of ground atoms that are expressed in terms of the specified predicates. For the evidence predicates, the value of the ground atoms are completely observed, i.e., all evidence atoms are assigned a soft truth value in $[0, 1]$. For query predicates, the value of the ground

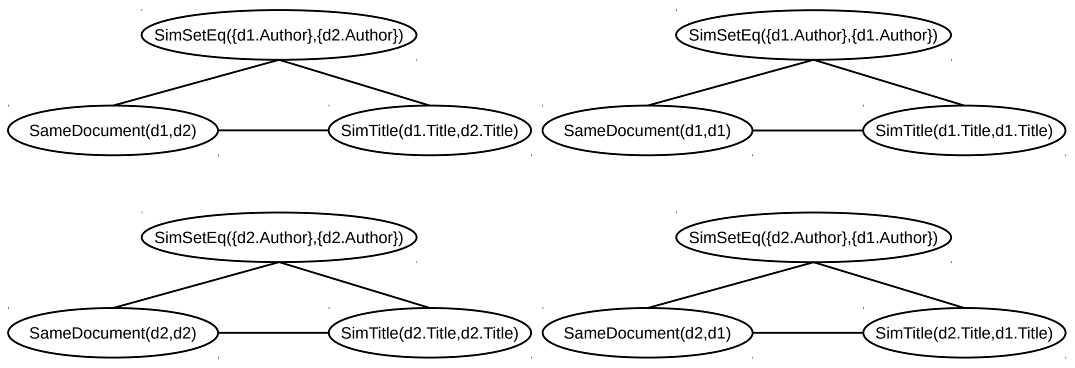

SimSetEq({d1.Author},{d2.Author})
SameDocument(d1,d2)
SimTitle(d1.Title,d2.Title)
SimSetEq({d1.Author},{d1.Author})
SameDocument(d1,d1)
SimTitle(d1.Title,d1.Title)
SimSetEq({d2.Author},{d2.Author})
SameDocument(d2,d2)
SimTitle(d2.Title,d2.Title)
SimSetEq({d2.Author},{d1.Author})
SameDocument(d2,d1)
SimTitle(d2.Title,d1.Title)

Figure 4.2: The Markov network created by grounding the first rule in Listing 4.1 using two individuals

atoms is unobserved. Note that if a ground atom does not have a specified value, i.e., unobserved, it will have a default value of 0 if its predicate is an evidence predicate or it will remain unobserved if the predicate is a query predicate. Similar to MLNs, the rules are grounded in a Markov network where nodes in the network correspond to ground atoms in the input, and edges are created between two nodes if the corresponding ground atoms occur in a ground rule. Figure 4.2 illustrates the ground network created for the rules in Listing 4.1 given the two documents `d1` and `d2` in Figure 4.1.

The main difference between PSL and Markov logic is that the ground nodes take values in the interval $[0, 1]$. To perform inference, the degree to which a ground rules is satisfied from its constituent atoms requires relaxing (with respect to the Boolean definitions) the semantics of the logical connectives. PSL uses the Łukasiewicz t-norm and its corresponding co-norm to relax the logic AND and OR, respectively. The relaxation is exact for the extremes, i.e. 0 and 1, and provide a consistent mapping for all other values.

To distinguish them from their the Boolean operators, the relaxed connectives notated with a capping tilde are defined as follows:

$$a \mathbin{\tilde{\wedge}} b = \max\{0, a + b - 1\} \tag{4.2}$$

$$a \mathbin{\tilde{\vee}} b = \min\{a + b, 1\} \tag{4.3}$$

$$\tilde{\neg}\, a = 1 - a \tag{4.4}$$

Then, $A \mathbin{\tilde{\leftarrow}} B \equiv \tilde{\neg}\, B \mathbin{\tilde{\vee}} A$. For example, if the values of the ground atoms a and b are 0.7 and 0.5, respectively, then based on equations 4.2 and 4.3 the value of $a \mathbin{\tilde{\wedge}} b$ is 0.2 and the value of $a \mathbin{\tilde{\vee}} b$ is 1.

In order to perform inference over a ground network, the following probabilistic distribution is assumed over the network:

$$P(\mathbf{Y}|\mathbf{X}) = \frac{1}{\mathbf{Z}} \exp\left[-\sum_{r \in R} w_r d_r \Big(I(y_r, x_r) \Big)^p \right] \tag{4.5}$$

Where $\mathbf{Y}$ is the set of query ground atoms, $\mathbf{X}$ is the set of evidence atoms, $\mathbf{Z}$ is a normalizing constant that integrates over all possible assignments, R is a set of ground rules , w_r is the weight corresponding to the ground rule $r \in R$, x_r is the evidence ground atoms in r, and y_r is the query ground atoms in r. Given a set of ground atoms $\mathcal{A} = \{a_1, \ldots, a_n\}$ and a set of ground rules $R = \{r_1, \ldots, r_n\}$, an interpretation function I assigns truth values to atoms in $\mathcal{A}$ and rules in R. Note that the interpretation of a ground rule is computed using formulas 4.2 – 4.4. Given a ground rule r and its interpretation $I(r)$, the function d_r defines the distance-to-satisfaction for the given ground rule as follows:

$$d_r = 1 - I(r) \tag{4.6}$$

For example, if `SimTitle(d1.Title, d2.Title)` is 0.7, the truth value of {`d1.Author`} `SimSetEq`[`SimAuthor`] {`d2.Author`} is 1.0, and `SameDocument(d1,d2)` is assigned the value 0.4, then the truth value of the

Algorithm 4.1 MAP inference in PSL (as quoted from [Bröcheler et al., 2010])

```
1: $I_0(\mathbf{Y}) \leftarrow$ init. unknown to zeros
2: $R \leftarrow$ intital ground rules obtained by$I(\mathbf{X}) \cup I_0(\mathbf{Y})$
3: while $R$ has been updated do
4:     $i \leftarrow$ current iteration
5:     $O \leftarrow$ generateOptimizationProb($R$)
6:     $I_i \leftarrow$ optimize($O$)
7:     for $y \in \mathbf{Y}$ do
8:         if $I_i(y) > 0.01$ then
9:             $R_y \leftarrow$ ground rules containing $y$
10:            $R \leftarrow R \cup R_y$
11:        end if
12:    end for
13: end while
```

first rule in Listing 4.1 (i.e., the truth value assigned by the interpretation function I) is $min\{max\{0, 0.7 + 1.0 - 1\} + 0.4, 1\} = 0.7$, and the distance-to-satisfaction is 0.3. Note, that a rule is satisfied if the distance to satisfaction is 0. Finally, p, in the above probabilistic distribution, takes a value in $[1, 2]$. It defines a penalty for not satisfying a rule. In most SRL approaches the penalty for not satisfying ground a ground rule is linear (i.e., $p = 1$) [Bröcheler et al., 2010]. A squared penalty, however, provides a better trade-off between different forms of potentially conflicting forms evidence [Bach et al., 2015], which is our desired goal in using PSL, so we adopt it.

4.1.3 PSL Inference

The inference process in PSL takes as input a PSL program, a specification of both query and evidence predicates and observed truth values of ground evidence atoms. The output of the inference process is the inferred probabilities of ground query atoms.

The inference process in PSL is a type of maximum *a posteriori* (MAP) inference, where the goal is to find a joint assignment of truth values to a set of query predicates, given the observed values of the evidence predicates. For instance, in our running example, we would like to predict which documents are the same given the titles and the authors of each document. In PSL, inference is tantamount to finding the truth values of ground query atoms that minimize the distance-to-satisfaction of ground rules. This is formulated as the following optimization problem:

$$\begin{aligned}\arg\max_{y} P(\mathbf{Y}|\mathbf{X}) &= \arg\max_{y\in[0,1]} \left[-\sum_{r\in R} w_r d_r \Big(I(y_r, x_r) \Big)^p \right] \\ &= \arg\min_{y\in[0,1]} \left[\sum_{r\in R} w_r d_r \Big(I(y_r, x_r) \Big)^p \right]\end{aligned} \tag{4.7}$$

The ground network is transformed into a convex optimization problem whose solution provides the truth assignment of the ground query atoms. In contrast with the sampling inference approach that is implemented over MLNs, PSL's optimization approach to inference does not require the full ground network to be constructed at once [Bach et al., 2015; Bröcheler et al., 2010]. PSL's optimization approach makes an implicit assumption that, by default, all ground atoms have zero value, unless there is evidence which suggests otherwise. It then performs a lazy grounding technique that grounds atoms that have non-zero values. Adopting this technique mitigates the problem of memory footprint, which is one of the reasons for the more efficient inference in PSL compared to Markov logic, in which, by default, all ground atoms have probability 0.5. MLNs inference first generates the full ground network, then adjusts the initial probability as more evidence is provided. Sampling over a full ground MLN is less efficient than using PSL's optimization approach [Beltagy et al., 2014]. Having an efficient inference process is, of course, a desirable property because it allows us to scale the rule base with additional rules and types of evidence compared to what would be possible with MLNs.

PSL's inference procedure is summarized in Algorithm 4.1. To illustrate how the inference procedure works, assume the PSL program given in Listing 4.2. Assume the set {`X(a) = 1, X(b) = 0.8, Z(a,b) = 1.0`} as evidence, and the query predicate is `Q(a,b)`. The changes in the ground network as the inference proceeds are shown

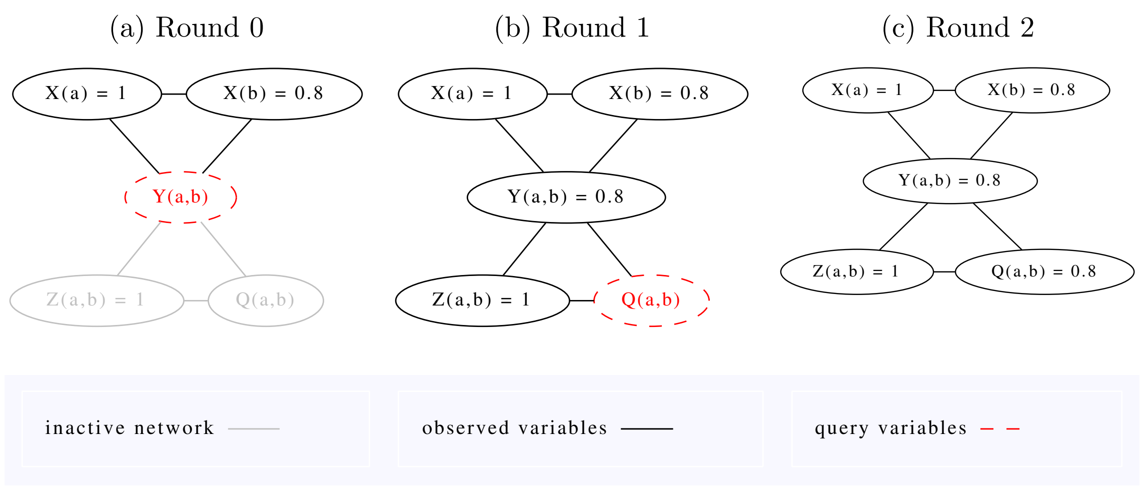


Figure 4.3: An example of PSL's iterative inference process

```
w1:  Y(A,B)   ←̃  X(A)  ∧̃  X(B)
w2:  Q(A,B)   ←̃  Y(A,B)  ∧̃  Z(A,B)
C : ¬̃Y(A,A) ←̃  X(A)
```

Listing 4.2: A simple PSL program used for illustrating PSL inference algorithm

in Figure 4.3. As per Algorithm 4.1, the initial ground network (round 0 in Fig. 4.3) contains single activated ground rule, i.e, the first rule in Listing 4.2. The unknown variable in this rule is `Y(a,b)`. After running the optimization step, Line 6 in Algorithm 4.1, which infers that `Y(a,b)` $= 0.8$, the second rule gets activated and the ground network now contains two ground nodes that corresponds to both rules in Listing 4.2. Given the change to the ground network, the optimization step is executed again, this time inferring a value of 0.8 for the query node `Q(a,b)`. Since the second execution of the optimization step does not activate any new rules, the algorithm terminates with the assignments shown in Round 2 in Figure 4.3.

4.1.4 PSL Weight Learning

The PSL weight learning takes as input a training dataset and an unweighted PSL program and it computes the optimal weights of the PSL rules by maximizing the likelihood of the training data. As in the inference process, the weight learning task takes as input a specification of both evidence and query predicates. The difference is that the query ground atoms should be instantiated.

The goal of the weight learning task in PSL is to assign weights to each rule in the rule base so as to maximize the probability of the training data, i.e., the probability of ground query predicates given the evidence. In PSL, this requires finding the partial derivative of the log-likelihood of the training data with respect to the weights, as follows:

$$\frac{\partial}{\partial W_r} \log P(\mathbf{Y}|\mathbf{X}) = \mathbb{E}_W \big[\Phi(\mathbf{Y}|\mathbf{X})\big] - \Phi(\mathbf{Y}|\mathbf{X}) \tag{4.8}$$

where

$$\Phi(\mathbf{Y}|\mathbf{X}) = \sum_{r \in R} w_r d_r \Big(I(y_r, x_r)\Big)^p \tag{4.9}$$

and $\mathbb{E}_W$ is the expectation of the distance-to-satisfaction according to the current estimate of parameters W. Since computing the expectation is intractable in general [Kimmig et al., 2015], PSL's maximum pseudo-likelihood estimation (MPLE) learning method approximates the expectation by computing the pseudo-likelihood through conditioning the values of query predicates Y on their immediate neighbours, i.e., values of X which appear in the same ground rule [Bach et al., 2015]. Another style of learning that is supported by PSL is called Large-Margin Estimation (LME). This style of learning drops the probabilistic interpretation of the PSL model and views inference as a prediction problem. The learning task focuses on finding weights W_r that produce highly accurate predictions given the training data. This learning approach is useful, as shown in Section 4.3.4, for learning weights that enable accurate classification of resources based on their RDF annotations. PSL also supports an expectation maximization (EM) style of learning. EM is a standard learning method where the data is not fully observed [A. P. Dempster, 1977]. In PSL, expectation maximization alternates between running inference on the query predicates and updating the weights of the rules. As we will see in Section 4.3, we use the EM learning style to learn the weights of feedback rules. The reason for adopting EM for learning the weights of feedback rules is because we make an assumption that feedback is obtained on a sample of the inferred results. Thus, the values of the feedback predicates (see Section 4.2.3) are not fully observed. Hence the use of EM learning in this case.

4.2 Integrating LD search results using PSL

We now describe our PSL-based approach for integrating LD search results. To guide the discussion that follows, we will use the RDF from of the search results shown in Figure 4.5.

Our approach can be summarized as follows. Firstly, as in our MLN approach, we define a meta-model (shown in Figure 4.4) that characterizes the inferred structure of the target reports. The main types in our meta-model are `Entity`, `EntityType`, `Property`, and `PropertyValue`. The main difference between this an the previous model (in Figure 3.5) is that it reflects the PSL capability of defining similarity relationships over soft truth values that Markov logic lacks. It extends the previous meta-model with similarity relationships (i.e., `SimEntity`, `SimEntityType`, `SimProperty` and `SimPropertyValue`) between instances of the meta-types in order to identify duplications in the results. Similar to the meta-model shown in Figure 3.5, this meta-model defines a `HasType` relationship, which relates individuals in the extent of the tabular report to their inferred entity types. We have also defined a `HasProperty` relationship, which relates the properties of individuals to their inferred types. In addition, we define the `HasDomain` and `HasPropertyValue` relationships. The `HasDomain` relationship relates a property value (either a URI or a literal) to a property (a URI), where the first is a value and the second is a property that characterises the domain of the value. The `HasPropertyValue` relationship relates a value to an entity. The last two relationships are useful in

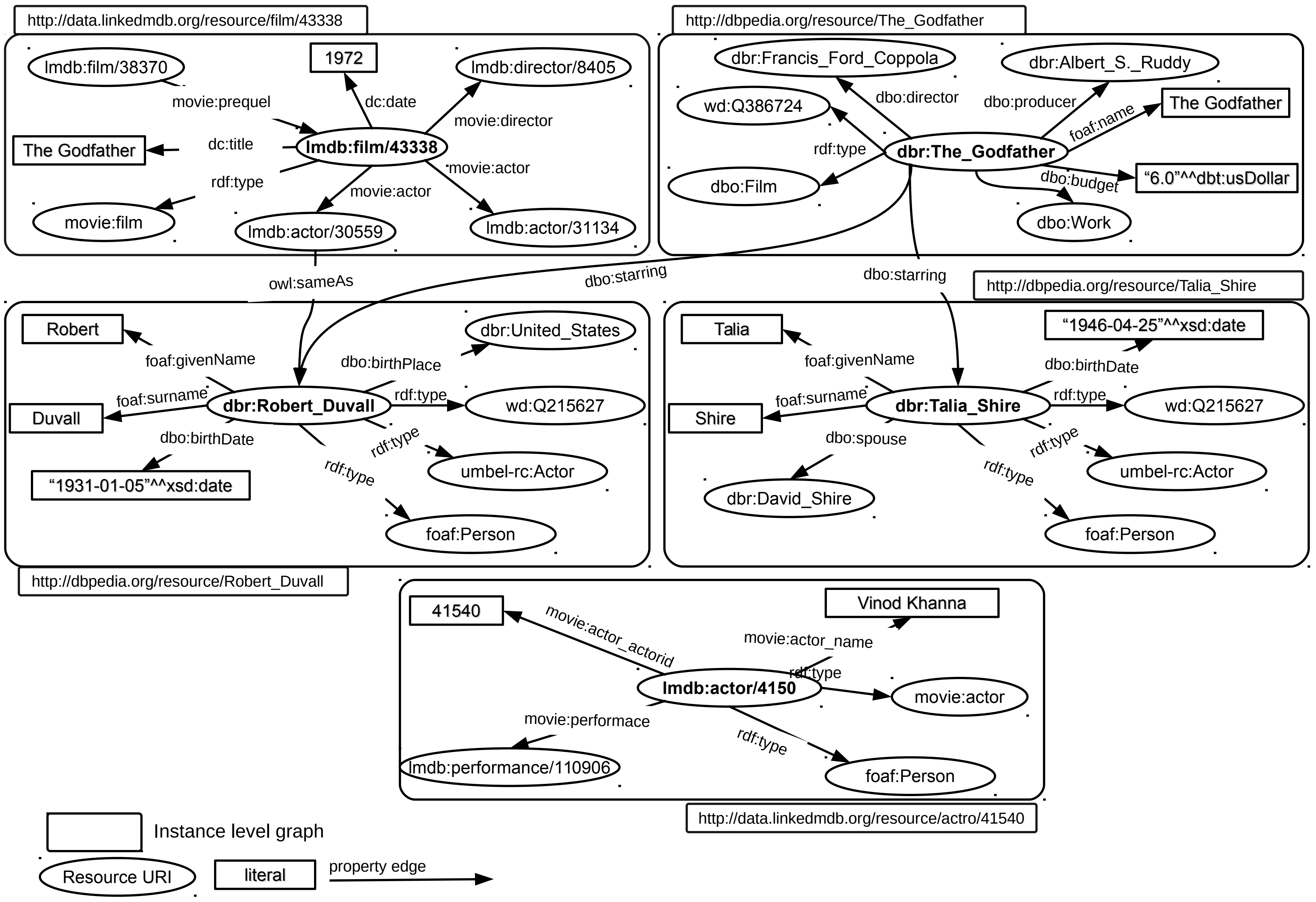


Figure 4.5: A RDF Sub-graph for some of the results returned for the search terms `Godfather actors`

modelling the similarity between instances of `Entity` and `Property`, as explained later in Section 4.2.1.

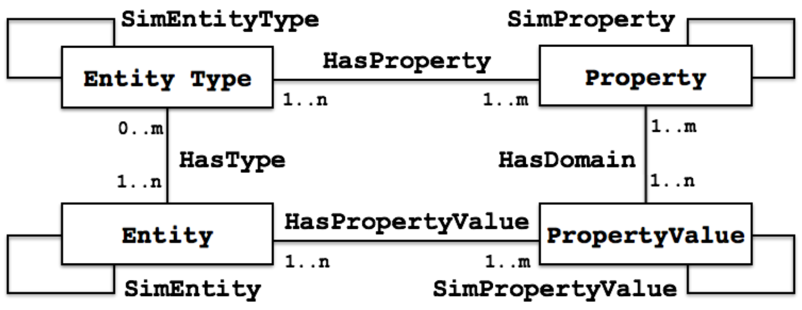


Figure 4.4: The meta-model for population with LD search results.

Next, we express the semantics of the meta-model using unweighted PSL rules. We define a set of rules that build on syntactic evidence alone. We refer to this set of rules as *the baseline model*, denoted by $\mathbb{B}$. In order to assimilate semantic evidence, we add rules to $\mathbb{B}$ that build on additional ontological evidence extracted from LD vocabularies. We refer to this model as *the semantic model*, denoted by $\mathbb{S}$, where $\mathbb{S} \supset \mathbb{B}$. Using the dataset compiled from LD search results (see Section 3.3.2) and a set of ontologies, we use a PSL implementation[1] to learn the weights of the rules in $\mathbb{S}$ and obtain the corresponding PSL program $PSL(\mathbb{S})$ This also yields $PSL(\mathbb{B})$, i.e., the subset of $PSL(\mathbb{S})$ which contains only those rules in $\mathbb{B}$. Then, given the returns of an LD search and domain ontologies, we use PSL inference from $PSL(\mathbb{S})$ to populate a meta-model with the most probable characterization of the returned results. In order to assimilate evidence from user feedback, we have defined a separate set of rules, which we refer to as the *feedback model*, denoted by $PSL(\mathbb{F})$. In order to obtain weights for rules in $PSL(\mathbb{F})$, we generated synthetic feedback based on the results of $PSL(\mathbb{S})$ and used the same PSL implementation. The PSL program obtained by $PSL(\mathbb{S}) \cup PSL(\mathbb{F})$, simply referred to as $PSL(\mathbb{F})$, integrates LD search results given syntactic, semantic and feedback evidence.

[1]`github.com/linqs/psl`

```
/*  Pre-processing RDF triples into mapped evidence predicates*/
C1 : RDFSubject(S)          ⇜ Triple1(S,P,O)
C2 : RDFSubject(S)          ⇜ Triple2(S,P,O)
C3 : RDFPredicate(P)        ⇜ Triple1(S,P,O)
C4 : RDFPredicate(P)        ⇜ Triple2(S,P,O)
C5 : RDFType(T)             ⇜ Triple1(S,"rdf:type",T)
C6 : RDFIsInstOf(S,T)       ⇜ Triple1(S,"rdf:type",T)
C7 : RDFSubjPredicate(S,P) ⇜ Triple1(S,P,O)
C8 : RDFSubjPredicate(S,P) ⇜ Triple2(S,P,O)
C9 : Label(S,O)             ⇜ Triple1(S,P,O) ∧̃
                               IsLabel(P)
C10: Label(S,O)             ⇜ Triple2(S,P,O) ∧̃
                               IsLabel(P)
C11: Name(S,O)              ⇜ Triple1(S,P,O) ∧̃
                               IsName(P)
C12: Name(S,O)              ⇜ Triple2(S,P,O) ∧̃
                               IsName(P)
```

Listing 4.3: Pre-processing constraints

4.2.1 The Baseline Model: Assimilating Syntactic Evidence

The key ethos of our approach is that we postulate uncertain relationships between RDF triples returned by the search results and instantiations of our meta-model using PSL rules. Correspondingly, the first step in the construction of a baseline model is to map, using syntactic evidence, RDF triples onto predicates that assert membership of meta-model constructs. To achieve this mapping, we employ three types of rules in the baseline model. First, we define a number of *pre-processing rules* as hard constraints to instantiate a mapped form of RDF triples. Second, we use RDF triples in their raw and mapped forms as evidence to define *membership rules* that infer meta-model predicates. Last, we define a number of similarity rules that enables de-duplicating of instances in our target meta-model.

Pre-processing Constraints

Raw RDF triples in the search results are represented using the evidence predicates `Triple1` and `Triple2`. This allows us to distinguish between triples that contain a URI from triples that contain a literal in the object position without the need to adding additional predicates that distinguish between values in the object position as

```
/*  Property rule using pre-processed triple predicates */
R1: Property(P) ⇜ RDFSubjPredicate(S,P) ∧̃ RDFIsInstanceOf(S,T)

/*  Property rule using raw triple predicates */
R2: Property(P) ⇜ Triple1(S, P, O) ∧̃ Triple1(S,"rdf:type",T)
```

Listing 4.4: Rules for inferring `Property` using triple and pre-processed relations

discussed in Section 3.3.1 . We further pre-process `Triple1` and `Triple2` predicates using the PSL constraints shown in Listing 4.3 to infer a mapped form of triples. For example, rule `C6` concludes that two URIs `S` and `T` are related by `RDFIsInstOf` based on the evidence of an RDF triple of the form `S rdf:type T`. Rules `C9` to `C12` utilize user-defined functions (i.e, `IsName` and `IsLabel`) to populate the `Name` and `Label` relations. These functions return 1.0 if the local name of the RDF property contains the keyword `label` or `name`. Examples of RDF properties that are matched using `C9` to `C12` rules in Figure 4.5 include: `rdfs:label`, `foaf:name`, `movie:actor_name`, and `geonames:name`. Their values often contain human-readable descriptions of RDF resources which are useful in finding descriptions of similar entities. In our experimental dataset, 92% of the resources are annotated with an RDF predicate that contains the `label` keyword, and 100% are annotated with an RDF predicate that contains the keyword `name`. Note that, by definition, PSL constraints are certain, hence the uncertainty described earlier is not addressed using the rules in Listing 4.3. We address the uncertainty of relationships between the RDF triples and instantiations of our meta-model using the rules in subsequent listings.

Pre-processing triples in this a way allows us to write shorter forms of subsequent rules, which leads to a smaller ground network, which makes the learning and inference processes more efficient. For example, Figure 4.6 shows the difference in the growth of the number of ground rules created for the rules in Listing 4.4 as the number of input triples increases. As shown in Figure 4.6, the footprint of the ground network in memory for a rule-base that uses `Triple` relations grows much faster than the rule-base which uses a pre-processed form of triples, which makes learning

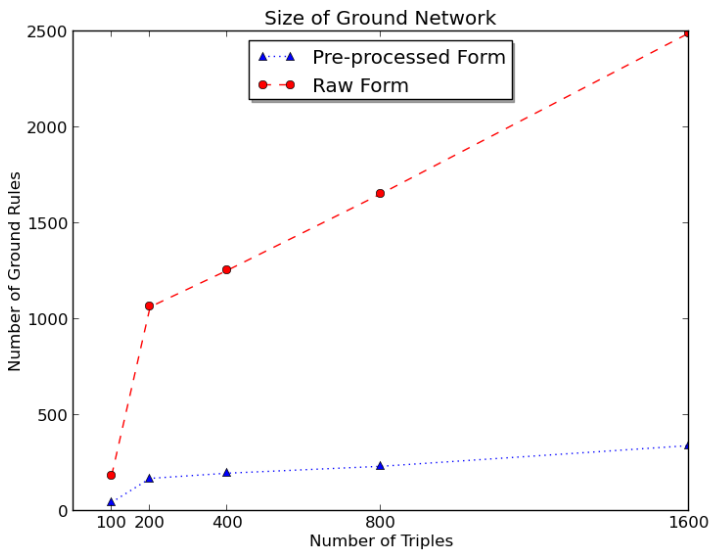


Figure 4.6: The effect of using pre-processed relations in the size of the ground network

an inference less efficient if we were to use triple relations only in our prospective rule-base.

In addition, pre-processing triples into binary relations between the constituents of RDF triples allows us to write rules that take advantage of PSL set aggregation. In PSL, using rules that aggregate over sets generates a ground Markov network with fewer nodes when compared to writing rules over elements of sets.

Membership Rules

The second type of rule in our baseline model is *membership rules*. As the name suggests, these rules express membership in some construct in the meta-model in Figure 4.4. Unlike pre-processing rules, membership rules are uncertain. The uncertainty is reflected in rule weights that are learned from sample data, as described in Section 4.3. As mentioned in Chapter 3, there is uncertainty because mapping RDF triples onto constructs in our meta-model only holds with some likelihood, and this, in turn, is because of non-uniform publication practices for LD resources and the uncertainty associated with the qualities of search results. An additional amount of

```
/*  Inference of meta-model instantiations */
R1: EntityType(T)          ⇜ RDFSubject(S) ∧̃
                              RDFIsInstOf(S,T)
R2: EntityType(T)          ⇜ RDFSubject(S) ∧̃
                              RDFIsInstOf(S,T) ∧̃
                              IsDictWord(T)
R3: Entity(S)              ⇜ RDFIsInstOf(S,T) ∧̃
                              EntityType(T)
R4: Property(P)            ⇜ RDFSubjPred(S,P) ∧̃
                              Entity(S)
R5: PropertyValue(O)       ⇜ Triple1(S,P,O) ∧̃
                              Entity(S) ∧̃
                              Property(P)
R6: PropertyValue(O)       ⇜ Triple2(S,P,O) ∧̃
                              Entity(S) ∧̃
                              Property(P)
R7: HasPropertyValue(S,O) ⇜ Triple1(S,P,O) ∧̃
                              Entity(S) ∧̃
                              Property(P)
R8: HasPropertyValue(S,O) ⇜ Triple2(S,P,O) ∧̃
                              Entity(S) ∧̃
                              Property(P)
R9: HasDomain(O,P)         ⇜ Triple1(S,P,O) ∧̃
                              Entity(S) ∧̃
                              Property(P)
R10: HasDomain(O,P)        ⇜ Triple2(S,P,O) ∧̃
                              Entity(S) ∧̃
                              Property(P)
R11: HasType(S,T)          ⇜ RDFIsInstOf(S,T) ∧̃
                              EntityType(T)
R12: HasProperty(T,P)      ⇜ RDFPredicate(P) ∧̃
                              RDFType(T) ∧̃
                              {P.RDFSubjPred(inv)} SimSetEq[URI]
                              {T.RDFIsInstOf(inv)}
```

Listing 4.5: Membership rules

uncertainty stems from using syntactic matchers to infer similarity between instances of constructs in our meta-model.

The bodies of the rules in Listing 4.5 show how RDF triples in their raw form (e.g., `R5` using a `Triple1` predicate) or in mapped form (e.g., `R2` using an `RDFIsInstOf` predicate) provide evidence for meta-model membership predicates. For example in Rule `R1`, a returned RDF triple `S rdf:type T` counts as evidence that `T` is an entity type. In the baseline model, as in our MLN approach, atoms that appear in the head (and, hence, are inferred) may appear in rule bodies as further evidence. This allows us to propagate uncertain inferences using the structure of the rule base. User-defined functions also count as evidence in our model. For example, in Rule `R2`, the function `IsDictWord` provides additional evidence that supports the inference of entity types. Consider again the example search results in Figure 4.5. Given the membership rules of the baseline model, and assuming that the rules have equal weights, `EntityType(Movie)` would be inferred with more probability than `EntityType(Q386724)` because the former satisfies Rules `R1` and `R2`, whereas, `Property(birthDate)` would be more probable than `Property(spouse)` because the latter is a predicate of fewer resources.

To infer the `HasProperty` relationship between RDF types and predicates, `R12` utilizes PSL set similarity (i.e., using `SimSetEq`[`URI`]) constructs. Figure 4.7 illustrates the intuition behind the `HasProperty` rule. The idea is that a `HasProperty` relationship is more likely when RDF types and predicates co-occur in a given set of resources. This co-occurrence in RDF resources is computed using set similarity constructs in PSL. Based on our running example, the probability of `HasProperty(Person, surname)` is greater than the probability of `HasProperty(Person, spouse)` because `surname` is a predicate of more resources than `spouse`.

In the problem at hand, note that the inference of the `HasProperty` relationship is akin to inferring OWL domain restriction axioms from LD search results. In here, we use the co-occurrence between candidate properties and types as a heuristic for the inference of this relation. The intuition behind using co-occurrence to infer the `HasPropoerty` relationship is illustrated in the following example based on the DBpedia ontology. In DBpedia, the RDF properties `dbo:academyAward` and `dbo:birthDate` both co-occur with the RDF type `dbo:Person` in a number of resources. The property `dbo:academyAward` co-occurs with the type `dbo:Person` only in 48 resources, whereas `dbo:birthDate` co-occurs with the same RDF type in $87,872$ resources. This indicates that the type `dbo:Person` is a more suitable domain restriction to the property `dbo:birthDate` than `dbo:academyAward`. In fact

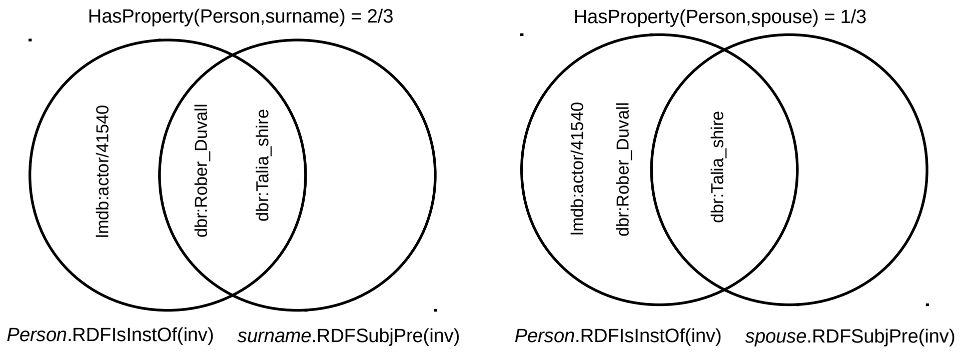


Figure 4.7: An illustration of the intuition of using set similarity to infer the `HasProperty` relation.

the DBpedia ontology explicitly restricts the domain of `dbo:birthDate` to by the type `dbo:Person`, whereas the domain of the property `dbo:academyAward` is the RDF type `dbo:Artist`.

Last, we define a prior rules for each query predicate in our PSL model to indicate lower prevalence of meta-model constructs to URIs and literals in the absence of evidence. As additional evidence is assimilated, the prior will have less effect in the prediction of the meta-model construct.

Similarity Rules

In order to identify semantic correspondences between the instances of the meta-types (i.e., `Entity`, `EntityType`, `Property` and `PropertyValue`), we define rules to infer similarity relationships. The bodies of the rules in Listing 4.6 show how the similarity relationships can be inferred from user-defined predicates (such as `LexSim` in, among others, Rule `R13`) and from user-provided definitions of set similarity (such as `SimSetEq`[] in, among others, Rule `R13`). These rules assimilate evidence obtained from syntactic similarity scores to derive the probability of similarity relationships. The advantage of using a declarative approach to infer similarity is that it allows us to incorporate different matching functions. In our model, we compute a value for `LexSim` using cosine similarity for strings with length greater than fifty and

```
R13: SimPropertyValue(V1,V2) ←̃ PropertyValue(V1) ∧̃
                                PropertyValue(V2) ∧̃
                                LexSim(V1,V2)
R14: SimEntityType(T1,T2)    ←̃ EntityType(T1) ∧̃
                                EntityType(T2) ∧̃
                                LexSim(T1,T2)
R15: SimEntityType(T1,T2)    ←̃ EntityType(T1) ∧̃
                                EntityType(T2) ∧̃
                                {T1.HasProperty}
                                  SimSetEq[SimProperty]
                                {T2.HasProperty}
R16: SimEntity(E1,E2)        ←̃ Entity(E1) ∧̃
                                Entity(E2) ∧̃
                                Label(E1,L1) ∧̃
                                Label(E2,L2) ∧̃
                                LexSim(L1,L2)
R17: ¬̃SimEntity(E1,E2)       ←̃ Entity(E1) ∧̃
                                Entity(E2) ∧̃
                                Label(E1,L1) ∧̃
                                Label(E2,L2) ∧̃
                                ¬̃LexSim(L1,L2)
R18: SimEntity(E1,E2)        ←̃ Entity(E1) ∧̃
                                Entity(E2) ∧̃
                                Name(E1,N1) ∧̃
                                Name(E2,N2) ∧̃
                                LexSim(N1,N2)
R19: ¬̃SimEntity(E1,E2)       ←̃ Entity(E1) ∧̃
                                Entity(E2) ∧̃
                                Name(E1,N1) ∧̃
                                Name(E2,N2) ∧̃
                                ¬̃LexSim(N1,N2)
R20: SimEntity(E1,E2)        ←̃ Entity(E1) ∧̃
                                Entity(E2) ∧̃
                                {E1.HasPropertyValue}
                                  SimSetEq[SimPropertyValue]
                                {E2.HasPropertyValue}
R21: SimProperty(P1,P2)      ←̃ Property(P1) ∧̃
                                Property(P2) ∧̃
                                LexSim(P1,P2)
R22: SimProperty(P1,P2)      ←̃ Property(P1) ∧̃
                                Property(P2) ∧̃
                                {P1.HasDomain(inv)}
                                  SimSetEq[SimPropertyValue]
                                {P2.HasDomain(inv)}
```

Listing 4.6: Similarity rules

Levenshtein (or edit) distance otherwise.

Given that resources that occur in search results are often described using different heterogeneous vocabularies, traditional entity resolution techniques cannot

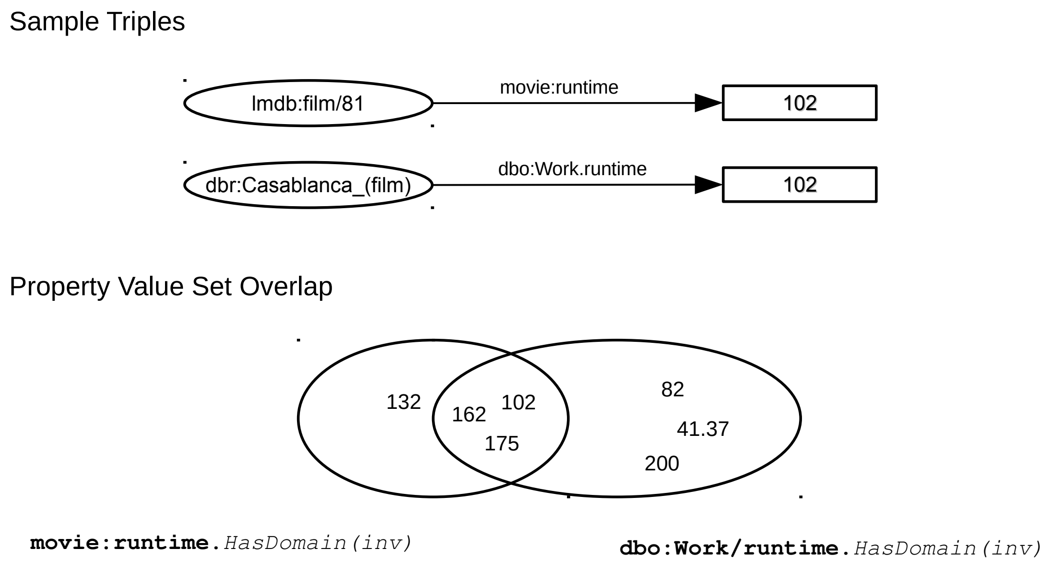


Figure 4.8: Property matching using overlapping sets of object values.

be used because they assume aligned vocabulary elements (i.e., types and properties) [Elmagarmid et al., 2007]. We do not have prior evidence of alignment between vocabulary elements (except for a few cases as discussed in Section 4.2.3). Thus, Rules `R16`-`R19` use the `Name` and `Label` attributes as evidence of similarity. Rule `R20` treats an entity as a set of property values, and infers similarity between entities from the overlap of property value sets. Rule `R22` uses object overlap (a metric that is commonly used to align properties [Zapilko and Mathiak, 2014; Gunaratna et al., 2013]) to infer similarity of properties. In the case of `SimProperty`, the set similarity predicate (i.e., `SimSetEq`[]) is parametrized by the values in the domain of the `HasDomain` relation, as illustrated in Figure 4.8

4.2.2 The Semantic Model: Adding Ontological Evidence

The baseline model only assimilates syntactic evidence. We can extend it with rules that assimilate evidence from ontologies to yield the model we call semantic. Ontologies are computational artefacts that formally describe concepts and relationships

Ontological Statement	PSL Predicate
T rdfs:label L	Label(T,L)
P rdfs:label L	Label(P,L)
T rdf:type rdfs:Class T rdf:type owl:Class	OntType(T)
P rdf:type rdf:Property P rdf:type owl:DatatypeProperty P rdf:type owl:ObjectProperty	OntProperty(P)
T1 owl:equivalentClass T2	OntEqType(T1,T2)
P1 owl:equivalentProperty P2	OntEqProperty(P1,P2)
T1 owl:disjointWith T2 T1 owl:complementOf T2	OntDisjointType(T1,T2)
P rdfs:domain T	OntHasProperty(T,P)

Table 4.1: Mapping ontological statements into PSL predicates

in a given domain. Thus, in terms of the meta-model we target, they describe entity types and their properties and provide evidence that complements the baseline model described above. Our approach is to make use of statements in ontologies about types and properties. This is then used to ground predicates that represent ontological evidence. Ontological evidence from LD vocabularies can be classified as: (i) syntactic knowledge in the form of text that is used to annotate an ontology construct, such as labels (e.g., `rdfs:label`) and comments (e.g, `rdfs:comment`); and (ii) semantic knowledge in the form of relationships between ontological constructs such as RDF class subsumption (e.g., `rdfs:subClassOf`) and equivalence (e.g., `owl:equivalentClass`). Table 4.1 shows the different kinds of evidence we extract from ontologies and how it is mapped into PSL predicates.

As we have done in the baseline model, we define a number of PSL constraints, i.e., rules that are assigned infinite weights during inference, (see Listing 4.8) that populate PSL predicates representing ontological evidence from triples found in LD vocabularies. Some of these constraints (e.g., rules `C30-C38`) are inspired by the *ter Horst* RDFS/OWL entailment rules [ter Horst, 2005a]. These constraints allows us to infer ontological evidence beyond what is explicitly defined by the vocabularies. In other words, we use these rules to instantiate a number of unary (e.g., `OntType`) and binary (e.g., `OntEqType`) predicates which are used in subsequent rules to assimilate ontological evidence. For example, if the triples in the Listing 4.7 are defined in an input vocabulary, in addition to evidence drawn from explicit relations given by the triples, Rule `C37` in Listing 4.8 adds, among others, `OntDisjointTypes(dbo:Film, dbo:Actor)` to the pool of ontological evidence.

```
dbo:Film          rdfs:subClassOf  dbo:CreativeWork .
dbo:Actor         rdfs:subClassOf  dbo:Person .
dbo:CreativeWork  owl:disjointWith dbo:Person
```

Listing 4.7: An example of ontological triples

```
C13: Label(T,L)                ←̃ OntTriple(T,"rdfs:label",L) ∧̃
                                 OntType(T)
C14: Label(P,L)                ←̃ OntTriple(P,"rdfs:label",L) ∧̃
                                 OntProperty(P)
C15: OntType(T)                ←̃ OntTriple(T,"rdf:type","rdfs:Class")
C16: OntType(T)                ←̃ OntTriple(T,"rdf:type","owl:Class")
C17: OntProperty(P)            ←̃ OntTriple(P,"rdf:type","rdf:Property")
C18: OntProperty(P)            ←̃ OntTriple(P,"rdf:type","owl:DatatypeProperty")
C19: OntProperty(P)            ←̃ OntTriple(P,"rdf:type","owl:ObjectProperty")
C20: OntAnnotProperty(AP)      ←̃ OntTriple(AP,"rdf:type","owl:AnnotationProperty")

C21: OntHasProperty(T,P)       ←̃ OntTriple(P,"rdfs:domain",T)
C22: OntHasProperty(T,P)       ←̃ OntTriple(P,"rdfs:domain",ListID) ∧̃
                                 InList(ListID,T)
C23: OntEqType(C1,C2)          ←̃ OntTriple(C1,"owl:equivalentClass",C2)
C24: OntEqProp(P1,P2)          ←̃ OntTriple(P1,"owl:equivalentProperty",P2)
C25: OntSubType(C1,C2)         ←̃ OntTriple(C1,"rdfs:subClassOf",C2)
C26: OntSubProperty(P1,P2)     ←̃ OntTriple(P1,"rdfs:subPropertyOf",P2)
C27: OntDisjointTypes(C1,C2)   ←̃ OntTriple(C1,"owl:disjointWith",C2)
C28: OntDisjointTypes(C1,C2)   ←̃ OntTriple(C1,"owl:complementOf",C2)
C29: OntDisjointProps(C1,C2)   ←̃ OntTriple(C1,"owl:propertyDisjointWith",C2)
C30: OntDisjointTypes(B,A)     ←̃ OntDisjointTypes(A,B)
C31: OntEqType(B,A)            ←̃ OntEqTypes(A,B)
C32: OntDisjointTypes(A2,B2)   ←̃ OntEqType(A1,A2) ∧̃
                                 OntEqType(B1,B2) ∧̃
                                 OntDisjointType(A1,B1)
C33: OntDisjointType(B,C)      ←̃ OntEqType(A,B) ∧̃
                                 OntDisjointType(A,C)
C34: OntSubType(A1,A3)         ←̃ OntSubType(A1,A2) ∧̃
                                 OntSubType(A2,A3)
C35: OntEqType(A,B)            ←̃ OntSubType(A,B) ∧̃
                                 OntSubType(B,A)
C36: OntEqProp(A,B)            ←̃ OntSubProperty(A,B) ∧̃
                                 OntSubProperty(B,A)
C37: OntDisjointTypes(AS,BS)   ←̃ OntSubType(AS,A) ∧̃
                                 OntSubType(BS,B) ∧̃
                                 OntDisjointTypes(A,B)
C38: OntDisjointProps(AS,BS)   ←̃ OntSubProperty(AS,A) ∧̃
                                 OntSubProperty(BS,B) ∧̃
                                 OntDisjointTypes(A,B)
```

Listing 4.8: Ontology pre-processing constraints

In addition to constraints that populate PSL ontological predicates, we define a set of rules that assimilate ontological evidence. The main idea is that being defined as a concept or a property in some pertinent ontology counts as additional evidence that the returned result (e.g, resource URI) is an entity or property, respectively, in

```
R23: Entity(S)              ⇜ RDFIsInstanceOf(S,OT) ∧̃
                               OntType(OT)
R24: Entity(S)              ⇜ RDFIsInstanceOf(S,T) ∧̃
                               OntType(OT) ∧̃
                               LexSim(T,OT)
R25: Entity(S)              ⇜ RDFIsInstanceOf(S,T) ∧̃
                               OntEqTypes(T,OT)
R26: Property(OP)           ⇜ Entity(S) ∧̃
                               RDFSubjPredicate(S,OP) ∧̃
                               OntProperty(OP)
R27: Property(P)            ⇜ Entity(S) ∧̃
                               RDFSubjPredicate(S,P) ∧̃
                               OntProperty(OP) ∧̃
                               LexSim(P,OP)
R28: Property(P)            ⇜ Entity(S) ∧̃
                               RDFSubjPredicate(S,P) ∧̃
                               OntEqProps(P,OP)
R29: EntityType(OT)         ⇜ RDFIsInstanceOf(S,OT) ∧̃
                               OntType(OT)
R30: EntityType(T)          ⇜ RDFIsInstanceOf(S,T) ∧̃
                               OntType(OT) ∧̃
                               LexSim(T,OT)
R31: EntityType(T)          ⇜ RDFIsInstanceOf(S,T) ∧̃
                               OntEqTypes(T,OT)
R32: HasProperty(OT,OP)     ⇜ RDFSubjPredicate(S,OP) ∧̃
                               RDFIsInstanceOf(S,OT) ∧̃
                               OntHasProperty(OT,OP)
R33: HasProperty(T,OP)      ⇜ RDFSubjPredicate(S,OP) ∧̃
                               RDFIsInstanceOf(S,T) ∧̃
                               OntHasProperty(OT,OP) ∧̃
                               LexSim(T,OT)
R34: HasProperty(OT,P)      ⇜ RDFSubjPredicate(S,P) ∧̃
                               RDFIsInstanceOf(S,OT) ∧̃
                               OntHasProperty(OT,OP) ∧̃
                               LexSim(P,OP)
R35: HasProperty(T,P)       ⇜ RDFSubjPredicate(S,P) ∧̃
                               RDFIsInstanceOf(S,T) ∧̃
                               OntHasProperty(OT,OP) ∧̃
                               LexSim(P,OP) ∧̃
                               LexSim(T,OT)
R36: HasType(S,OT)          ⇜ RDFIsInstanceOf(S,OT) ∧̃
                               OntType(OT)
R37: HasType(S,T)           ⇜ RDFIsInstanceOf(S,T) ∧̃
                               OntType(OT) ∧̃
                               LexSim(T,OT)
R38: HasType(S,T)           ⇜ RDFIsInstanceOf(S,T) ∧̃
                               OntEqTypes(T,OT)
R39: SimProperty(P1,P2)     ⇜ Property(P1) ∧̃
                               Property(P2) ∧̃
                               OntEqProps(P1,P2)
R40: SimProperty(P1,P2)     ⇜ Property(P1) ∧̃
                               Property(P2) ∧̃
                               OntEqProps(OP1,OP2) ∧̃
                               OntSimURI(P1,OP1) ∧̃
                               OntSimURI(P2,OP2)
R41: SimProperty(P1,P2)     ⇜ Property(P1) ∧̃
                               Property(P2) ∧̃
                               OntProperty(OP) ∧̃
                               OntSimURI(P1,OP) ∧̃
                               OntSimURI(P2,OP)
R42: SimEntityType(T1,T2)   ⇜ EntityType(T1) ∧̃
                               EntityType(T2) ∧̃
                               OntEqType(T1,T2)
R43: SimEntityType(T1,T2)   ⇜ EntityType(T1) ∧̃
                               EntityType(T2) ∧̃
                               OntType(OT) ∧̃
                               LexSim(T1,OT) ∧̃
                               LexSim(T2,OT)
R44: SimEntityType(T1,T2)   ⇜ EntityType(T1) ∧̃
                               EntityType(T2) ∧̃
                               OntType(OT) ∧̃
                               {T1.HasProperty} SimSetEq[SimProperty]
                                   {OT.OntHasProperty} ∧̃
                               {T2.HasProperty} SimSetEq[SimProperty]
                                   {OT.OntHasProperty}
```

Listing 4.9: Rules the adds ontological evidence to our model

```
C39: ~¬SimEntityType(T1,T2) ~← EntityType(T1) ~∧
                                EntityType(T2) ~∧
                                OntDisjointTypes(T1,T2)
C40: ~¬SimEntityType(T1,T2) ~← EntityType(T1) ~∧
                                EntityType(T2) ~∧
                                OntDisjointTypes(OT1,OT2) ~∧
                                OntSimURI(T1,OT1) ~∧
                                OntSimURI(T2,OT2)
C41: ~¬SimEntity(E1,E2)     ~← Entity(E1) ~∧
                                Entity(E2) ~∧
                                HasType(E1,T1) ~∧
                                HasType(E2,T2) ~∧
                                OntDisjointTypes(T1,T2)
C42: ~¬SimEntity(E1,E2)     ~← Entity(E1) ~∧
                                Entity(E2) ~∧
                                HasType(E1,T1) ~∧
                                HasType(E2,T2) ~∧
                                OntDisjointTypes(OT1,OT2) ~∧
                                OntSimURI(T1,OT1) ~∧
                                OntSimURI(T2,OT2)
```

Listing 4.10: Additional constraints which the assimilates ontological evidence to our model

terms of our meta-model. Thus, we use the PSL predicates in Table 4.1 to construct the rules in Listing 4.9 and the constraints in Listing 4.10. For example, if `"The Godfather"` appears in a triple in the returned results as an instance of `Movie`, where `Movie` is asserted to be a concept(e.g., in Movie Ontology[2]) this adds strength to the belief that `"The Godfather"` is an instance of `Entity` in our meta-model (see rule `R23` in Listing 4.9). We also use ontological statements to supplement evidence for similarity relationships (rules `R39-R44`).

Rule `R42` exemplifies the direct use of ontological evidence. Rule `R43` exemplifies the direct use of ontological evidence mediated by lexical similarity. Rule `R44` exemplifies the use of set-similarity predicates where one of the sets contain elements from ontological statements. In this case, ontological statements help reconcile the heterogeneity in the returned results.

Finally, we define a number of constraints (`C39-C42`) that enforce any type disjointness defined in pertinent ontologies by negating candidate `SimEntity` and

[2]`www.movieontology.org/`

`SimEntityType` relations.

Subsuming the baseline model, the semantic model contains syntactic rules `R1-R22`, rules `R23-R44` that assimilate additional evidence to the predicates in the baseline model, and the constraints `C39-C42`.

4.2.3 The Feedback Model: Assimilating User-Provided Evidence

The semantic model assimilates syntactic and semantic evidence. We now extend it with rules that assimilate user-provided evidence in the form of feedback.

There has been a growing interest in assimilating user feedback in data integration (see [Belhajjame et al., 2011] for a general proposal, and [Paton et al., 2016] for a general methodology and recent work in the area). The use of feedback in data integration arose because of the inadequacy of classical approaches to address the integration of highly dynamic sources (e.g., Web resources), i.e., those that evolve rapidly [Paton et al., 2012]. Classical data integration approaches are often characterized as *labour-intensive* approaches, as they are heavily dependent on intensive expert manual effort, aimed at the construction of *high-quality* semantic mappings between data sources. Such approaches are effective but have proved to be too costly in Web environments [Halevy et al., 2006]. The *pay-as-you-go* approach has emerged as an attempt to achieve high-quality integration results at a reduced up-front cost. This approach involves the use of automated techniques to discover mappings between data sources followed by improvement that requires user feedback. The idea here is that the results of such automated techniques are sufficiently good to motivate users make use of the resulting integration without any initial up-front investment. The hope is that, through further interaction with the bootstrapped artefact, cost-effective feedback can be procured that enables the incremental improvement of the initial integration outcome. With recent advances in crowdsourcing systems [Doan et al., 2011], feedback for solving complex data integration tasks can be obtained more cost-effectively.

Another reason for assimilating user-provided feedback as evidence in the context of this dissertation is the intuition that combining ontological evidence and syntactic evidence may not be sufficient to provide for robust accurate inference of instances of our meta-model. For example, in our experiment, existing LOD vocabularies (i) seldom use expressive meta-vocabulary constructs, and (ii) seldom use cross-ontology links.

We surveyed 338 vocabularies (see Appendix B for detail on the survey methodology) obtained from the Linked Open Vocabularies[3] repository. Figure 4.10 provides an overview on the meta RDF terms used in the surveyed vocabularies. The x-axis gives a count of the distinct subject URIs for each RDF term shown in the y-axis. It suggests that the construct most used in LOD vocabularies is `rdf:type`. At the vocabulary-level, this construct is used to assert that an RDF term (i.e., RDF class or property) is a member of meta-classes defined by RDFS/OWL (e.g., `owl:Class` and `rdf:Property`). Vocabulary features, which our semantic evidence rules rely upon, such as domain restriction (i.e., `rdfs:domain`), equivalence relations (e.g, `owl:equivalentClass`) and disjointness axioms (e.g., `owl:disjointWith`) are much less frequent. The less expressive a vocabulary, the less likely it will contribute semantic evidence for the inference of instances of the target structure.

Our survey also shows a lack of reuse of ontologically defined terms. The total number of ontological statements in these vocabularies was about 200K. Only 8.5% of the statements are defined across ontologies (i.e, the subject and object of the statement belong to different namespaces). Figure 4.9 shows the meta-vocabulary features that are used to refer to external constructs. The y-axis shows the number of distinct triples for each meta-predicate shown in the x-axis. The figure compares the number of statements that are defined across (*grey bars*) versus the number of statements that are defined within (*dark bars*) ontologies. As shown in Figure 4.9,

[3]lov.okfn.org/dataset/lov/

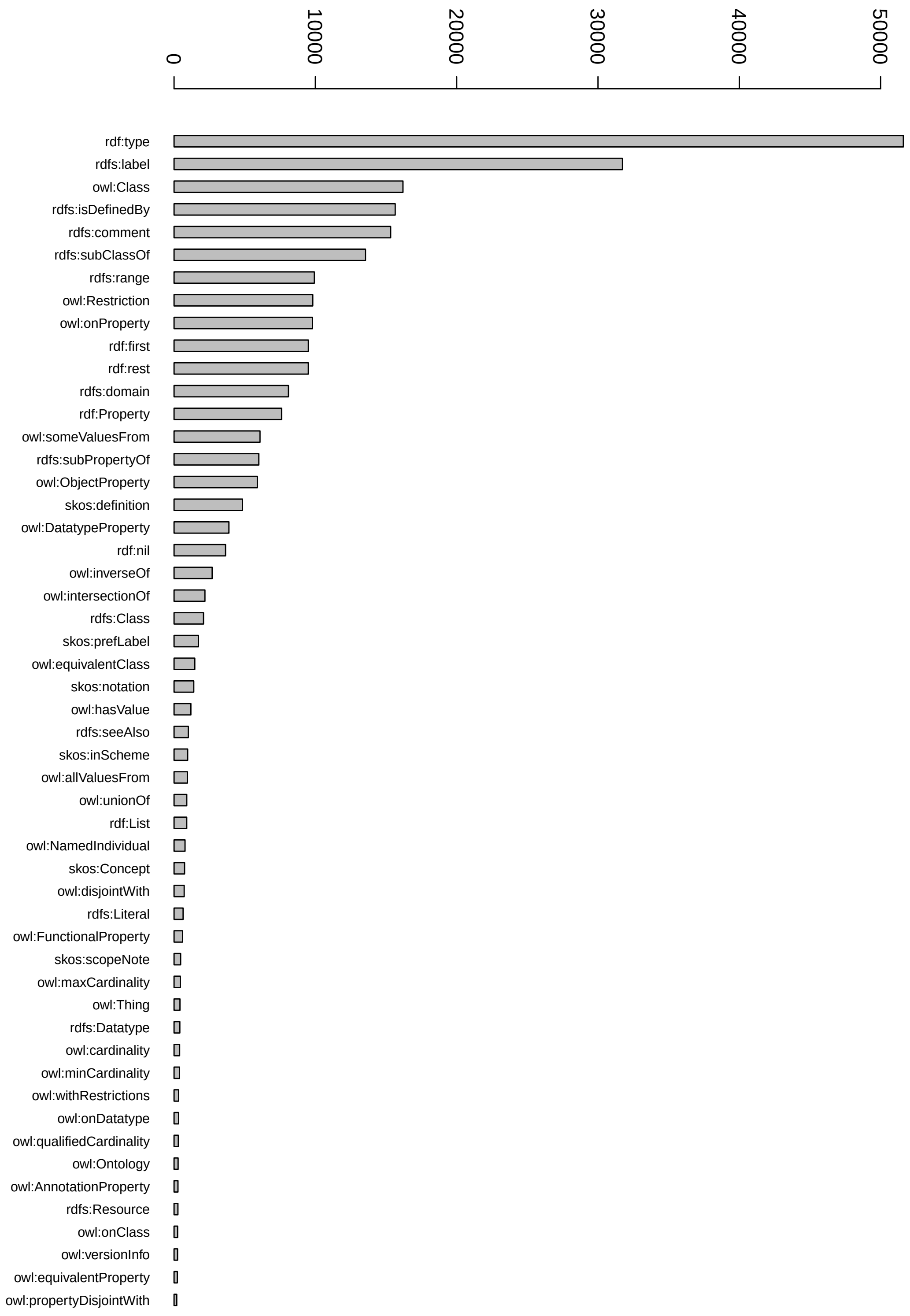


Figure 4.10: Vocabulary predicates in the surveyed ontologies

some features that we use to extract evidence from ontologies (e.g., `rdfs:domain` and `owl:disjointWith`) are often only defined between resources in the same namespace. As the search results comes for different datasets, existing ontological evidence may not be sufficient to align the RDF terms that occur in search results.

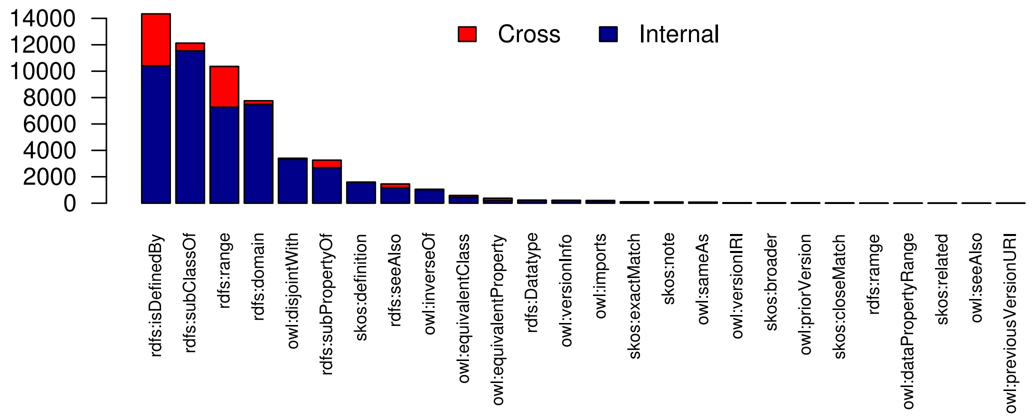


Figure 4.9: Meta-vocabulary predicates which are often used to link define cross-ontology links.

Given the above, we use feedback to supplement the syntactic and ontological evidence. In our approach we use feedback in two ways. The first is to take advantage of accumulated, user-provided feedback to improve the inference results before a report is produced. In this case, feedback on inference results is another type of evidence that can be assimilated. In our approach, we assume that the user knows the domain of the search term and is motivated by obtaining high quality results from the search, which we believe to be reasonable in the context of search results personalization. The second is to refine the report presented to the user. When the returned results have been structured and integrated using PSL, a report is presented to the user (as described in Chapter 5). Feedback can be provided that triggers the refinement of the report. For example, a prior state of the report shown in Figure 1.3 could have contained another row in the `Movie` table referring to the `Mumbai Godfather` film which the user rules out as a false positive. In this case, feedback on inference results underpins a form of filtering (i.e., data cleaning),

PSL Feedback Predicate	**Example**
`EntityTypeFB(uri,term)`	`EntityTypeFB(Q11424,no)`
`HasTypeFB(uri,uri,term)`	`HasType(dbr:The_Godfather,CreativeWork, yes)`
`HasPropertyFB(uri,uri,term)`	`HasPropertyFB(CreativeWork, author, yes)`
`SimEntityTypeFB(uri,uri,term)`	`SimEntityType(Film, Movie, yes)`
`SimPropertyFB(uri,uri,term)`	`SimPropertyFB(duration, runtime, yes)`
`SimEntityFB(uri,uri,term)`	`SimEntityFB(dbr:The_Godfather, lmdbr:film/43338, yes)`

Table 4.2: PSL predicates used to gather user feedback (with examples)

generating an incentive for the user to provide feedback in the first place.

Table 4.2 shows the feedback predicates used in our model. The feedback we consider here is simply a testimony from a user on the correctness of the inferred query atom. We use the PSL predicates in Table 4.2 to construct the PSL rules shown in Listing 4.11. For example, Rules `R45-R48` assimilate user-provided feedback as evidence that a given resource `S` is, or is not, an entity. For example, if `"War and Peace"` appears in a triple in the returned results as an instance of `Movie`, where `Movie` is confirmed by feedback to be an entity type, this adds strength to the belief that `"War and Peace"` is an instance of `Entity` in our meta-model. Given that the PSL program $PSL(\mathbb{F})$ subsumes the rules in the semantic model $\mathbb{S}$, inference made through the feedback rules is propagated to the semantic model rule-base. For example, inference from rules `R54` and `R55` is propagated to the set-similarity rule `R15`, given that the latter is parametrized by the `HasProperty` predicate. This creates a knock-on effect of the provided feedback that, in principle, makes it more cost-effective to collect.

Another use of feedback that could be considered, is the application of active learning techniques by presenting the inference results of query predicates to an expert for manual labeling. Then, feedback can be collected and used to do the weighting of rules. For example consider the rule-base in Listing 4.12. In this rule-base, one could solicit for feedback on the inferences of `SimPropertyValue`. Instances of such feedback can be captured using the evidence predicate `SimPropertyValueFB`. Using instances of this evidence predicate, we can relearn the weights of the second rule in Listing 4.12.

```
/*  Rules R45-63: Extending inference with feedback evidence */
R45: Entity(S)             ←̃ RDFIsInstanceOf(S,T) ∧̃
                              EntityTypeFB(T,UID,"yes")
R46: ¬̃Entity(S)            ←̃ RDFIsInstanceOf(S,T) ∧̃
                              EntityTypeFB(T,UID,"no")
R47: Entity(S)             ←̃ RDFIsInstanceOf(S,T) ∧̃
                              HasTypeFB(S,T,UID,"yes")
R48: ¬̃Entity(S)            ←̃ RDFIsInstanceOf(S,T) ∧̃
                              HasTypeFB(S,T,UID,"no")
R49: EntityType(T)         ←̃ EntityTypeFB(T,UID,"yes")
R50: ¬̃EntityType(T)        ←̃ EntityTypeFB(T,UID,"no")
R51: HasType(S,T)          ←̃ HasTypeFB(S,T,UID, "yes")
R52: ¬̃HasType(S,T)         ←̃ HasTypeFB(S,T,UID, "no")
R53: Property(P)           ←̃ HasPropertyFB(T,P,UID,"yes")
R54: ¬̃Property(P)          ←̃ HasPropertyFB(T,P,UID,"no")
R55: HasProperty(T,P)      ←̃ HasPropertyFB(T,P,UID,"yes")
R56: ¬̃HasProperty(T,P)     ←̃ HasPropertyFB(T,P,UID,"no")
R57: SimProperty(P1,P2)    ←̃ SimPropertyFB(P1,P2,UID,"yes")
R58: ¬̃SimProperty(P1,P2)   ←̃ SimPropertyFB(P1,P2,UID,"no")
R60: SimEntityType(T1,T2)  ←̃ SimEntityTypeFB(T1,T2,UID,"yes")
R61: ¬̃SimEntityType(T1,T2) ←̃ SimEntityTypeFB(T1,T2,UID,"no")
R62: SimEntity(E1,E2)      ←̃ SimEntityFB(E1,E2,UID,"yes")
R63: ¬̃SimEntity(E1,E2)     ←̃ SimEntityFB(E1,E2,UID,"no")
```

Listing 4.11: A rule-base that assimilates user feedback

```

W1: SimPropertyValue(V1,V2) ←̃ PropertyValue(V1) ∧̃
                              PropertyValue(v2) ∧̃
                              LexSimPropertyValue(V1,V2)

W2: SimEntity(E1,E2)       ←̃ Entity(E1) ∧̃
                           Entity(E2) ∧̃
                           HasPropertyValue(E1,V1) ∧̃
                           HasPropertyValue(E2,V2) ∧̃
                           SimPropertyValueFB(V1,V2)
```

Listing 4.12: A rule-base for demonstrating active learning in PSL

A different method of applying active learning in SRL is presented by Fisher et al. [2016]. In their approach they have used an active learning technique for generating training examples for an entity resolution problem. Their method also allowed domain experts to add new rules to an MLN based on feedback collected on the results of matching records. The new training examples as well new rules were used to modify the weights of an initial MLN.

4.2.4 Extending The Baseline Model With Domain Specific Rules

Successful applications of SRL approaches have made use of domain-specific knowledge to encode predicates and relations in a given model. Singla and Domingos [2006] used knowledge about the relationships between authors and papers in the *publication* domain to write rules that infer the equivalent entities in that domain using MLNs. Their approach used predicates that are based on specific attributes (e.g., `HasAuthor` and `HasVenue`) of publications. These predicates were then used to define rules that encoded knowledge about the relations between attributes to de-duplicate entities as exemplified in the rule shown in Listing 4.13.

```
w: SameAuthor(y3,y4)  ← HasAuthor(x1, y1) ∧
                        HasAuthor(x2, y2) ∧
                        HasAuthor(x1, y3) ∧
                        HasAuthor(x2, y4) ∧
                        SamePaper(x1, x2) ∧
                        SameAuthor(y1, y2)
```

Listing 4.13: A rule that encodes domain-specific knowledge, adopted from [Singla and Domingos, 2006]

Similarly, Niu et al. [2012] used domain-specific knowledge rules to populate a relational database from uncertain natural language extraction process. They used Markov logic rules to encode relations between sequences of word mentions and the output of named entity recognition (NER) tools to infer relationships between mentions as shown for example in Listing 4.14.

```
w_pat: KnownBirthPlace(per,loc) ← WordSequence(s,m1,m2,pat) ∧
                                   MentionPerson(per, m1) ∧
                                   MentionLocation(loc, m2)
```

Listing 4.14: An example domain-specific rule used in Elementary [Niu et al., 2012]

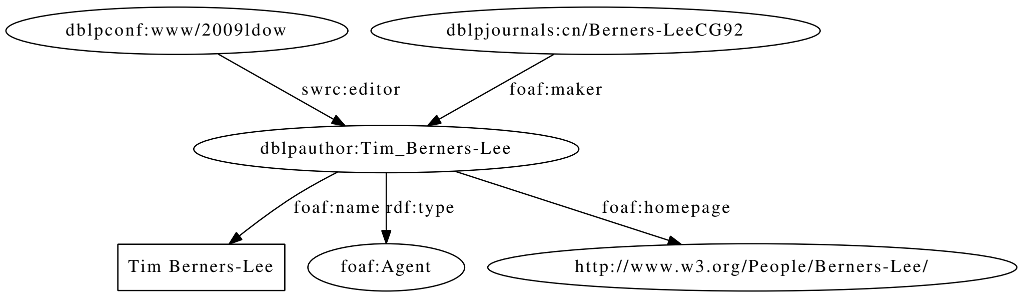


Figure 4.11: An RDF description of Tim Berners-lee in the DBLP dataset. The resource is annotated with `foaf:Agent` type. A fine-grained data-type could possibly be assigned based on the given description.

So far, our application of PSL has relied on domain-independent rules to superimpose a structure on the returned search results. However, our model can be extended with domain-specific rules. In particular, we introduce rules that are useful at identifying fine-grained entity types (see Definition 14) for a returned resource based on the RDF properties used to annotate the property values of the resource. This can be useful, for example, in resolving co-referent descriptions of entities in a heterogeneous dataset. For example, identifying that the resource depicted in Figure 4.11, which describes a person, belongs to the entity type *scientist*, in addition to being an instance of `foaf:Agent`, can facilitate merging this resource with other resources of a similar type (e.g., its DBpedia equivalent, i.e., `dbr:Tim_Berners-Lee`). To extend the baseline model with fine-grained type identification capabilities, we use a knowledge base for bootstrapping a set of rules that associate RDF properties with domain-specific entity types.

Definition 14 (Fine-grained Entity Type)**.** Given a triple $t = (s, \texttt{rdf:type}, c)$ and an RDF graph $\mathcal{D}$ containing t, the *RDF class* c is said to be a fine-grained entity type if it has no subtypes that are used in the graph $\mathcal{D}$, i.e., the RDF class c has no subclasses.

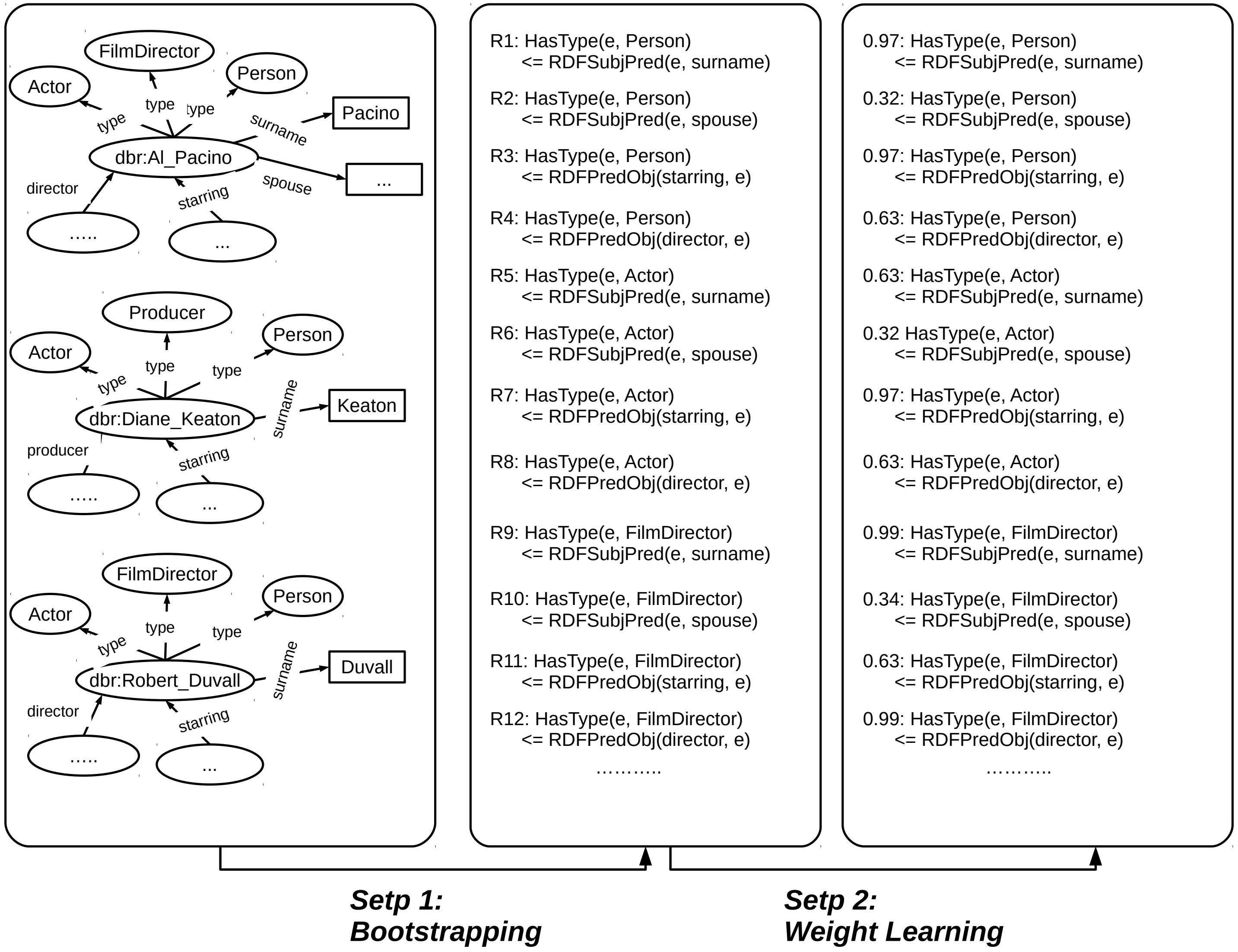
```
wt1: HasType(S, type) ⇜ RDFSubjPred(S, pred)
wt2: HasType(S, type) ⇜ RDFPredObj(pred, S)
```

Listing 4.15: Rule template used for bootstrapping domain-specific PSL rule-base

Example 3 (Fine-grained entity types)**.** Typically, resources in the WoD are annotated with more than one RDF class. For example, the resource `dbr:The_Godfather` is annotated (among others) with the classes `owl:Thing`, `dbo:Work`, `dbo:Film`. A fine-grained entity type for this resource is *film*. Note that a resource could have more than one fine-grained entity type. For example, the resource `dbr:Al_Pacino` describes a *person* who is both an *actor* and a *director*.

In our experiment, we have used the DBpedia dataset to bootstrap the rules. However, our approach is flexible enough to work with any well-defined knowledge base. The bootstrap rules instantiate the rule templates in Listing 4.15. The rule template w_{t1} associates an *outgoing* RDF property with an RDF resource type, whereas the template w_{t2} associates an *incoming* RDF property with an RDF resource type. Algorithm 4.2 shows the bootstrapping procedure of the rules from the templates shown in Listing 4.15. The process take as an input an RDF graph $\mathcal{D}$. For each typed individual in the input graph, the process find the sets of type, incoming property, and outgoing property labels associated with an individual. For each (type, property) pair, it generates a rule based on one of the above templates depending on whether the property is an incoming or an outgoing property. For example, for the RDF graph shown in Figure 4.11 it will generate the rules in Listing 4.16.

Note that the bootstrap rules are unweighted rules. We then use the weight learning process in PSL to learn how discriminating an RDF property is for a given type. Figure 4.12 shows the overall process used to learn domain-specific rules from an RDF dataset. The first step in the process bootstraps a rule-base, as explained earlier. In the second step, we use the weight learning process to learn the weight of each RDF property found in the RDF graph. For example, in Figure 4.12, based



R1: HasType(e, Person) <= RDFSubjPred(e, surname)	0.97: HasType(e, Person) <= RDFSubjPred(e, surname)
R2: HasType(e, Person) <= RDFSubjPred(e, spouse)	0.32: HasType(e, Person) <= RDFSubjPred(e, spouse)
R3: HasType(e, Person) <= RDFPredObj(starring, e)	0.97: HasType(e, Person) <= RDFPredObj(starring, e)
R4: HasType(e, Person) <= RDFPredObj(director, e)	0.63: HasType(e, Person) <= RDFPredObj(director, e)
R5: HasType(e, Actor) <= RDFSubjPred(e, surname)	0.63: HasType(e, Actor) <= RDFSubjPred(e, surname)
R6: HasType(e, Actor) <= RDFSubjPred(e, spouse)	0.32 HasType(e, Actor) <= RDFSubjPred(e, spouse)
R7: HasType(e, Actor) <= RDFPredObj(starring, e)	0.97: HasType(e, Actor) <= RDFPredObj(starring, e)
R8: HasType(e, Actor) <= RDFPredObj(director, e)	0.63: HasType(e, Actor) <= RDFPredObj(director, e)
R9: HasType(e, FilmDirector) <= RDFSubjPred(e, surname)	0.99: HasType(e, FilmDirector) <= RDFSubjPred(e, surname)
R10: HasType(e, FilmDirector) <= RDFSubjPred(e, spouse)	0.34: HasType(e, FilmDirector) <= RDFSubjPred(e, spouse)
R11: HasType(e, FilmDirector) <= RDFPredObj(starring, e)	0.63: HasType(e, FilmDirector) <= RDFPredObj(starring, e)
R12: HasType(e, FilmDirector) <= RDFPredObj(director, e)	0.99: HasType(e, FilmDirector) <= RDFPredObj(director, e)
………..	………..

Figure 4.12: The process of learning domain-specific rules. The first step bootstraps a rule base from an RDF graph. In the second step we use PSL weight learning process to learn the discriminability of RDF predicates

```
w1: HasType(S, "Agent") ⇜ RDFSubjPred(S, "name")
w2: HasType(S, "Agent") ⇜ RDFSubjPred(S, "homepage")
w3: HasType(S, "Agent") ⇜ RDFPredObj("maker", S)
w4: HasType(S, "Agent") ⇜ RDFPredObj("editor", S)
```

Listing 4.16: Bootstrap rule-base for the RDF graph shown in Figure 4.11

on the learned weights, the RDF property `surname` is more indicative of the type `Person` than of the type `Actor`.

Algorithm 4.2 bootstrapping domain-specific rules

```
1: function BOOTSTRAP($\mathcal{D}$)
2:     $I \leftarrow TypedIndividuals(\mathcal{D})$          ▷ Returns a set
3:     $R \leftarrow \emptyset$                               ▷ Set of bootstrap rules
4:     for $i \in I$ do
5:         $T \leftarrow Types(i)$                  ▷ Returns a set of type labels of the individual $i$
6:         $\overrightarrow{P} \leftarrow OutProperties(i)$      ▷ Returns a set of outgoing property labels
7:         $\overleftarrow{P} \leftarrow InProperties(i)$       ▷ Returns a set of incoming property labels
8:         for $t \in T$ do
9:             for $p \in \overrightarrow{P}$ do             ▷ Add rules according to rule template $w_{t1}$
10:                $rule :=$ HasType(S, t) ⇜ RDFSubjPred(S, p)
11:                $R \leftarrow R \cup rule$            ▷ Add rule to the set $R$
12:            end for
13:            for $p \in \overleftarrow{P}$ do             ▷ Add rules according to rule template $w_{t2}$
14:                $rule :=$ HasType(S, t) ⇜ RDFPredObj(p, S)
15:                $R \leftarrow R \cup rule$            ▷ Add rule to the set $R$
16:            end for
17:        end for
18:    end for
19:    return $R$                                  ▷ Returns the set of bootstrap rules
20: end function
```

4.3 Experimental Evaluation

This section presents our experimental studies to evaluate the effectiveness of the PSL approach to assimilating different types of evidence, as described above, for inferring a tabular structure from LD search results by instantiating the adopted meta-model. The goal of these experimental studies is to measure the quality of the integration results obtained by different PSL programs, as additional types of

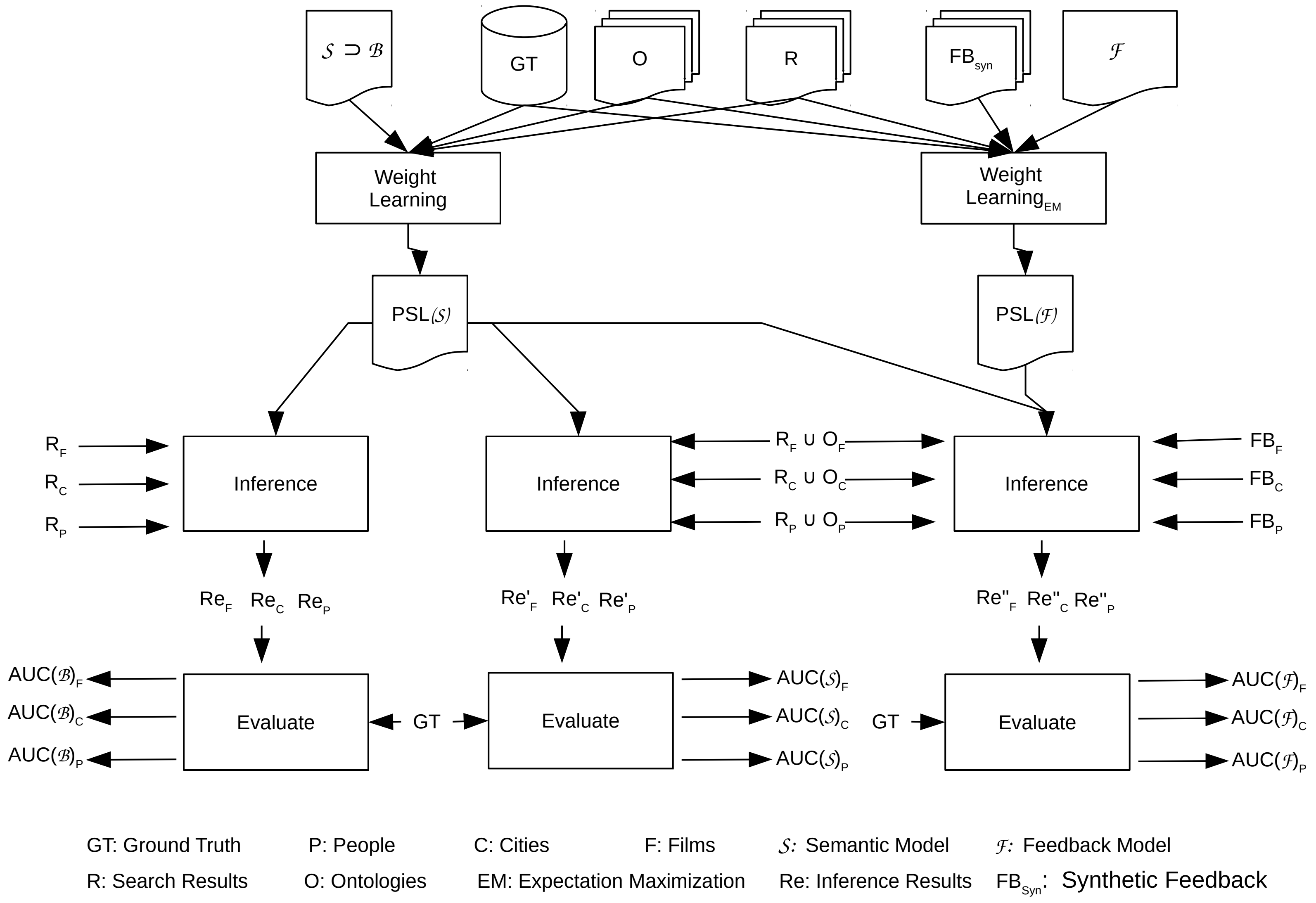


Figure 4.13: A flow digram of the experimental procedure

evidence are incrementally taken into account. Figure 4.13 summarizes the overall experimental process used in this evaluation.

4.3.1 Weight Learning for PSL Program Generation

We generated PSL programs $PSL(\mathbb{S})$ and $PSL(\mathbb{F})$ by learning weights for the rules in $\mathbb{S}$ and $\mathbb{F}$, respectively. As mentioned earlier, the semantic model subsumes the baseline model, i.e., $\mathbb{B} \subset \mathbb{S}$, and $\mathbb{F}$ is created by adding feedback rules to $\mathbb{S}$. Recall from Chapter 3 that our dataset consists of RDF resources collected by conducting a number of searches using Sindice [Oren et al., 2008] and Falcons [Cheng and Qu, 2009] LD search engines. For each search engine, the top 20 results were collected for each search term. The main reason for limiting ourselves to the top 20 is because of a limitation in the underlying SRL approaches used. As reported by Khosravi and Bina [2010], one of the main limitations of SRL approaches is the computational complexity of learning and inference. This is because the size of the ground Markov network is proportional to number of rules and predicates used in the model. This limits the capability of the proposed model to handle increasing amounts of evidence. To counter this, we limit our investigation on the top 20 results for each search. Note that all the experiments are executed on the following machine: Linux Ubuntu 14.04.2 LTS (64 bit), Intel Core i5-3470 3.20 GHz, 8 GB RAM.

We extended the dataset described in Section 3.3.2 by adding relevant ontologies. The vocabularies used were: for the `Films` domain, the DBpedia[4] and Movie[5] ontologies; for the `Cities` domain, the GeoFeatures[6] and GeoNames[7] ontologies; for the `People` domain, the FOAF and SWRC[8] ontologies. We chose these ontologies for the task in hand because they provide reasonable ontological coverage for the dataset used in the experiments. Ontological coverage is a metric that is often used

[4]We used a subset that is relevant to the films domain
[5]movieontology.org
[6]www.mindswap.org/2003/owl/geo/
[7]www.geonames.org/
[8]ontoware.org/swrc

when ascertaining the quality of an ontology for a particular task [Brewster et al., 2004]. Given an annotated database (e.g., RDF data), the coverage measures the extent to which the database uses concepts and properties from a given ontology. Here, concept coverage is the fraction of RDF classes occurring in the dataset that are defined by the given ontology. Precisely, concept coverage is defined by

$$ConceptCoverage(\mathcal{O}, \mathcal{D}) = \frac{\sum\limits_{oc \in C_{\mathcal{O}}} \sum\limits_{dc \in C_{\mathcal{D}}} I(oc, dc)}{|C_{\mathcal{D}}|} \tag{4.10}$$

where $\mathcal{O}$ is an ontology, $\mathcal{D}$ is a dataset, $C_{\mathcal{O}}$ is the set of RDF classes defined by $\mathcal{O}$, $C_{\mathcal{D}}$ is the set of RDF classes used in $\mathcal{D}$, and $I(oc, dc)$ is function that returns 1 if the local names of *oc* and *dc* are equivalent and 0 otherwise. Likewise, property coverage is defined by

$$PropertyCoverage(\mathcal{O}, \mathcal{D}) = \frac{\sum\limits_{op \in P_{\mathcal{O}}} \sum\limits_{dp \in P_{\mathcal{D}}} I(op, dp)}{|P_{\mathcal{D}}|} \tag{4.11}$$

Table 4.3 provides key statistics as well as coverage metrics of the vocabularies used in our experiments. In addition to concept and property coverage metrics, Table 4.3 shows counts of the following axioms. Note in each case we only consider the axioms where either property or a class label is an exact match to a property or a class in the collected search results.

- Property Domain Axioms (i.e., `p rdfs:domain c`).
- Concept Equality Axioms (i.e., `c1 owl:equivalentClass c2`).
- Property Equality Axioms (i.e., `p1 owl:equivalentProperty p2`).
- Disjoint Type Axioms (i.e., `c1 owl:disjointWith c2` and `c1 owl:complementOf c2`).
- Disjoint Property Axioms (i.e., `p1 owl:propertyDisjointWith p2`).

Domain	**Ontology**	**Concept Coverage**	**Property Coverage**	**Property Domain Axioms**	**Concept Eq. Axioms**	**Property Eq. Axioms**	**Disjoint Type Axioms**	**Disjoint Property Axioms**
Films	DBpedia	12%	30%	25	9	4	2	0
	Movies	11%	0%	0	0	0	0	0
People	FOAF	22%	16%	10	4	1	4	0
	SWRC	24%	16%	1	0	0	0	0
Cities	GeoNames	2%	1%	9	1	0	0	0
	GeoFeatures	11%	2%	2	0	0	0	0

Table 4.3: Concept and property coverage and key statistics of ontologies used in our experiment

Note that most of these counts, except for property domain and concept equality, are 0. This is consistent with our previous observations based on the results of the survey presented in Section 4.2.3. The above vocabulary features are seldom used, hence the observed counts.

Given the sample data from search results and vocabularies, we used PSL to learn the weights for the rules in the semantic model and yield the $PSL(\mathbb{S})$ program. We learned the weights discriminatively using maximum pseudo-likelihood. To reduce overfitting, we used 5-fold cross validation and we averaged the weights of each rule.

To learn the weights of the feedback model $\mathbb{F}$ and yield the PSL program $PSL(\mathbb{F})$, we proceeded as follows. As often done in evaluating dataspace approaches (e.g., [Belhajjame et al., 2013; Kot and Koch, 2009]), and in the absence of crowdsourced sample feedback instances, we simulated feedback acquisition to give a rise to a synthetic sample. The synthesis procedure we used is shown in Algorithm, 4.3. It can be described, intuitively, as follows. We take as input a random sample of inferences returned by $PSL(\mathbb{S})$ and fix a number of hypothetical users providing feedback (100, in this experiment). For each worker, we randomly generate 50 feedback instances, where a feedback instance is a true/false annotation on the inference returned for a query predicate. In our experiments we assume that feedback is reliable. However, introducing a per-user degree of unreliability only requires a simple change.

Algorithm 4.3 Feedback synthesis procedure

```
$rl \leftarrow$ reliability level a user
$u \leftarrow$ number of users
$c \leftarrow$ feedback instances per user
$Feedback \leftarrow \emptyset$
$CR \leftarrow$ candidate results returned by$PSL(\mathbb{S})$ inference
while size($Feedback$) $\leq u \times c$ do
    $t \leftarrow$ Uniformly assign a feedback target

    ▷ Choose a feedback candidate for target t from CR at random
    $fbi \leftarrow$ Choose($CR, t$)

    ▷ With probablity determined by 1-rl make feedback instance
      inconsistent with the ground truth
    $fbi \leftarrow$ MakeInconsistent($1 - rl, fbi$)

    $Feedback \leftarrow Feedback \cup fbi$
end while
```

4.3.2 Experiment 1: Inference with Syntactic and Semantic Evidence

Goal and Method

The goal of Experiment 1 is to measure the quality of $PSL(\mathbb{B})$, where only syntactic evidence is assimilated, and then measure the quality of $PSL(\mathbb{S})$, where semantic evidence is also assimilated, thereby allowing us to measure the impact of using ontologies on the quality. As in Chapter 3, we measure the quality of the outcome using the area under the precision-recall curve (AUC) for each query predicate in our PSL model. Recall that the precision/recall curve is computed by varying the probability threshold above which a query atom is predicted to be true. This means that the measurement does not depend on setting any threshold.

We first performed PSL inference on $PSL(\mathbb{B})$ on the search results, for each of the three domains in turn. We denote the measured quality as $AUC(\mathbb{B})$ with some abuse of notation. We then added semantic evidence extracted from the relevant vocabularies to the search results and performed inference on $PSL(\mathbb{S})$. We denote

Query Predicate	$AUC(\mathbb{B})$	$AUC(\mathbb{S})$	Δ	$AUC(\mathbb{S} \cup \mathbb{F})$	Δ'
`Entity`	.726	.736	.010	.827	.091
`EntityType`	.749	.812	.063	.954	.142
`HasProperty`	.512	.578	.057	.614	.036
`HasType`	.554	.604	.050	.709	.105
`Property`	.774	.774	.000	.779	.005
`SimEntity`	.083	.108	.025	.322	.214
`SimEntityType`	.320	.351	.030	.517	.166
`SimProperty`	.159	.184	.025	.621	.437

(a) Films

Query Predicate	$AUC(\mathbb{B})$	$AUC(\mathbb{S})$	Δ	$AUC(\mathbb{S} \cup \mathbb{F})$	Δ'
`Entity`	.645	.751	.105	.789	.039
`EntityType`	.828	.844	.017	.844	.000
`HasProperty`	.457	.511	.054	.559	.048
`HasType`	.557	.731	.174	.732	.000
`Property`	.683	.683	.000	.685	.001
`SimEntity`	.086	.583	.497	.612	.030
`SimEntityType`	.869	.891	.022	.891	.000
`SimProperty`	.257	.230	-.027	.537	.307

(b) People

Query Predicate	$AUC(\mathbb{B})$	$AUC(\mathbb{S})$	Δ	$AUC(\mathbb{S} \cup \mathbb{F})$	Δ'
`Entity`	.525	.651	.126	.882	.231
`EntityType`	.776	.793	.017	.793	.000
`HasProperty`	.470	.485	.015	.541	.056
`HasType`	.631	.668	.037	.820	.152
`Property`	.774	.780	.006	.794	.014
`SimEntity`	.082	.344	.261	.411	.068
`SimEntityType`	.648	.662	.014	.662	.000
`SimProperty`	.484	.386	-.098	.687	.302

(c) Cities

Table 4.4: AUC results for our PSL models with datasets in the test collection

the measured quality by $AUC(\mathbb{S})$. We then calculated the quality impact of using semantic evidence as $\Delta = AUC(\mathbb{S}) - AUC(\mathbb{B})$. Columns 2, 3 and 4 in each subtable in Table 4.4 list all the measurements obtained in Experiment 1 for the corresponding domain.

Results & Discussion

As measured in terms of the AUC, across the domains, the quality of $PSL(\mathbb{B})$ is good on average (around 0.65) if we discount the similarity relationships (except for `SimEntityType`) which are inherently dependent on semantic evidence. For example, we observed an average AUC of 0.7 on `Entity` and `EntityType` on the films and people domains. The results of the baseline model can be attributed to the amount of relevant results provided by the underlying search engines. However, these results are not without variations, in terms of the AUC, across the domains. One of the reasons of such variations can be attributed to the ambiguity of the search results. For example, the $AUC(\mathbb{B})$ of `Entity` in the cities domain is not as good as the corresponding results in the films and people domains because of the ambiguities in the individuals described in the results of the cities searches. Recall from Section 3.3.2, search of the cities domain, i.e., using the terms *Manchester* and *Berlin*, returns descriptions of (among others) artists, albums, and songs. The effect of ambiguity can be also observed in the result of `EntityType`. The $AUC(\mathbb{B})$ of `EntityType` in films domain is less than of that people domain. This is because the results in films contain more irrelevant types (e.g., as *Organization*, *City*, and *RailWayLine*) when compared, for example, with the results in the people domain.

For similarity predicates, we observed low $AUC(\mathbb{B})$ for `SimEntity`. This is because inference of `SimEntity` is dependent, given the structure of our rule-base, on inference of `Entity`. The lack of context information lead to low probability scores of instances of `Entity` in the baseline. For example, the inferred probability of `Entity(dbr:The_Godfather)` is 0.43 and `Entity(lmdb:/film/4338)` is 0.18 which leads to a probability score close to 0 for `SimEntity(dbr:The_Godfather,lmdb:/film/4338)`. This is observed for most instances of `SimEntity`. Thus, the value of AUC for `SimEntity` query predicate is low.

Assimilating ontological evidence indeed leads to improvement in the AUC, particularly w.r.t. similarity relationships, with a knock-on positive effect on the quality of the tabular representation. Thus, the corresponding average $AUC(\mathbb{S})$ increases to close to 0.7. The degree of improvement varies across domains depending on the coverage of the ontologies used. For example, the $\mathbb{B}$-probability of `HasProperty(dbo:Person, dbo:spouse)` is 0.04, whereas its $\mathbb{S}$-probability is much higher, at 0.80, as the DBpedia ontology explicitly defines this relationship. In this case the inferred probability due to the additional ontological evidence is less than 1 because our semantic rule base is uncertain, i.e., weighted. The additional evidence about the domain restriction relation between `dbo:spouse` and `dbo:Person` triggers rules `R32-R35`. The weighted distances to satisfaction of these rules is aggregated with the weighted distance to satisfaction of rule`R12` which results in a probability score of 0.80. In other cases, explicit assertion of type disjointness has a significant effect too. Thus, the $\mathbb{B}$-probability of `SimEntity(dbr:Casablanca, dbr:Casablanca_(film))` is 0.55, whereas its $\mathbb{S}$-probability is much lower, at 0.01, as a consequence of constraint rules C41 and C42, because `dbo:Work` and `dbo:wgs84_pos:SpatialThing` are disjoint in the DBpedia ontology. In the case of `Entity`, the assertion of a type by an ontology acts as a reliable anchor for individuals returned in the search results. For example, the $\mathbb{B}$-probability of `Entity(lmdb:film/43338)` is 0.22, whereas its $\mathbb{S}$-probability is higher, at 0.38, because its type, `lmdb:film`, matches the type `dbo:Film`, in the MO. As hinted above, improvements in the inference of metatypes (e.g., `Entity`) has a knock-on effect on the corresponding set-similarity relationship (`SimEntity` in this example). In the case of `SimEntity`, the improvement is more significant in the `People` and `Cities` domain than for `Films` domain because of inherent type ambiguity. For example, searching with `"Casablanca"` returns films, organizations, and a city. Type ambiguity is perhaps best solved with user feedback, as the next experiment shows.

4.3.3 Experiment 2: Inference Using Feedback

Goal and Method

The goal of Experiment 2 is to measure the quality of $PSL(\mathbb{S} \cup \mathbb{F})$, where feedback evidence is also assimilated, thereby allowing us to measure the impact of using feedback on the quality. As humans are highly unlikely to provide feedback on all candidate instantiations of the meta-model, we simulated the feedback evidence as being provided for the top 5% of the inference results produced by semantic model $PSL(\mathbb{S}$ for each feedback target. This assumes a strategy in which feedback is targeted at removing false positives that occur in the top 5% likely candidates. We denote the measured quality by $AUC(\mathbb{S} \cup \mathbb{F})$. We then calculate the quality impact of using semantic evidence as $\Delta' = AUC(\mathbb{S} \cup \mathbb{F}) - AUC(\mathbb{S})$. Columns 5 and 6 in each subtable in Table 4.4 list all the measurements obtained in Experiment 2 for the corresponding domain.

Results & Discussion

As measured in terms of the AUC, across the domains, the quality of $PSL(\mathbb{S} \cup \mathbb{F})$ is quite good on average (around 0.7) even if we include the similarity relationships. In other words, user feedback seems to address some of the ambiguity issues that caused the quality of $PSL(\mathbb{S})$ to be lower for similarity relationships. Thus, although the impact of feedback is not uniformly high, it seems complementary and corrective, i.e., it improves the most where the most improvement is needed, viz., the similarity relationships, where we can observe AUC improvements that can reach 80%, 130%, up to almost 240%. This, the highest improvement, was observed for `SimProperty` in the `Films` domain. The reason for this is that $PSL(\mathbb{S})$ produces many false positives for `SimProperty`. One possible reason is that property names (e.g., `name`) are often reused without qualification for very different concepts. Combining syntactic and ontological evidence seems insufficient in that case.

4.3.4 Experiment 3: Using Domain Specific Rules

Goal and Method

The goal of Experiment 3 is to build a model that is capable of identifying fine-grained entity types by learning weights that establish which property assertions provide best evidence for type assertions. Our approach is predicated on using instances from an existing knowledge base to bootstrap a PSL model that associates RDF properties to the RDF types used in the description of these instances. We use the PSL weight learning processes to learn the degree to which a certain property is indicative of a given type. This, in turn, becomes a supervised classification approach where *features* are local names of RDF properties used to describe the input instances, and the output is the likely entity type, given those features.

We evaluated our approach using 1,000 instances from the `Films` domain sampled from the DBpedia dataset. We used a 5-fold cross validation process for training and testing the models used in this experiment. We randomly sampled an almost equal number of instances that belong to the entity types `Actor`, `Director`, `Producer`, `Composer`, `Editor`, `Writer`, `Film` and `Work`. Given that each instance is often annotated with more than one entity type, the total number of entity types in the resulting dataset is 97. We then used this dataset to bootstrap a rule-base using the templates described in Section 4.2.4 and Algorithm 4.2. The bootstrapping phase generated $35,819$ rules which is almost equivalent to the product of the number distinct types (i.e., 97) and number of distinct properties (i.e., 1068) divided by 3. Note that most of properties appear with only a third of the types in the dataset.

In this experiment, the large margin estimation (LME) learning method that is provided by the PSL implementation was chosen. As mentioned previously, LME is a PSL weight learning method which drops the probabilistic interpretation of a PSL program and focuses the learning task at finding rule weights which provide accurate predication [Bach et al., 2015]. Models which are learned using the LME method can produce more accurate predications when compared to other learning approach such as the maximum pseudo-likelihood estimation (MPLE) in some structured predication problem such as collective classification and collaborative filtering [Bach et al., 2013b].

After running the weight learning stage on each fold, the weight for some rules was small. In the inference process, we eliminated rules with weight less than or

```
/* outgoing properties */
1.16: HasType(S, "Editor")  ⇠ RDFSubjPred(S, "surname")
1.16: HasType(S, "Editor")  ⇠ RDFSubjPred(S, "givenName")
0.81: HasType(S, "Editor")  ⇠ RDFSubjPred(S, "award")
0.60: HasType(S, "Editor")  ⇠ RDFSubjPred(S, "child")
0.58: HasType(S, "Editor")  ⇠ RDFSubjPred(S, "awards")

/* incoming properties */
4.06: HasType(S, "Editor")  ⇠ RDFPredObj("editing", S)
1.26: HasType(S, "Editor")  ⇠ RDFPredObj("award", S)
0.52: HasType(S, "Editor")  ⇠ RDFPredObj("editor", S)
0.26: HasType(S, "Editor")  ⇠ RDFPredObj("producer", S)
0.18: HasType(S, "Editor")  ⇠ RDFPredObj("spouse", S)
```

Listing 4.17: A sample of learned weights for rules in our domain specific model

equal to 0.01. The total number of rules which received a weight higher than 0.01 was $5,517$ (15.4%) out of the original $35,819$ (100%) rules. Listing 4.17 shows a sample of learned weights for some of the rules for `Editor`.

To establish a baseline, we compared the results of our approach to the results of two well-known *ensemble learning* approaches: *Random Forest* [Geurts et al., 2006] and *AdaBoost* [Freund and Schapire, 1997]. These approaches are capable of handling multi-class classification problems as opposed to binary ones only. *AdaBoost* is a technique where a group of classifiers is trained in an iterative process. In each iteration, a basic learning model is employed to train a weak classifier. At the end of each round, mis-classified examples are identified and emphasis is given to them in a new training set that is then fed back for training a new model. The idea is that the new model should be able to correct the mis-classifications of the earlier model. Random forest is a learning method that trains several decision trees. Each tree is trained on new training set that is chosen at random from training data. The classification results of the random forest are the result of voting from each of these trees. In this experiment, we used the Python Scikit [9] API implementation of AdaBoost and random forest classifiers. The number of underlying decision trees classifiers that were used for each of the two learning methods was 50.

[9]http://scikit-learn.org

Classifier	AdaBoost	RandomForest	PSL (LME)
Accuracy	0.33 ± .05	0.49 ± .05	0.88 ± .03

Table 4.5: Cross-validated accuracy results for the classifiers used in Experiment 3 ± the standard deviation across different folds

We used standard machine learning metrics to evaluate our approach. To compare the performance of the different classifiers we used the classification accuracy defined as follows:

$$accuracy = \frac{\text{Number of correct predictions}}{\text{Number of all predictions}} \tag{4.12}$$

To evaluate the classification of individual types, we used the precision, recall and F-measure metrics. We took the *true positives* to be the count of correctly classified instances, the *false positives* to be the a count of incorrectly classified instances, and the *false negatives* to be a count of correct predictions that were missed by the classifier.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{4.13}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{4.14}$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4.15}$$

In the case of our PSL model, we rounded the output probability for each query proposition to the nearest integer (i.e., either 0 or 1).

Results & Discussion

Table 4.5 shows the accuracy for the classifiers averaged over the different folds in our experiment. The predictions of the PSL approach are at least 30% more accurate than the predictions of the other classification approaches (i.e., random forest and AdaBoost). Figure 4.15 shows the confusion matrices for a number of selected entity types for each of the classifiers used in the experiment. A confusion matrix is a two-dimensional matrix indexed by one dimension by the true type of the individual and by the type that the classifier assigns in the other. It is used to visualize the performance of a classifier with respect to some testing dataset. Elements in the diagonal represent number of instances for which the predicted entity type is equal to the true entity type. The propositional classifiers tend to incorrectly predict (as shown in Figure 4.15) the types of instances that belong to the same subsumption subtree (e.g., `Director`, `Editor`). Although the propositional classifiers achieve at least 50% precision on most of the target types (see Figure 4.14), their recall tends to be lower than the recall of the PSL approach. For example, for the `Film` entity type, while the AdaBoost classifier achieves a precision score of 0.98, the recall, at 0.56, is lower than that of the PSL approach. The reason for this is that the AdaBoost classifier often assigns instances of `Film` to the entity type `Work` only. Furthermore, the predictions made by the propositional classifiers (in this case, AdaBoost) for the entity types `Editor` and `Composer` tend to be imprecise because there are very few properties which can signal such entity types (`editing` for `Editor`, and `music` for `Composer`). For example, the property that distinguished between entity type `Editor`, on the one hand, and entity types `Actor` and `Director` on the other, is the incoming property `editing`. Given that the entity types `Actor` and `Director` share many properties with `Editor` (they are all subclasses of `Person`) instances of `Actor` and `Director` are also classified as `Editor`. The advantage of using an SRL approach such as PSL in this case is that the properties which are likely to distinguish certain entity types are learned with higher weights (see Listing 4.17 for an example), which leads to better predictions.

Finally, we used the domain-specific model for the `Films` domain in the inference of the meta-model constructs from LD searches. We added these rules to the baseline

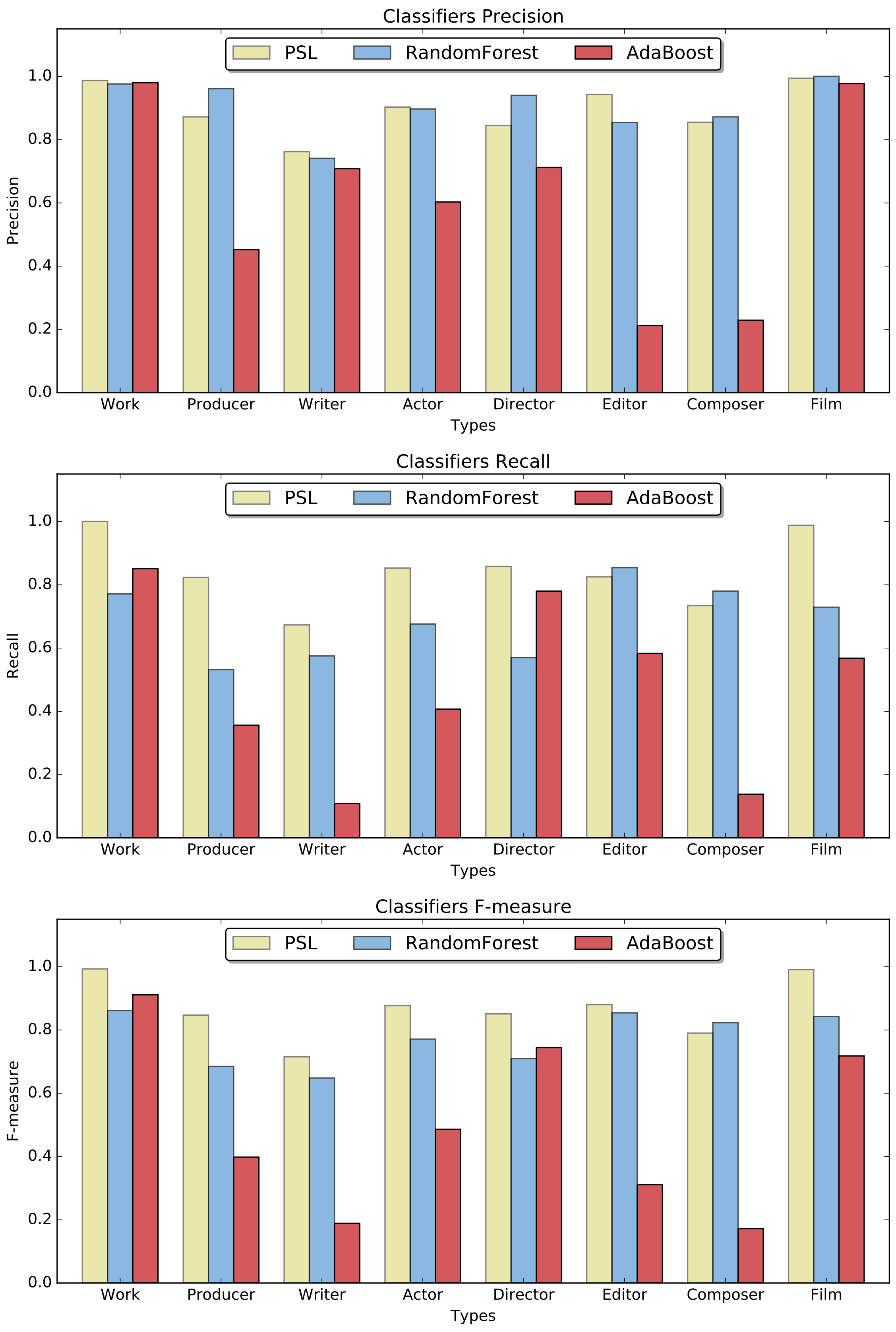
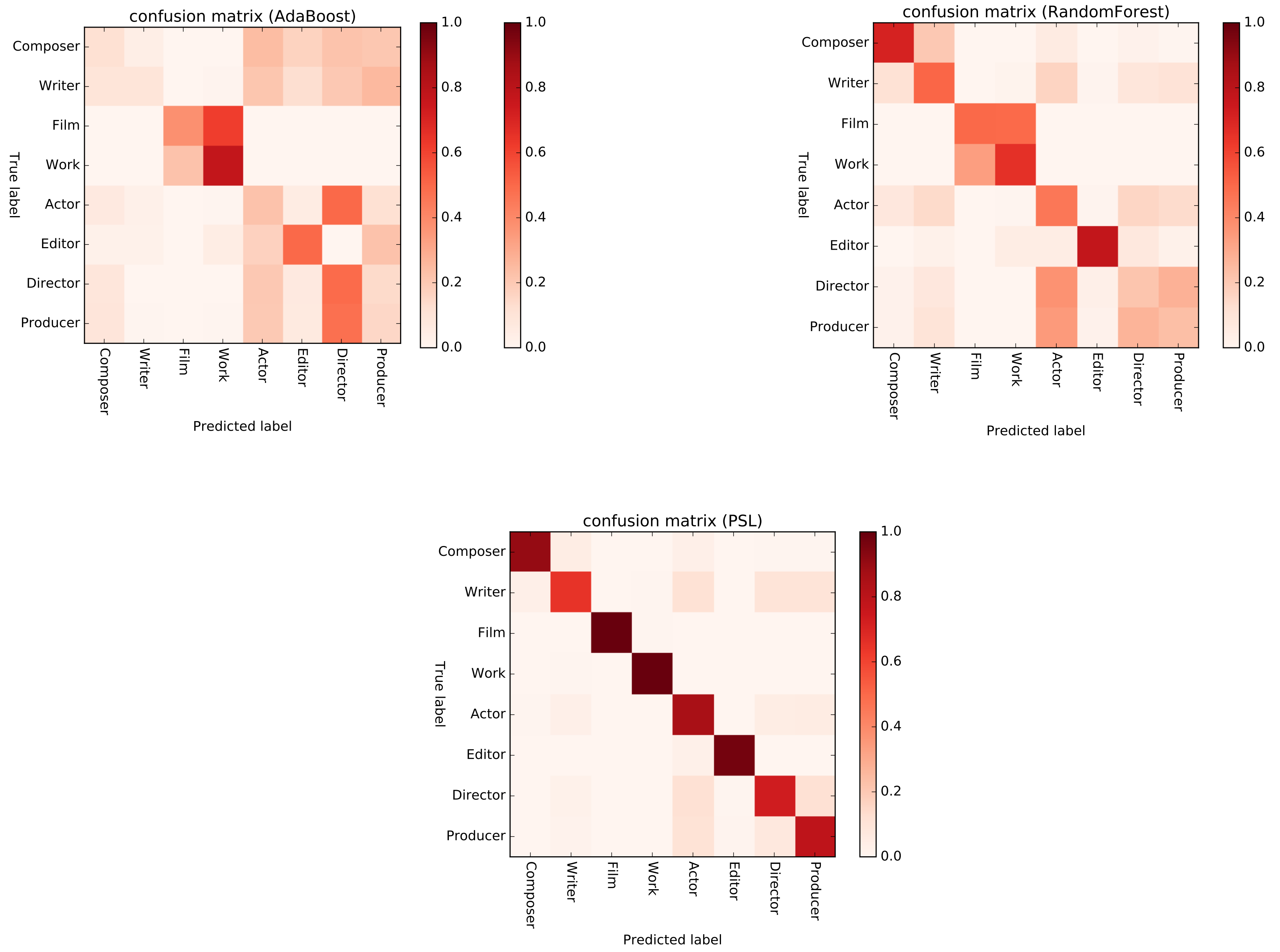


Figure 4.14: Precision, Recall, and F-measure score for a number of entity types

Figure 4.15: Confusion matrices for the classifiers used in exp. 3



PSL program $PSL(\mathbb{B})$ to give a rise to the new PSL program $PSL(\mathbb{B} \cup \mathbb{DS})$ and ran the inference procedure on the search results from the `Films` domain. We then measured the quality of the inferred results, which we denote by $AUC(\mathbb{B} \cup \mathbb{DS})$. We then calculated the impact of using these rules as $\Delta'' = AUC(\mathbb{B}) - AUC(\mathbb{B} \cup \mathbb{DS})$.

Table 4.6 shows the impact of using domain-specific rules in the quality of inference results. As measured in terms of AUC, using such rules improves the inference of instances of `HasType`. The additional rules improve in the inferred probability for individuals originating from the DBpedia dataset, which is to be expected, since the domain-specific model was trained on examples from the DBpedia dataset. For example, the $\mathbb{B}$-probability for `HasType(dbr:Diane_Keaton, Actor)` is 0.46, whereas its $\mathbb{B} \cup \mathbb{DS}$-probability is at 0.95.

There is the potential for such improvements to be obtained for individuals found in other datasets. However, this requires mapping RDF types and properties found in other datasets to the RDF types and properties in the bootstrapped model. While PSL facilitates such mappings, investigating the impact on quality of their use is left for future work

Query Predicate	$AUC(\mathbb{B})$	$AUC(\mathbb{B} \cup \mathbb{DS})$	Δ''
`Entity`	.726	.726	.000
`EntityType`	.749	.749	.000
`HasProperty`	.512	.513	.001
`HasType`	.554	.732	.178
`Property`	.774	.774	.000
`SimEntity`	.083	.083	.000
`SimEntityType`	.320	.319	-.001
`SimProperty`	.159	.159	.000

Table 4.6: AUC results showing the impact of using domain specific rules in the `Films` domain

4.4 Related Work

This chapter took the view that the integration of LD search results is better addressed by seeking to combine multiple sources of evidence. Proposals for inferring semantic correspondences in LD integration tasks often derive their results by combining together different pieces of evidence. One source of evidence that is often utilized by ontology matching approaches is existing background knowledge in the form of LOD vocabularies. Approaches such as SMatch [Giunchiglia et al., 2005] and Carnot [Huhns et al., 1993] use generic upper vocabularies to match terms that refer to concepts. The authors of SMatch use WordNet to match concept labels from two XML schemas, whereas the Carnot system utilizes the Cyc knowledge base for inferring semantic correspondences between relational database schemas. Although such sources of evidence are useful for generic concept matching tasks, they may not provide expressive background knowledge for inferring semantic correspondences between properties, or between concepts and properties as required in our settings.

Such knowledge is found in domain vocabularies, which describe concepts and properties which pertain to specific domains. For example, the DICE ontology was used by Aleksovski et al. [2006] to match two semi-structured medical vocabularies. This approach used the DICE ontology as an integration schema by aligning the matched vocabularies to the DICE ontology. Then the ontological axioms in the DICE ontology were used to find semantic relations between the concepts in the input vocabularies. In contrast, our approach can use multiple vocabularies to provide evidence for our target structure. Furthermore, given that an evidence vocabulary might express relationships that are irrelevant to the searches, we treat ontological evidence as uncertain, and we use PSL's supervised weight learning processes to capture the uncertainty of the rules which utilize such evidence.

The idea of using multiple vocabularies as evidence in a data integration task was explored in Scarlet [Sabou et al., 2008]. Scarlet works by relating ontologies to be matched with ontologies found in the WoD. The basic idea in Scarlet is that the use of more than one ontology improves the matching results. However, Scarlet is limited to finding similarity relations between concepts in the matched ontologies.

In contrast, our approach seeks to infer a richer set of semantic relations between concepts, properties and individuals.

Combining multiple sources of evidence in data integration tasks requires handling the inherent uncertainty stemming from using different sources of evidence. Probabilistic approaches are often used to model uncertainty in such tasks. As discussed in Chapter 3, our use of a SRL approach to combine different sources of evidence, as also presented in this chapter, is motivated by previous applications of SRL approaches to data integration tasks (e.g., [Niu et al., 2012] and [Niepert et al., 2011]). The use of PSL in this chapter was motivated by its support for reasoning with similarities and with sets, which is important when dealing with heterogeneous data. Other probabilistic methods that are used to assimilate different pieces of evidence include Bayesian updating (e.g., [Christodoulou et al., 2015a]). The idea of Bayesian updating [Spragins, 1965] is based on the iterative use of Bayes' rule to revise the probabilities in the light of new evidence. In matching tasks Bayes' rules is used to compute the posterior probability of a match giving different forms of evidence (e.g., semantic and syntactic evidences). To enable the iterative application of Bayes' rule, this approach requires the derivation of a likelihood function for each type of evidence involved. The limitation of this approach is that it is unclear how adaptable is the process of training probabilistic distribution functions for when there are complex dependencies to be modelled between the evidences and queries. Our approach avoids this by using PSL's declarative approach to introduce new forms of evidence, which allows us to propagate new evidence in the induced ground network.

Our use of feedback as a source of evidence is motivated by a growing body of literature where feedback is used for solving data integration problems (e.g., [Belhajjame et al., 2011; Sarasua et al., 2012; Isele and Bizer, 2013]). A number of ontology matching systems, e.g. [Shi et al., 2009; Cruz et al., 2012; Duan et al., 2010], also adopt a strategy where feedback is obtained from a single user. For example, Shi et al. [2009] implemented a matching system that interactively seeks user feedback to: correct the matching mistakes and propagate the feedback information to update the threshold used for selecting matched candidates. Our approach is different because our feedback model can be trained on feedback obtained from multiple users. In adapting a multi-user feedback strategy, our approach is similar to CrowdMap [Sarasua et al., 2012], where crowdsourcing is used to improve the precision of ontology alignment tasks, and ZenCrowd [Demartini et al., 2012], which combines human feedback with automatic integration for linking entity extractions from web pages to instances in the WoD. The limitation of our approach in comparison to these approach is that we do not account for per-user reliability in our feedback rules. Handling unreliable users in our approach requires the addition of rules that are corrected based on the reliability of users, or per-user sets of feedback rules and learning weights that capture the reliability of each individual user in a manner which is similar to the application of domain specific rules presented in Section 4.2.4. As such, in line with the goal pioneered by Sig.ma, our approach is more suitable for a setting in which the paramount need is for personalization of search results.

4.5 Summary and Conclusions

Integration of LD search results in the form of a tabular structure is challenging because it requires automating a number of complex subtasks, such as structure inference, and matching concepts and instances, each of which gives rise to uncertain outcomes. Such uncertainty cannot be avoided given the heterogeneous nature of the WoD. In this chapter, we provided experimentally-derived backing for the hypothesis that assimilating different sources of evidence is effective in inferring a good quality structure over LD search results. We have extended the approach described in

Chapter 3 by incorporating additional types of evidence. We have described how a PSL program has been constructed with which different sources of evidence can be assimilated in a principled and uniform way, where such sources are syntactic matching, domain ontologies, and user feedback. To do so, we have defined a meta-model that defines the structure to be inferred, and used PSL to express rules for populating this meta-model. The advantage of using PSL for such a task is that it allows us easily to encode the dependencies between different sources of evidence and the meta-model constructs, using a syntax that is based on first-order-logic. This allows us easily to add and remove dependencies that improve the quality of our model. We have demonstrated this approach through the use of PSL to build a rule base that enables the propagation of evidence even when it is scarce (e.g., in the case of ontological evidence). Another advantage of PSL is that it does not require specialized inference and learning algorithms other than those already assigned to PSL and available in its implementation.

Two main results have been obtained: SRL methods such as PSL can be used for the assimilation of different kinds of evidence in relation to the integration of LD search results, and when semantic and feedback evidences are present the quality of integrated results is improved, thereby backing our working hypothesis.

In Chapter 5, we show how the results of the PSL program presented in this chapter can be used to power a user interface by means of which a user can provide feedback that improves future quality, in a pay-as-you-go style.

Chapter 5

An Interface for Integrating LD Search Results

RDF data are becoming pervasive in the Web, as is evident in the growth of published LD sources. As the size of the WoD continues to grow, better tools are needed that more effectively support user access to this source of information.

As we saw in Chapter 2, RDF data in the Web can be characterized as data that tends to be large, highly heterogeneous, and not adhering to a specific schema. Effective user access to the WoD need to address the ensuing challenges. Many methods have been proposed for providing users access to RDF data in the Web. In general these methods can be classified into: (i) RDF querying (e.g., [Quilitz and Leser, 2008]), i.e. using SPARQL, (ii) keyword searches (e.g. Sindice [Oren et al., 2008]), (iii) graph visualization (e.g., [Frasincar et al., 2006]) and (iv) faceted navigation (e.g., [Oren et al., 2006]). Individually, none of these methods is sufficient to support all the information seeking tasks one can envisage as being appropriate in the context of the WoD.

RDF querying requires a user to have technical knowledge about the query language and to know the schemas, if any, in the underlying datasets. Studies conducted on relational databases suggest that formulating a query to an unknown dataset is a challenging task for a user [Jagadish et al., 2007; Nandi and Jagadish, 2011]. In the WoD, this problem is even more difficult as there is no distinction between instances and schemas and usually meta-data is missing. This also makes traditional non-expert query specification approaches (e.g., query-by-example) infeasible.

An alternative approach to RDF querying is keyword searching, which has proved to be an easy and intuitive way of interacting with the Web. As described in Chapter 2, there are two broad categories of keyword searching over RDF data. The first uses information retrieval techniques to index the RDF documents, thereby enabling the use of search terms as keywords to identify relevant documents. The second category matches search terms against elements of an RDF graph. In these approaches, only the sub-graphs containing the search term are returned, as a ranked list. In neither category is there any attempt to integrate the results into a structure that corresponds to the search terms provided.

RDF graph visualization techniques do not yet scale for large datasets [Frasincar et al., 2006]. Additionally, a graph may not be the best method for visualizing RDF data. Dense RDF graphs might be problematic regarding usability. As more data is depicted in the graph, the gist (i.e., a particular real-world entity) of what the graph is depicting often gets lost [Schraefel and Karger, 2006], which results in a failure to communicate the crucial information about the underlying entities.

Faceted interfaces offer yet another approach for exploring RDF datasets, especially when coupled with automatic means for generating *facets*. A *facet* is an user interface element that allows the user to quickly find information in an unknown dataset [Tunkelang, 2009]. With facets, results are partitioned into orthogonal categories, possibly organized into a hierarchy. Predominantly, facets are created manually [Croft et al., 2009]. The problem with manually-created facets is that they are static and domain-specific. Automatic approaches to creating facets are predicated on using features of the underlying resources, such as types and property names, to group similar resources together [Oren et al., 2006; Harth, 2010; Dachselt et al., 2008; Wang et al., 2009]. Given the heterogeneity of the WoD, it can be difficult to produce useful facets with such approaches [Erling and Mikhailov, 2009; Shangguan and McGuinness, 2010; Teevan et al., 2008].

In this chapter, we present an interface that is designed to enable non-expert end-users to retrieve integrated LD search results in the WoD. The interface builds upon RDF search while providing faceted navigation of the results to facilitate user exploration. It makes intensive use of the inference results obtained using the PSL program presented in Chapter 4. Starting with the RDF results returned by a search term, the PSL program populates a tabular structure that characterizes the result that we want to present to the end user. The inference decisions made by the PSL program are used to derive a user interface, thereby allowing the user to iteratively and interactively correct and refine the results.

Using this interface, we can populate tabular reports that describe the individuals that are returned by the given search term. The main reason for targeting tabular reports is that they provide a useful view for user, who are used to tables from spreadsheets, Web tables, etc. Such tabular structures can be easily imported into a relational database, which enables efficient querying and processing by downstream applications. For example, one could easily imagine a case where a table that was constructed for the results of the search terms `Godfather Actors` is then exported for use in an application that makes, for example, movie recommendations. In addition, a data table is a structure that can more effectively present complex data when compared to raw RDF triples. Such tables are likely to facilitate user navigation and interaction with the underlying data.

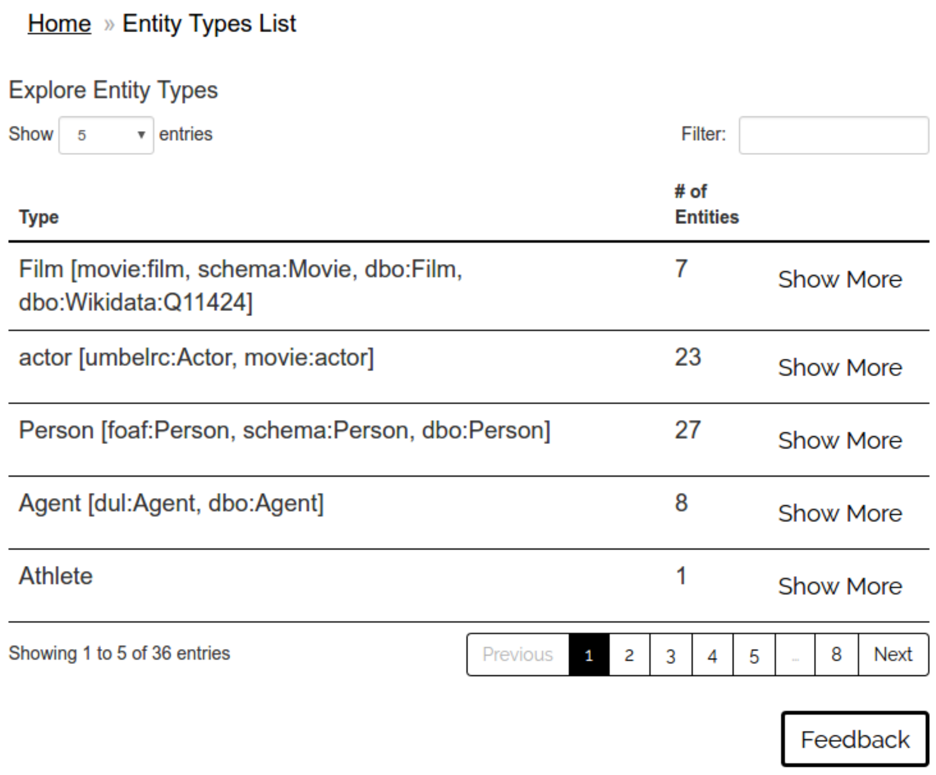
The main advantages of the proposed user interface, therefore, are (i) a simple way for finding relevant data from multiple sources, (ii) the integration of the retrieved results into tables that represent entity types that underpin the results and (iii) the possibility of correcting the integrated results and iterating the process.

In the following, we first describe a case study for the use of the proposed interface (Section 5.1). We then provide a general description of tool that provides the proposed interface (Section 5.2).

5.1 A Case Study

Assume that user gives `"Godfather actors"` as the search term. Relevant returned results come predominantly from the Linked Movie Database and DBpedia. A few, less relevant, results come from Linked WordNet, BookMashup, and MusicBrainz. The PSL program uses the results to make inferences as to how to instantiate the target metamodel in a way that integrates the returned results into a tabular report.

As depicted in Figure 5.1, our PSL-driven user interface then provides the user with a list of postulated entity types for the given search term. The PSL query predicate behind Figure 5.1 is `HasType`, with rows ordered by `EntityType` probability. The inference in Figures 5.1 and 5.2 are made based on evidences from the search results of search term `"Godfather actors"`, DBpedia and Movie ontologies. The types shown in Figure 5.1 are based on `rdf:type` assertions found in triples of the search results. However, given evidences in the search results and the input ontologies, the most likely entity types, as infered by the semantic model, are `Film`, `actor`, `Person`, `Agent`, and `Athlete`. The second column in Figure 5.1 is the count of instances for each candidate entity type in the results. This count is based on inferences of the `HasType` query predicate. At this point, the user can express an interest in one of the listed types by pressing on `Show More`, which would then list the inferred properties of the selected type, as shown in Figure 5.2 for the postulated entity type `Film`. The PSL query predicate behind Figure 5.2 is `HasProperty`, with rows ordered by `Property` probability. The properties shown in Figure 5.2 are based on RDF property assertions found in the search results. For example, `director` is


Home » Entity Types List

Explore Entity Types

Show 5 entries Filter:

Type	# of Entities	
Film [movie:film, schema:Movie, dbo:Film, dbo:Wikidata:Q11424]	7	Show More
actor [umbelrc:Actor, movie:actor]	23	Show More
Person [foaf:Person, schema:Person, dbo:Person]	27	Show More
Agent [dul:Agent, dbo:Agent]	8	Show More
Athlete	1	Show More

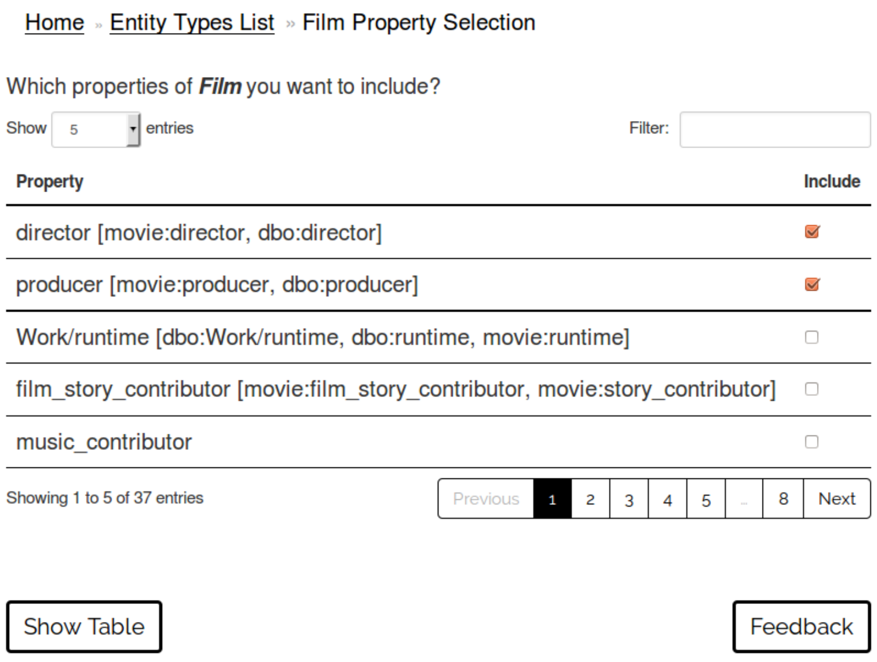
Showing 1 to 5 of 36 entries

Figure 5.1: Type selection

a property of the resource `lmdb:/film/43338`. However, in RDF there is no direct correspondence between type and property assertions, as publishers are not required to adhere to any specific schema. Our PSL model infers this relation, i.e., `HasProperty`, based on evidence using set-similarity functions as well as evidence based on domain restriction axioms defined by the input ontologies.

At this point, the user can tick which properties to include in the final tabular representation. Clicking on `Show Table` causes it to be shown as seen in Figure. 5.3 for the entity type `Film` after the properties `prequel`, `director` and `producer` have been selected. Some properties shown (e.g., `producer`) are candidates for fusion and some entities (e.g., `The Godfather`) are candidates for deduplication.

As the user interacts with the tool, click-throughs are considered as an implicit feedback on the inference results of the PSL model. For example, in the type selection screen, i.e., Figure 5.1, clicking on the `Show More` link corresponding to the second entry, i.e., `actor`, generates the feedback instance `EntityTypeFB(actor,yes)`


Home » Entity Types List » Film Property Selection

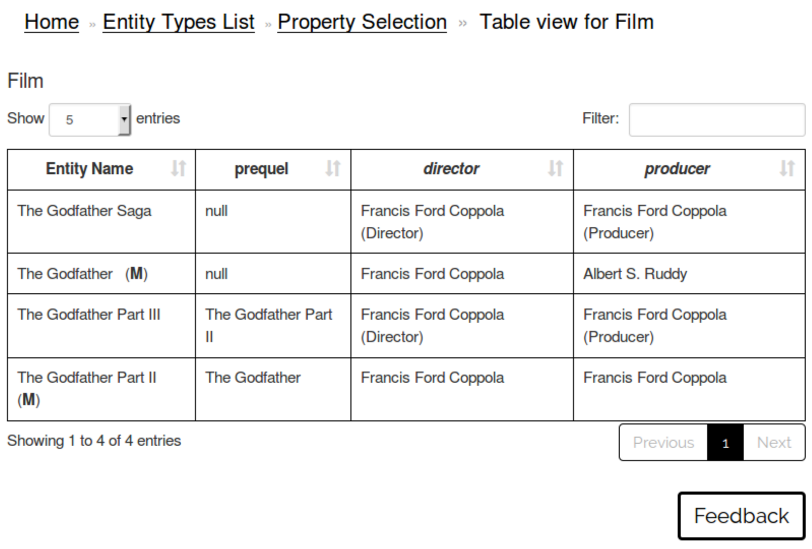
Which properties of ***Film*** you want to include?

Show 5 entries Filter:

Property	Include
director [movie:director, dbo:director]	✓
producer [movie:producer, dbo:producer]	✓
Work/runtime [dbo:Work/runtime, dbo:runtime, movie:runtime]	☐
film_story_contributor [movie:film_story_contributor, movie:story_contributor]	☐
music_contributor	☐

Showing 1 to 5 of 37 entries

Figure 5.2: Property selection


Home » Entity Types List » Property Selection » Table view for Film

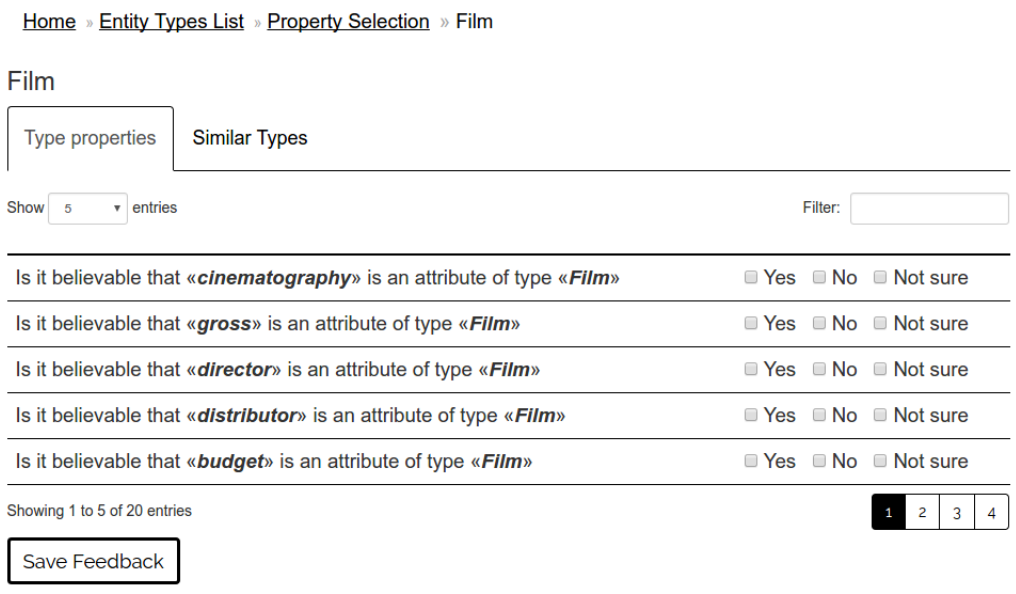
Film

Show 5 entries Filter:

Entity Name	prequel	*director*	*producer*
The Godfather Saga	null	Francis Ford Coppola (Director)	Francis Ford Coppola (Producer)
The Godfather (**M**)	null	Francis Ford Coppola	Albert S. Ruddy
The Godfather Part III	The Godfather Part II	Francis Ford Coppola (Director)	Francis Ford Coppola (Producer)
The Godfather Part II (**M**)	The Godfather	Francis Ford Coppola	Francis Ford Coppola

Showing 1 to 4 of 4 entries

Figure 5.3: Data table


Home » Entity Types List » Property Selection » Film

Film

Type properties | Similar Types

Show 5 entries Filter:

Is it believable that «*cinematography*» is an attribute of type «*Film*»	☐ Yes	☐ No	☐ Not sure
Is it believable that «*gross*» is an attribute of type «*Film*»	☐ Yes	☐ No	☐ Not sure
Is it believable that «*director*» is an attribute of type «*Film*»	☐ Yes	☐ No	☐ Not sure
Is it believable that «*distributor*» is an attribute of type «*Film*»	☐ Yes	☐ No	☐ Not sure
Is it believable that «*budget*» is an attribute of type «*Film*»	☐ Yes	☐ No	☐ Not sure

Showing 1 to 5 of 20 entries 1 2 3 4

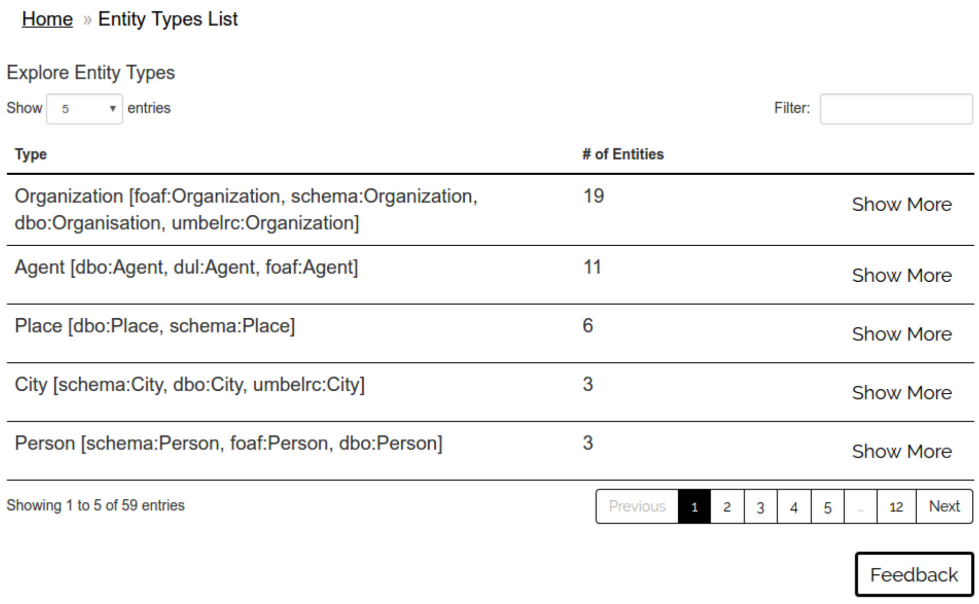
Save Feedback

Figure 5.4: Explicit feedback on `HasProperty` results

which can be then used as positive evidence that supports the proposition that `actor` is an entity type. Similarly, selecting properties to include in the tabular representation, as done in Figure 5.2, implicitly generates instances of `HasPropertyFB`, such as `HasPropertyFB(Film, director, yes)`.

At each stage in this process, the user can intervene by clicking on the `Feedback` button to contribute explicit feedback, which becomes evidence for use in future searches. For example clicking on the `Feedback` button on the property selection screen shown in Figure 5.2, causes the screen in Figure 5.4 which prompts the user to provide feedback. Explicit feedback from the user is a comment from a user on the correctness of the inferred query predicate. In this specific case it is a testimony on the correctness of the results of `HasProperty`. Explicit feedback is collected and used to any future search thus allowing the personalization of the search results based on the provided feedback.

To exemplify the effect of providing explicit feedback on the inference results, consider an example where a user provides the search term `Manchester`. The most likely entity type based on the inference results of the baseline model described in Chapter 4 is `Organization` (see Figure 5.5). This is because the results for the


Home » Entity Types List

Explore Entity Types

Show 5 entries Filter:

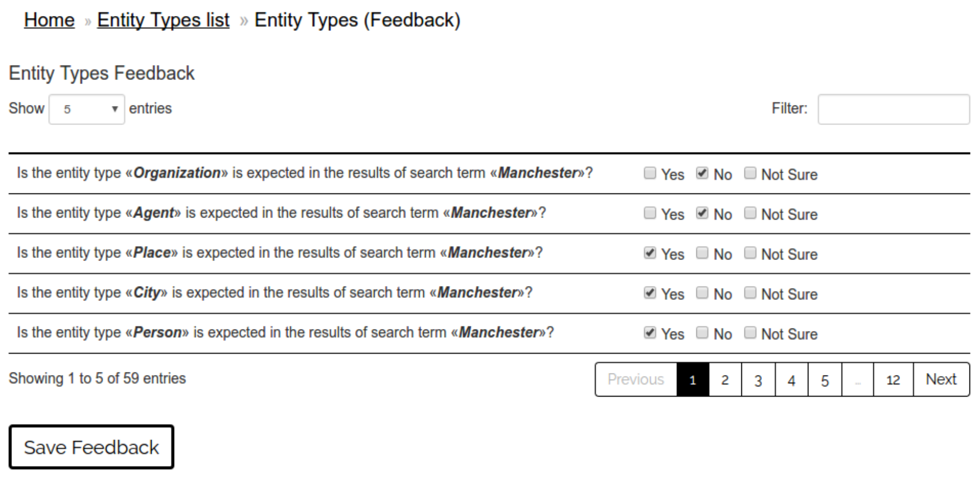
Type	# of Entities	
Organization [foaf:Organization, schema:Organization, dbo:Organisation, umbelrc:Organization]	19	Show More
Agent [dbo:Agent, dul:Agent, foaf:Agent]	11	Show More
Place [dbo:Place, schema:Place]	6	Show More
City [schema:City, dbo:City, umbelrc:City]	3	Show More
Person [schema:Person, foaf:Person, dbo:Person]	3	Show More

Showing 1 to 5 of 59 entries

Figure 5.5: Inferred instances of `EntityType` for the search term `Manchester`

search term `Manchester` contains results about record label companies that are annotated with entity type `Organization`. If the user provides feedback (by clicking on the `Feedback` button on the screen in Figure 5.5) that the expected type for this search term is `Place` and not `Organization` (as shown in Figure 5.6). The feedback instances that are generated based on the user action shown in Figure 5.6, among others, are `EntityType(Place, yes)` and `EntityType(Orgranizaton, no)`. If the user then proceeds with the search term `Casablanca` with the above feedback instances, the PSL program which assimilates feedback evidence produces the results shown in Figure 5.7, where `Place` is among the highest likely types for the search term `Casablanca`. Without the above feedback, `Place` does not rank highly among the likely type of the search term `Casablanca` (see Figure 5.8).

Note, therefore, that the use of a probabilistic framework allows us not only to structure and integrate the results but also to improve presentation (e.g., by ordering rows by likelihood) and to obtain targeted feedback. Note also that since the underlying PSL program models similarity relationships, the interface can make principled, uniform choices regarding deduplication (using `SimEntity` and `SimProperty`) and data fusion (using `SimProperty` and `SimPropertyValue`). In the example



Home » Entity Types list » Entity Types (Feedback)

Entity Types Feedback

Show 5 entries Filter:

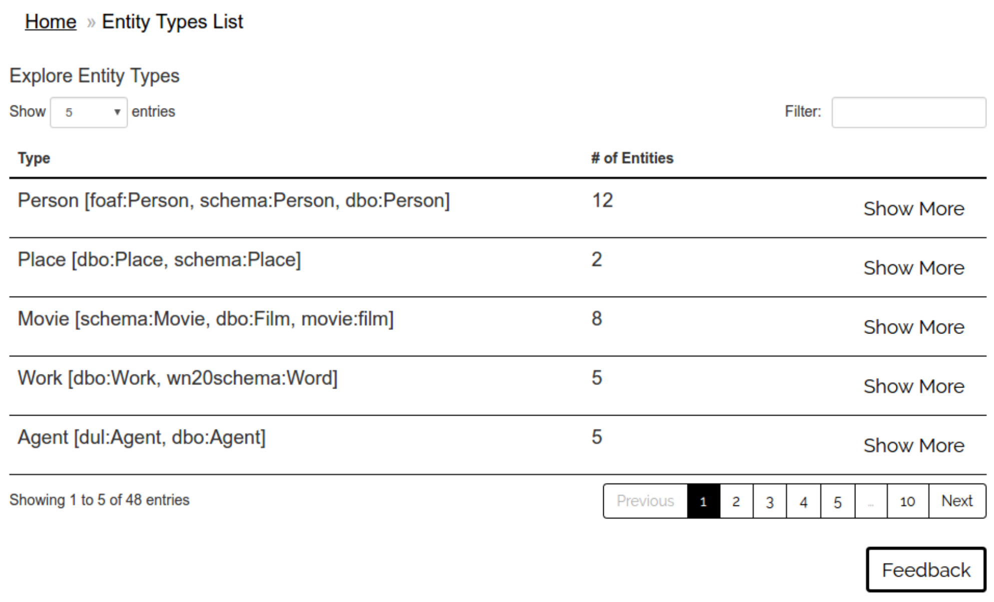
Is the entity type «*Organization*» is expected in the results of search term «*Manchester*»?	Yes [✓ No] Not Sure
Is the entity type «*Agent*» is expected in the results of search term «*Manchester*»?	Yes [✓ No] Not Sure
Is the entity type «*Place*» is expected in the results of search term «*Manchester*»?	[✓ Yes] No Not Sure
Is the entity type «*City*» is expected in the results of search term «*Manchester*»?	[✓ Yes] No Not Sure
Is the entity type «*Person*» is expected in the results of search term «*Manchester*»?	[✓ Yes] No Not Sure

Showing 1 to 5 of 59 entries

Previous 1 2 3 4 5 … 12 Next

Save Feedback

Figure 5.6: Explicit feedback on the instances of `EntityType` for the results of search term `Manchester`



Home » Entity Types List

Explore Entity Types

Show 5 entries Filter:

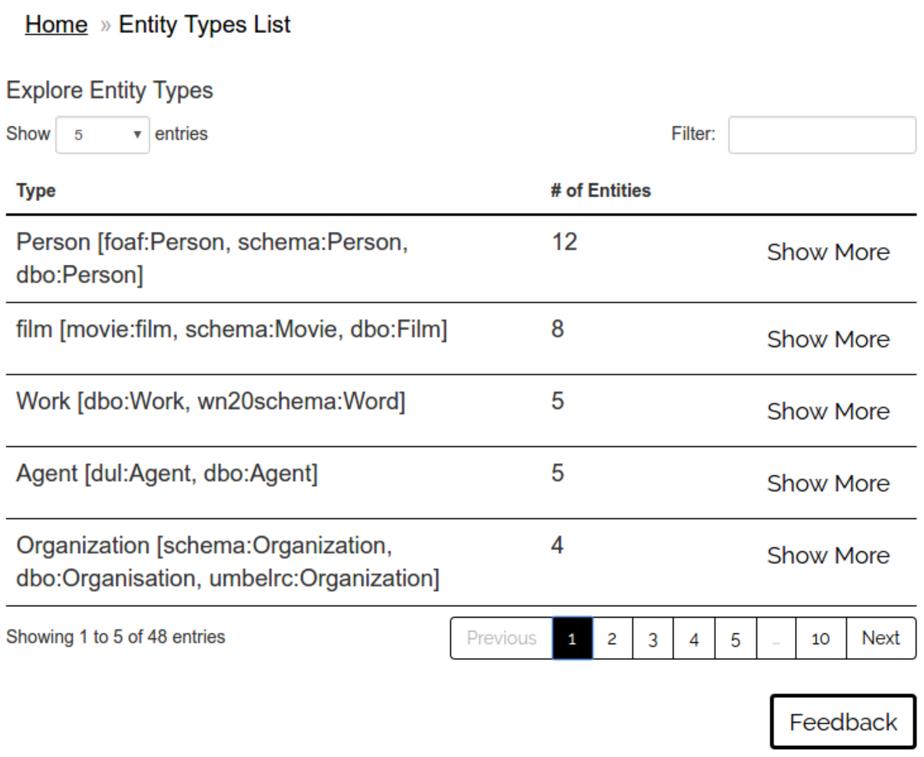
Type	# of Entities	
Person [foaf:Person, schema:Person, dbo:Person]	12	Show More
Place [dbo:Place, schema:Place]	2	Show More
Movie [schema:Movie, dbo:Film, movie:film]	8	Show More
Work [dbo:Work, wn20schema:Word]	5	Show More
Agent [dul:Agent, dbo:Agent]	5	Show More

Showing 1 to 5 of 48 entries

Previous 1 2 3 4 5 … 10 Next

Feedback

Figure 5.7: Inferred instances of `EntityType` for the search term `Casablanca with` feedback


Home » Entity Types List

Explore Entity Types

Show 5 entries Filter:

Type	# of Entities	
Person [foaf:Person, schema:Person, dbo:Person]	12	Show More
film [movie:film, schema:Movie, dbo:Film]	8	Show More
Work [dbo:Work, wn20schema:Word]	5	Show More
Agent [dul:Agent, dbo:Agent]	5	Show More
Organization [schema:Organization, dbo:Organisation, umbelrc:Organization]	4	Show More

Showing 1 to 5 of 48 entries

Figure 5.8: Inferred instances of `EntityType` for the search term `Casablanca` **without** feedback

screenshots, some candidate properties and entity types (e.g., resp., `director` and `Film`) have been fused on the basis of , resp., `SimProperty` and `SimEntity`. Without this, the final table might be more heavily polluted by the natural redundancy one expects in search results.

5.2 A Tool For Integrating LD Search Results

We now describe a tool which provides the user interface that we illustrated in Section 5.1. The tool takes as input a search term and, optionally, a number of ontologies that underpin the domain of the search. It generates a tabular structure that instantiates the metamodel described in Chapter 4. Figure 5.9 shows the data flow of the tool, where components process the inferred results of the PSL model. Each query predicate in the metamodel is mapped onto a relational table in a database that underpins the tool. The main components are the *LD Search Interface*, the

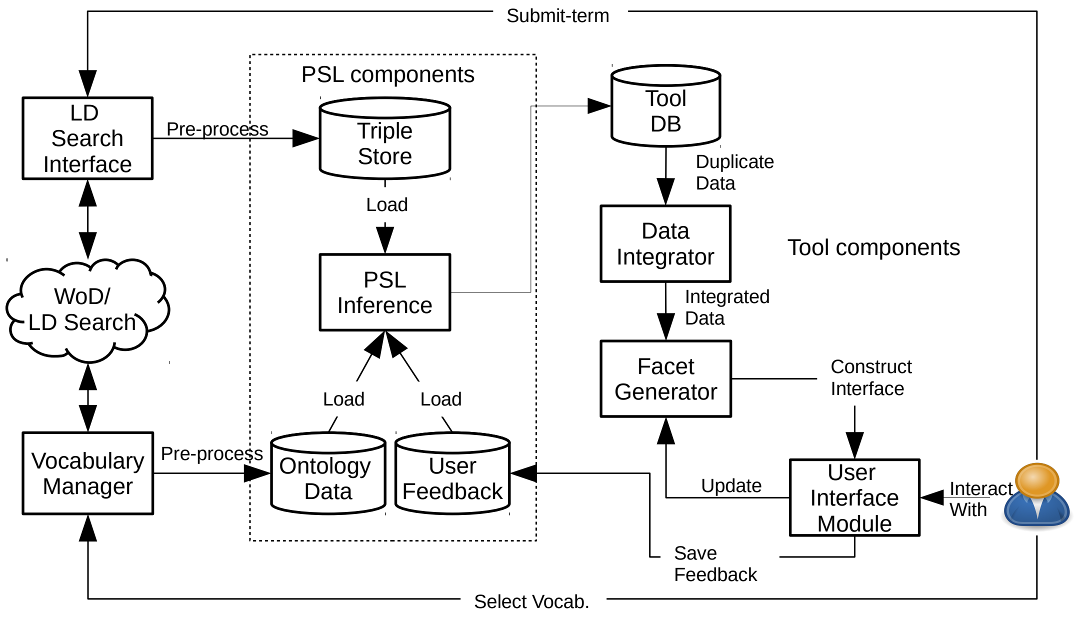


Figure 5.9: Dataflow the underpins the tool for integrating LD search results

Vocabulary Manager, the *PSL Inference*, the *Data Integrator*, the *Facet Generator*, and the *User Interface Module*. The following sections describe them in details.

5.2.1 LD Search Interface

The LD Search Interface component takes as input one or more search terms, and passes them to search engines which retrieves a list of resource URIs. As mentioned in Chapter 3, we use the Falcons [Cheng and Qu, 2009] and Sindice [Oren et al., 2008] search engines, both of which provide APIs to expose services to software applications. These APIs return a list of resources the match the search terms. The LD Search Interface component dereferences the returned resources to obtain the RDF of each resources. As discussed in Chapter 3, we obtain the RDF for the top 20 hits from each search engine. The obtained RDF is the pre-processed into `Triple1` and `Triple2` constructs to enable the use of the RDF data in our PSL program.

5.2.2 Vocabulary Manager

The Vocabulary Manager component takes as input one or more URIs of LD vocabularies (e.g., `http://xmlns.com/foaf/0.1/`). It dereferences the given URIs in order to obtain the RDF of the given vocabulary. In this component, we use the Jena API[1] to handle HTTP content negotiation [Fielding et al., 1999]. Note that content negotiation is often implemented by vocabulary publishers to provide both human and machine-processable content via the same URI [Heath and Bizer, 2011]. The RDF of the obtained vocabularies is pre-processed into `OntTriple` construct which is used by our PSL program to obtain ontological evidence as discussed in Chapter 4.

5.2.3 PSL Inference

The PSL Inference component implements the PSL program $PSL(\mathbb{S} \cup \mathbb{F})$. Recall from Chapter 4 that this PSL program assimilates syntactic evidence from LD search results, semantic evidence obtained from vocabularies and user provided feedback evidence. As such, this component takes as input data pertaining to each type of evidence included. We use the PSL implementation[2] to run the inference process using the available evidence which instantiate the query predicates of our target metamodel. The results of the inference is then loaded in a relational database that underpins the tool being described here. For example, the results of `HasProperty` query predicate is loaded in the relational table `HasProperty(E,P,S)`. In this table the `E` attribute corresponds to the domain of the `HasProperty` relation (i.e., an inferred entity type, e.g., `Film`), `P` attribute corresponds to the range of the of the `HasProperty` relation (i.e., an inferred property, e.g., `runtime`) and the `S` attribute corresponds to the inferred probability score of the query atom.

[1]`jena.apache.org`
[2]`github.com/linqs/psl`

Algorithm 5.1 Merge entity types

```
1: function MergeEntityTypes(threshold)
2:     typeList ← SelectTypes()                        ▷ Returns a list
3:     clusters ← ClusterTypes(typeList, threshold)
4:     types ← ∅                                       ▷ initialize to an empty list
5:     for (k,c) ∈ clusters do
6:         types ← types ∪ ChooseCandidateType(c)
7:     end for
8:     return types                                    ▷ Returns a list of candidate entity types
9: end function
```

5.2.4 Data Integrator

The Data Integrator comprises the PSL-driven integration algorithms that aim to improve presentation of the results. It receives as input instantiations of the metatypes `Property`, `EntityType` and `Entity`, as well as instantiations of the corresponding similarity relations (i.e., `SimProperty`, `SimEntityType` and `SimEntity`). It then uses these results to underpin the algorithms that merge tables columns, and tuples according to the identified correspondences between instances of metatypes.

The data fusion procedures described by Algorithms 5.1 and 5.2, are used, respectively, to merge the inferred entity types and properties. Both algorithms use clustering procedures that employs the Union-Find (UF) [Kozen, 1992] data structure for finding groups of similar properties/types. The UF data structure offers an efficient way for grouping a set of elements into a number of disjoint subsets. In our case we use the UF data structure to group properties/types into disjoint sets using the inferred similarity between pairs of properties/types. This data structure places a pair of properties or types in the same group if the value of `SimProperty` or `SimEntityType` is greater than or equal a given threshold. The reason for using clustering instead of just comparing pairs is because there are often cases where more than two instances of `EntityType` or `Property` match (see Figures 5.1 and 5.2 for examples).

Algorithm 5.1 starts be instantiating a list of candidate entity types. The procedure `SelectTypes` in Line 2 returns a list of 2-tuples where the first item in the tuple is a candidate entity type and the second is its inferred probability. Considering the example shown in Figure 5.1, the `SelectTypes` procedure returns the type list shown in Listing 5.1.

```
typeList = [
 (movie:actor, 0.97), (foaf:Person, 0.97),
 (dbo:Film, 0.96)    , (schema:Person, 0.95),
 (dbo:Person,0.95)   , (dbo:Agent, 0.94),
 (dul:Agent, 0.94)   , (schema:Movie, 0.89),
 (movie:film, 0.86)  , (umbelrc:Actor, 0.83),
 (dbo:Athlete, 0.63), (dbo:Wikidata:Q11424, 0.22)]
```

Listing 5.1: The type list that underpins Figure 5.1

In Line 3 of Algorithm 5.1, the procedure `ClusterTypes` groups elements of the returned `typeList` into disjoint subsets using a UF data structure. The `ClusterTypes` procedure takes as input a list of candidate types and a threshold to determine which elements of the `typeList` to be clustered. Assuming that the threshold is 0.5, still considering the example in Figure 5.1, the pairs of candidate types which have an inferred similarity that is greater than on equal 0.5 are shown in Listing 5.2.

```
SimEntityType = [
 (foaf:Person, schema:Person, 0.78),
 (foaf:Person, dbo:Person, 0.78),
 (dul:Agent, dbo:Agent, 0.78),
 (schema:Movie, dbo:Film, 0.65),
 (umbelrc:Actor, movie:actor, 0.61),
 (dbo:Film, movie:Film, 0.56),
 (dbo:Wikidata:Q11424, dbo:Film, 0.53)]
```

Listing 5.2: Inferred probabilities of `SimEntityType` for the results in Figure 5.1

Given the above similarities the clustering procedure produces the 5 clusters shown in Listing 5.3.

```
clusters = {
  1: [(movie:actor, 0.97), (umbelrc:Actor, 0.83)],
  2: [(foaf:Person, 0.97), (schema:Person, 0.95),
      (dbo:Person,0.95)],
  3: [(dbo:Film, 0.96), (schema:Movie, 0.89), (movie:film, 0.86),
      (dbo:Wikidata:Q11424, 0.22)],
  4: [(dbo:Agent, 0.94), (dul:Agent, 0.94)],
  5: [(dbo:Athlete, 0.63)]}
```

Listing 5.3: The clusters produced which underpins the results in Figure 5.1

Algorithm 5.2 Merge properties

```
1: function MergeProperties(type, threshold)
2:     propertyList ← SelectProperties(type)
3:     clusters ← ClusterProperties(propertyList, threshold)
4:     properties ← $\emptyset$
5:     for (k,c) $\in$ clusters do
6:         properties ← properties $\cup$ ChooseCandidateProperty(c)
7:     end for
8:     return properties                    ▷ Returns a list of candidate properties
9: end function
```

For each cluster produced, a single candidate type is chosen in order to generate a label for the corresponding cluster. The label is simply the local name of the resource URI representing the candidate entity type. The `ChooseCandidateType` procedure in Line 6 takes as input a cluster and chooses a candidate type to represent the cluster. This done by choosing a candidate entity with the highest inferred probability (i.e., using the inferred probabilities of `EntityType` query predicate). For example, as shown in Figure 5.1,the selected candidate type for the cluster {`(dbo:Film, 0.96), (schema:Movie, 0.89), (movie:film, 0.86), (dbo:Wikidata:Q11424, 0.22)`} is `dbo:Film`, and the label generated for the cluster is `Film`.

Algorithm 5.2 used to merge inferred properties is similar to Algorithm 5.1 in the sense that it executes a similar set of procedures. However, Algorithm 5.2 is different from Algorithm 5.1 because it takes as input a selected type stemming from interaction with the screen in Figure 5.1. For example, if the user chooses to drill down on the `Film` entity type, the producer `SelectProperties` in Line 2 of Algorithm 5.2 returns a list of inferred properties for the entity type `Film`. The query predicate which underpins this procedure is `HasProperty`. For example, for the `Film` entity type, the property list returned by the producer `SelectProperties` includes, among others, `[(dbo:director, 0.41), (dbo:producer, 0.40), (dbo:runtime, 0.46), (movie:story_contributor, 0.37)]`, where the value of the probability is obtained through the `HasProperty` query predicate. For instance, in this example, `HasProperty(dbo:Film, dbo:director) = 0.41` and `HasProperty(movie:film, movie:story_contributor) = 0.37`. The `ClusterProperties` procedure groups the properties into disjoint subsets. It uses the inferred probabilities of `SimProperty` and a given threshold to cluster similar properties. For cluster for properties produced by `ClusterProperties` procedure, a candidate property is selected based on the inferred probability of `Property` query predicate in our PSL model.

Before generating the data table (e.g., Figure 5.3), tuples that potentially describe the same entity are merged. This is described in Algorithm 5.3. The algorithm takes as an input a selected type and a set of selected properties (i.e., stemming from interaction with the screens in Figures 5.1 and 5.2). In Algorithm 5.3, similar individuals are clustered using the inferred `SimEntity` similarity scores. Each candidate entity is assigned a property values, one for each of the selected properties. The property value that is assigned is determined by the function `BestValue`, which selects the most likely property value using the inference results of `PropertyValue`.

5.2.5 Facet Generator

The Facet Generator is responsible for constructing the user interface in response to user actions. It creates user interface elements that allow the user to explore the search results. The tool has two kinds of facets *type facets* and *property facets*. Type facets allow the user to filter the results through the set of inferred type. Property

Algorithm 5.3 Merge individual entities

```
1: function MergeIndividuals(type, properties, threshold)
2:     individualList ← SelectIndividuals(type)
3:     clusters ← ClusterIndividuals(individualList, threshold)
4:     individuals ← $\emptyset$
5:     for (k,c) $\in$ clusters do
6:         entity ← ChooseCandidateIndividual(c)
7:         for property $\in$ properties do
8:             entity.propertyMap[property] ← BestValue(entity, property)
9:         end for
10:        individuals ← individuals $\cup$ entity
11:    end for
12:    return individuals                    ▷ Returns a list of candidate individuals
13: end function
```

facets allow the user to filter properties of a given entity types. As shown in Section 5.1, these facets are generated using the underlying instantiations of the query predicates of the PSL model. One important aspect of automatic facet generation is ranking the facets so as to determine which ones are more useful [Arenas et al., 2014; Tunkelang, 2009]. A probabilistic framework such as PSL makes it possible to use inferred probabilities for ranking, i.e., this makes it easier for downstream applications, such as the tool presented in here, to decide on usefulness of facets.

5.2.6 User Interface Module

The user interface module interacts with user with a view to obtaining results that satisfy the user's search intention. Through the interaction with the interface, implicit feedback is collected to allow for better inference for future search terms (e.g., as in, property selection screen, Figure 5.2). Further, the user can provide explicit feedback on the results of the search by clicking on the `Feedback` button. Explicit feedback is prompted through a simple yes/no question, as shown in Figure 5.4. The Feedback obtained by the user is fed to the facet generator component for immediate update of the results in addition to being saved for future inferences.

5.3 Summary and Conclusions

This chapter presented an interface that builds on the inference results of the PSL model presented in the previous chapter. The interface allows users to (i) choose between different types represented in the search result, (ii) refine the properties of the types, (iii) merge both properties and types, and; (iv) fuse instances. All of these activities are informed by the probabilities inferred by the PSL program over the search results. Not only this, but the inferred probabilities allows us to rank the user interface elements in a way that is beneficial to the end user. It was shown how the PSL program can drive a user interface by means of which the user can provide feedback that improves future quality, in a pay-as-you-go style. Moreover, the expressiveness of PSL allows the program to express similarity relationships from which, as shown, it is possible to perform immediate duplicate detection and data fusion prior to showing cleaner results to the user.

Chapter 6

Conclusions

This chapter summarises the major research contributions presented in this dissertation in Section 6.1. In Section 6.2 we discuss some limitations and possible future work in this context.

6.1 Summary of Results and Contributions

The number of RDF documents has grown rapidly in the past decade giving rise to the WoD. To address this growth, a number of LD search engines were developed to allow non-expert users to access this source of data. One limitation of LD search engines is that their main focus is in finding resources relevant to a search term given by the user. The search approach does not address the problem of integrating the heterogeneous results of the search. Aimed at providing an improved visual representation of the search results, Sig.ma [Tummarello et al., 2010], addresses the problem of integrating LD search results on individual entities by building a property graph of the results. However, Sig.ma uses heuristics whose only input is syntactic evidence, and accumulates information about a candidate entity from different LD resources without using a principled evidence-assimilation technique. Furthermore, it assumes that the search term describe a single real-world entity, not a collection of different individual entities. The work presented in this dissertation was motivated by the opportunity to complement the work to date on search engines for LD by devising a technique to infer a structure over the returned resources. Specifically, we have adopted SRL approaches to assimilate different kinds of evidence in a principled manner to produce an integrated representation of the results.

This has motivated Objective O1: *To identify, describe and evaluate approaches that allow us to infer, with uncertainty, a structure for LD search results.* The contribution associated with this objective provided a characterization of LD search integration as an SRL problem. More specifically, we have described two rule-bases using MLNs (Chapter 3) and PSL (Chapter 4) which instantiate a meta-model of our target structure. To capture the uncertainty of these rules, we have learned the weights of the rules using a sample of searches and used the learned models to infer (with uncertainty) the entity types, instances, properties in search results.

One question that this dissertation has addressed is *how to incorporate additional sources of evidence, along with syntactic evidence, into the task of integrating LD search results?* Data integration techniques have taken advantage of evidence that is encoded in domain ontologies and conceptual hierarchies [Hu et al., 2011; Nikolov et al., 2012; Saïs et al., 2009; Scharffe et al., 2009]. Furthermore, several approaches have been proposed to use human feedback to train models that improve data integration quality over approaches that use syntactic means only [Demartini et al., 2013; Isele and Bizer, 2013; Kejriwal and Miranker, 2015]. This has led us to Objective O2: *To extend the approaches in O1 to take advantage of the knowledge encoded by the domain ontologies that are used to describe LD sources with a view to improving the structure inferred in O1* and Objective O3: *To explore the impact of incorporating user feedback as an additional source of evidence to the approaches explored in O1.* The resulting contribution is the extension of a baseline PSL rule-base with a set of rules that assimilate evidence extracted from domain ontologies and from user feedback. Chapter 4 described an empirical evaluation of this evidence-based approach, in which it is shown how the principled, uniform use of different types of evidence improves integration quality.

The significance of the contributions resulting from Objectives O1 to O3 stems from the novelty of adapting SRL approaches for the task of integrating LD search results. In doing so, we have

- used a domain-independent metamodel that characterizes the inferred structure of the results;
- proposed a set of rules that systematically reconciles different types of evidence with the aim of identifying relevant entities, entity types, attributes and attribute values from the results; and
- indicated how it is possible to extend our approach with domain-specific rules, which have the potential of further improving the integration of LD search results if needed.

One question that arises from theses results is *how to use the instantiated metamodel to present the results to the end user?* To address this question, in Chapter 5 we discussed how we have met Objective O4: *To investigate how the results from O1 to O3 can be used to underpin a user interface to LD search that helps users identify the data that is most relevant to them.* This has resulted in an implementation of a prototype user interface that is driven by the inference results of the PSL model we described in Chapter 4. We showed how the use of a probabilistic framework such as PSL allows us not only to structure the results but also to improve the presentation through principled generation of facets that enable the user to navigate the relevant results. The significance of this contribution stems from the opportunities arising from combining the results of keyword search with automatic means for generating facets using PSL in order to provide an improved report of results. This approach opens up opportunities for principled assimilation of existing ontologies as well as user feedback in relation to LD search results personalization.

In this thesis, we presented two rule-bases that rely on SRL approaches for interpreting and integrating LD search results. The PSL rule-base assimilates different sorts of evidence to inform how to structure search results according to a defined meta-model. Our approach demonstrates the feasibility of SRL based methods in the integration problem that require the principled management of different kinds of uncertain evidence. Our approach is domain generic so the rule-bases can be applied to LD search results which stem from different domains. However, the quality of the inference results obtained by such rule-bases may vary across domains. The kind of weights learned is dependent on the data in the weight learning process. Using a different dataset from the one we presented will most likely produce different weights and thus will produce different inference results from the ones observed in our experiments. Also, quality of the generated reports from the instantiation of our meta-model constructs is dependent on the quality of the input search results of the LD search engine. If some LD search engine produces predominately irrelevant result to the search terms used, this will effect the generated reports. Nonetheless, the SRL approach can be used to assimilate user feedback in a way that incrementally refines the obtained results as demonstrated in our experiments.

6.2 Limitations and Future Work

6.2.1 Application of Feedback

A key strategy in dataspaces is the reliance on user feedback for improving the quality of integrations resulting from a bootstrapping phase based on automated techniques. LD is viewed by many as a large dataspace, where datasets are published by various publishers [Heath and Bizer, 2011; Umbrich et al., 2012; Christodoulou, 2015], and the integration is carried out incrementally through later linkage.

SRL approaches, as demonstrated in this dissertation, are useful for, among other things, assimilation of different sources of evidence. Essentially, our approach of acquiring evidence, including feedback, on the inference results of the our PSL model can, in principle, be used to improve the results of automated LD linkage tasks. For example, feedback on the results of `HasProperty` and `SimProperty` can be used to generate concept-level linkage in heterogeneous datasets. This feedback is needed because, as we have explained in our empirical study in Section 4.2.3, there is a lack of cross-ontology links.

To successfully deal with feedback coming from different user communities, our approach needs to be extended to handle variable quality feedback. The experiment in Section 4.3.3, assumed that feedback on inference results has been obtained from a single user with expertise on the domain of the search. To learn from feedback stemming various of users with possibly conflicting points of view, the rules need to encode features such as the reliability of each type/class of user. In other words, rules need to be specialized to take into account the user that is providing feedback on candidate results. This can be achieved by introducing rule templates for different user class/type and generating rules based on such templates in a similar manner to what was demonstrated in the experiment in Section 4.3.4.

In addition, further work could investigate the impact of certain kinds of feedback on the overall integration quality of search results. As shown, in Section 4.3.3, we obtain feedback on the candidates of various query predicates in our model. However, one should explore the focussed application of feedback on ,e.g., a subset of query predicates, such as the results of `SimEntity`, and the propagate this feedback for making inferences about a different set of query predicates. The expressivity of MLNs and PSL syntax enables us to easily specify how feedback is propagated. This could be beneficial because certain types of feedback could be easier obtain than others, for reasons that include, for example, the lack of expertise. Thus, one could reap greater benefits from feedback that is less expensive to obtain.

6.2.2 User Interface Evaluation

In Chapter 5, we presented an interface to provide the user with an integrated view of the search results based on inferences generated by the PSL program. The key idea was to use the instantiated meta-model to generate facets that enable easy navigation of the search results. Also, inference over similarity relationships was used to merge tables, columns and individuals in the resulting tables. The purpose of the interface was to demonstrate how the inference results could be used to derive a user interface. One possible piece of future work is to carry out a formal evaluation of the proposed interface with real users. An evaluation of the interface could be obtained through a usability study that measured the extent to which the interface facilitates finding cogent, concise and relevant information in the WoD. The baseline for the study could be one of the LD search methods, discussed in Chapter 2. The study could consider subjective aspects (e.g., in terms of presentation style and functionality) of interacting with interface to understand the experience of the end user. The interface could be evaluated along a number of dimensions that could include the time to find relevant answers and the suitability of reported results.

6.2.3 Inference of an Extended Metamodel

The PSL model introduced in Chapter 4 infers various similarity relationships between the instances of the target metamodel. These inferred similarities are mainly used to reduce duplication in the results that are presented to the end user. However, different types of relations are typically involved, e.g., equivalence and subsumption. When we generated the tabular reports, we only considered the former and ignored the latter. Future work could consider extending the expressiveness of the target metamodel by inferring subsumption relationships. This would allow, e.g., improvement in the visual presentation of the results by building hierarchies over the inferred types.

Finally, our generated tabular reports only include single-valued attributes of entities. This is because we do not attempt to infer relationships between candidate entity types, such as the `starring` relationship between the entity types `movie` and `actor`. A possible future direction might consider the inference of relationships between the inferred entity types. This would enable the generation of tabular reports that cater for multivalued attributes.

Bibliography

A. P. Dempster, N. M. Laird, D. B. R. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

Aleksovski, Z., Klein, M., ten Kate, W., and van Harmelen, F. (2006). *Managing Knowledge in a World of Networks: 15th International Conference, EKAW 2006, Poděbrady, Czech Republic, October 2-6, 2006. Proceedings*, chapter Matching Unstructured Vocabularies Using a Background Ontology, pages 182–197. Springer Berlin Heidelberg, Berlin, Heidelberg.

Alshukaili, D., Fernandes, A. A. A., and Paton, N. W. (2015). Interpreting linked data search results using markov logic. In Fischer, P. M., Alonso, G., Arenas, M., and Geerts, F., editors, *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015.*, volume 1330 of *CEUR Workshop Proceedings*, pages 221–228. CEUR-WS.org.

Alshukaili, D., Fernandes, A. A. A., and Paton, N. W. (2016). Structuring linked data search results using probabilistic soft logic. In Groth, P. T., Simperl, E., Gray, A. J. G., Sabou, M., Krötzsch, M., Lécué, F., Flöck, F., and Gil, Y., editors, *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, volume 9981 of *Lecture Notes in Computer Science*, pages 3–19.

Arenas, M., Grau, B. C., Kharlamov, E., Marciuska, S., and Zheleznyakov, D.

(2014). Faceted search over ontology-enhanced RDF data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 939–948.

Bach, S. H., Broecheler, M., Huang, B., and Getoor, L. (2015). Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *arXiv preprint arXiv:1505.04406*.

Bach, S. H., Huang, B., London, B., and Getoor, L. (2013a). Hinge-loss markov random fields: Convex inference for structured prediction. *CoRR*, abs/1309.6813.

Bach, S. H., Huang, B., London, B., and Getoor, L. (2013b). Hinge-loss markov random fields: Convex inference for structured prediction. In Nicholson, A. and Smyth, P., editors, *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. AUAI Press.

Baeza-Yates, R., Calderón-Benavides, L., and González-Caro, C. (2006). The intention behind web queries. In *String processing and information retrieval*, pages 98–109. Springer.

Baeza-Yates, R. and Raghavan, P. (2010). Next Generation Web Search. In Ceri, S. and Brambilla, M., editors, *Search Computing*, volume 5950 of *Lecture Notes in Computer Science*, chapter 2. Springer Berlin Heidelberg.

Bechhofer, S., Yesilada, Y., Stevens, R., Jupp, S., and Horan, B. (2008). Using ontologies and vocabularies for dynamic linking. *Internet Computing, IEEE*, 12(3):32–39.

Beckett, D., Berners-Lee, T., Prud'hommeaux, E., and Carothers, G. (2014). RDF 1.1 Turtle: Terse RDF Triple Language. `https://www.w3.org/TR/turtle/`. Accessed: 2015-09-30.

Belhajjame, K., Paton, N. W., Embury, S. M., Fernandes, A. A. A., and Hedeler, C. (2013). Incrementally improving dataspaces based on user feedback. *Inf. Syst.*, 38(5):656–687.

Belhajjame, K., Paton, N. W., Fernandes, A. A. A., Hedeler, C., and Embury, S. M. (2011). User feedback as a first class citizen in information integration systems. In *CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, pages 175–183. www.cidrdb.org.

Beltagy, I., Erk, K., and Mooney, R. J. (2014). Probabilistic soft logic for semantic textual similarity. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1210–1219.

Berners-Lee, T. (2006). Linked Data. Design Issues. `https://www.w3.org/DesignIssues/LinkedData.html`. Accessed: 2015-09-30.

Berners-Lee, T. and Cailliau, R. (1990). WorldWideWeb: Proposal for a HyperText Project. `https://www.w3.org/Proposal.html`. Accessed: 2015-09-30.

Berners-Lee, T., Hendler, J., Lassila, O., and Others (2001). The semantic web. *Scientific american*, 284(5):28–37.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM.

Boyd, K., Eng, K. H., and Jr., C. D. P. (2013). Area under the precision-recall curve: Point estimates and confidence intervals. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*, pages 451–466.

Brewster, C., Alani, H., Dasmahapatra, S., and Wilks, Y. (2004). Data driven ontology evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal.*

Bröcheler, M., Mihalkova, L., and Getoor, L. (2010). Probabilistic similarity logic. In *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*, pages 73–82.

Castano, S., Ferrara, A., Montanelli, S., and Varese, G. (2011). Ontology and instance matching. In *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution - Bridging the Semantic Gap*, pages 167–195.

Cheng, G. and Qu, Y. (2009). Searching linked objects with falcons: Approach, implementation and evaluation. *Int. J. Semantic Web Inf. Syst.*, 5(3):49–70.

Christodoulou, K. (2015). *On techniques for pay-as-you-go data integration of linked data*. PhD thesis, School of Computer Science, The University of Manchester.

Christodoulou, K., Fernandes, A. A. A., and Paton, N. W. (2015a). *Web Information Systems Engineering – WISE 2015: 16th International Conference, Miami, FL, USA, November 1-3, 2015, Proceedings, Part I*, chapter Combining Syntactic and Semantic Evidence for Improving Matching over Linked Data Sources, pages 200–215. Springer International Publishing, Cham.

Christodoulou, K., Paton, N. W., and Fernandes, A. A. A. (2015b). Structure inference for linked data sources using clustering. *Trans. Large-Scale Data- & Knowledge-Centered Sys.*, 19:1–25.

Croft, W. B., Metzler, D., and Strohman, T. (2009). *Search Engines - Information Retrieval in Practice*. Pearson Education.

Cruz, I. F., Stroe, C., and Palmonari, M. (2012). Interactive user feedback in ontology matching using signature vectors. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 1321–1324. IEEE.

Cyganiak, R., Wood, D., and Wood, M. (2014). RDF 1.1 Concepts and Abstract Syntax. `https://www.w3.org/TR/rdf11-concepts/`. Accessed: 2015-09-30.

Dachselt, R., Frisch, M., and Weiland, M. (2008). Facetzoom: a continuous multi-scale widget for navigating hierarchical metadata. In *Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, 2008, Florence, Italy, April 5-10, 2008*, pages 1353–1356.

D'Aquin, M., Baldassarre, C., Gridinoc, L., Sabou, M., Angeletou, S., and Motta, E. (2007). Watson: supporting next generation semantic web applications.

D'Aquin, M. and Motta, E. (2011). Watson, more than a semantic web search engine. *Semantic Web*, 2(1):55–63.

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and ROC curves. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 233–240.

Demartini, G., Difallah, D. E., and Cudré-Mauroux, P. (2012). Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st international conference on World Wide Web*, pages 469–478. ACM.

Demartini, G., Difallah, D. E., and Cudré-Mauroux, P. (2013). Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *VLDB J.*, 22(5):665–687.

Ding, L., Finin, T. W., Joshi, A., Pan, R., Cost, R. S., Peng, Y., Reddivari, P., Doshi, V., and Sachs, J. (2004). Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 652–659.

Dix, A., Cowgill, R., Bashford, C., McVeigh, S., and Ridgewell, R. (2016). Spreadsheets as user interfaces. In *Proceedings of the International Working Conference on Advanced Visual Interfaces, AVI 2016, Bari, Italy, June 7-10, 2016*, pages 192–195.

Doan, A., Halevy, A. Y., and Ives, Z. G. (2012). *Principles of Data Integration.* Morgan Kaufmann.

Doan, A., Ramakrishnan, R., and Halevy, A. Y. (2011). Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96.

Domingos, P. M. and Lowd, D. (2009a). *Markov Logic: An Interface Layer for Artificial Intelligence.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Domingos, P. M. and Lowd, D. (2009b). *Markov Logic: An Interface Layer for Artificial Intelligence*, chapter 4 Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Domingos, P. M. and Lowd, D. (2009c). *Markov Logic: An Interface Layer for Artificial Intelligence*, chapter 3 Inference. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Dou, Z., Song, R., Yuan, X., and Wen, J.-R. (2008). Are click-through data adequate for learning web search rankings? In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 73–82, New York, NY, USA. ACM.

Duan, S., Fokoue, A., and Srinivas, K. (2010). One size does not fit all: Customizing ontology alignment using user feedback. In *The Semantic Web–ISWC 2010*, pages 177–192. Springer.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16.

Erling, O. and Mikhailov, I. (2009). Faceted views over large-scale linked data. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009.*

Euzenat, J. and Shvaiko, P. (2013). *Ontology Matching, Second Edition.* Springer.

Fakhraei, S., Foulds, J. R., Shashanka, M. V. S., and Getoor, L. (2015). Collective spammer detection in evolving multi-relational social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1769–1778.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874.

Ferrara, A., Nikolov, A., and Scharffe, F. (2011). Data linking for the semantic web. *Int. J. Semantic Web Inf. Syst.*, 7(3):46–76.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext transfer protocol–http/1.1. Request for Comments: 2616.

Fisher, J., Christen, P., and Wang, Q. (2016). Active learning based entity resolution using markov logic. In Bailey, J., Khan, L., Washio, T., Dobbie, G., Huang, J. Z., and Wang, R., editors, *Advances in Knowledge Discovery and Data Mining - 20th Pacific-Asia Conference, PAKDD 2016, Auckland, New Zealand, April 19-22,* *2016, Proceedings, Part II*, volume 9652 of *Lecture Notes in Computer Science*, pages 338–349. Springer.

Frasincar, F., Telea, A., and Houben, G.-J. (2006). *Adapting Graph Visualization Techniques for the Visualization of RDF Data*, pages 154–171. Springer London, London.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139.

Fu, L., Wang, H., Jin, W., and Yu, Y. (2012). Towards better understanding and utilizing relations in dbpedia. *Web Intelligence and Agent Systems*, 10(3):291–303.

Gal, A. (2006). Why is schema matching tough and what can we do about it? *SIGMOD Record*, 35(4):2–5.

Gandon, F. and Schreiber, G. (2014). RDF 1.1 XML Syntax. `https://www.w3.org/TR/rdf-syntax-grammar/`. Accessed: 2015-09-30.

Getoor, L. and Taskar, B. (2007). *Introduction to statistical relational learning*. MIT press.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Giunchiglia, F., Shvaiko, P., and Yatskevich, M. (2005). *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part I*, chapter Semantic Schema Matching, pages 347–365. Springer Berlin Heidelberg, Berlin, Heidelberg.

Guha, R., Brickley, D., and Macbeth, S. (2016). Schema. org: Evolution of structured data on the web. *Communications of the ACM*, 59(2):44–51.

Guha, R. V. and Brickley, D. (2014). RDF Schema 1.1. `https://www.w3.org/TR/rdf-schema/`. Accessed: 2015-09-30.

Gunaratna, K., Thirunarayan, K., Jain, P., Sheth, A., and Wijeratne, S. (2013). A statistical and schema independent approach to identify equivalent properties on linked data. In *Proc. 9th Int. Conf. Semantic Systems*, pages 33–40. ACM.

Haase, P., Horrocks, I., Hovland, D., Hubauer, T., Jimenez-Ruiz, E., Kharlamov, E., Pinkel, J. K. C., Rosati, R., Santarelli, V., Soylu, A., and Others (2013). Optique System: Towards Ontology and Mapping Management in OBDA Solutions. In *Second International Workshop on Debugging Ontologies and Ontology Mappings-WoDOOM13*, page 21.

Halevy, A. Y., Franklin, M. J., and Maier, D. (2006). Principles of dataspace systems. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 1–9.

Harth, A. (2010). Visinav: A system for visual search and navigation on web data. *J. Web Sem.*, 8(4):348–354.

Harth, A., Hogan, A., Umbrich, J., Kinsella, S., Polleres, A., and Decker, S. (2012). Searching and browsing linked data with SWSE. In *Semantic Search over the Web*, pages 361–414.

Harth, A., Umbrich, J., Hogan, A., and Decker, S. (2007). YARS2: A federated repository for querying graph structured data from the web. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.*, pages 211–224.

Heath, T. and Bizer, C. (2011). *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers.

Hedeler, C., Belhajjame, K., Paton, N. W., Campi, A., Fernandes, A. A. A., and Embury, S. M. (2010). Dataspaces. In *Search Computing*, pages 114–134. Springer.

Herman, I., Herman, I., and Patel-Schneider, P. F. (2004). OWL 2 Web Ontology Language Document Overview. `https://www.w3.org/TR/owl2-overview/`. Accessed: 2015-09-30.

Hogan, A. (2014). *Linked Data Management*, chapter Linked data and the Semantic Web standards. CRC Press.

Hogan, A., Harth, A., Passant, A., Decker, S., and Polleres, A. (2010). Weaving the pedantic web. In *Proceedings of the WWW2010 Workshop on Linked Data on the Web, LDOW 2010, Raleigh, USA, April 27, 2010.*

Hogan, A., Harth, A., and Polleres, A. (2009). Scalable authoritative OWL reasoning for the web. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(2):49–90.

Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., and Decker, S. (2012). An empirical survey of linked data conformance. *J. Web Sem.*, 14:14–44.

Hu, W., Chen, J., and Qu, Y. (2011). A self-training approach for resolving object coreference on the semantic web. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 87–96.

Huhns, M. N., Jacobs, N., Ksiezyk, T., Shen, W.-M., Singh, M. P., and Cannata, P. E. (1993). Integrating enterprise information models in carnot. In *Intelligent and Cooperative Information Systems, 1993., Proceedings of International Conference on*, pages 32–42.

Isele, R. and Bizer, C. (2013). Active learning of expressive linkage rules using genetic programming. *J. Web Sem.*, 23:2–15.

Isele, R., Umbrich, J., Bizer, C., and Harth, A. (2010). Ldspider: An open-source crawling framework for the web of linked data. In *Proceedings of the ISWC 2010 Posters & Demonstrations Track: Collected Abstracts, Shanghai, China, November 9, 2010*.

J. Hayes, P. and F. Patel-Schneider, P. (2014). RDF 1.1 Semantics. `https://www.w3.org/TR/rdf11-mt/`. Accessed: 2015-09-30.

Jagadish, H. V., Chapman, A., Elkiss, A., Jayapandian, M., Li, Y., Nandi, A., and Yu, C. (2007). Making database systems usable. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 13–24.

Jansen, B. J., Booth, D. L., and Spink, A. (2008). Determining the informational, navigational, and transactional intent of web queries. *Information Processing & Management*, 44(3):1251–1266.

Kejriwal, M. and Miranker, D. P. (2015). Semi-supervised instance matching using boosted classifiers. In *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, pages 388–402.

Khosravi, H. and Bina, B. (2010). A survey on statistical relational learning. In *Advances in Artificial Intelligence*, pages 256–268. Springer.

Kimmig, A., Bach, S., Broecheler, M., Huang, B., and Getoor, L. (2012). A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, pages 1–4.

Kimmig, A., Mihalkova, L., and Getoor, L. (2015). Lifted graphical models: a survey. *Machine Learning*, 99(1):1–45.

Kiryakov, A., Ognyanov, D., and Manov, D. (2005). OWLIM–a pragmatic semantic repository for OWL. In *Web Information Systems Engineering–WISE 2005 Workshops*, pages 182–192. Springer.

Kot, L. and Koch, C. (2009). Cooperative update exchange in the youtopia system. *PVLDB*, 2(1):193–204.

Kozen, D. C. (1992). *The Design and Analysis of Algorithms*, chapter Union-Find, pages 48–51. Springer New York, New York, NY.

Lowd, D. and Domingos, P. M. (2007). Efficient weight learning for markov logic networks. In Kok, J. N., Koronacki, J., de Mántaras, R. L., Matwin, S., Mladenic, D., and Skowron, A., editors, *Knowledge Discovery in Databases: PKDD 2007, 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, Warsaw, Poland, September 17-21, 2007, Proceedings*, volume 4702 of *Lecture Notes in Computer Science*, pages 200–211. Springer.

Madhavan, J., Cohen, S., Dong, X. L., Halevy, A. Y., Jeffery, S. R., Ko, D., and Yu, C. (2007). Web-scale data integration: You can afford to pay as you go. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 342–350. www.cidrdb.org.

Manning, C. D., Raghavan, P., and Schütze, H. H. (2008). *Introduction to Information Retrieval.* Cambridge University Press.

Manola, F., Miller, E., and McBride, B. (2004). RDF primer. W3C recommendation. `http://www.w3.org/TR/rdf-primer/`. Accessed: 2015-09-30.

McGuinness, D. L. and Van Harmelen, F. (2004). OWL Web Ontology Language Overview. `https://www.w3.org/TR/owl-features/`. Accessed: 2015-09-30.

Miller, G. A. (1995). Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

Mitchell, T., Cohen, W., and et al., E. H. (2015). Never-ending learning. In *Proc 29th AAAI*.

Mitchell, T. M. (1997). *Machine learning*. McGraw Hill series in computer science. McGraw-Hill.

Nandi, A. and Jagadish, H. V. (2011). Guided interaction: Rethinking the query-result paradigm. *PVLDB*, 4(12):1466–1469.

Neville, J. and Jensen, D. (2007). Relational dependency networks. *Journal of Machine Learning Research*, 8:653–692.

Ngomo, A. N. and Auer, S. (2011). LIMES - A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2312–2317.

Niepert, M., Noessner, J., Meilicke, C., and Stuckenschmidt, H. (2011). Probabilistic-logical web data integration. In *Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, pages 504–533.

Nikolov, A., d'Aquin, M., and Motta, E. (2012). Unsupervised learning of link discovery configuration. In *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, pages 119–133.

Niu, F., Zhang, C., Ré, C., and Shavlik, J. W. (2012). Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *Int. J. Semantic Web Inf. Syst.*, 8(3):42–73.

Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., and Tummarello, G. (2008). Sindice.com: a document-oriented lookup index for open linked data. *IJMSO*, 3(1):37–52.

Oren, E., Delbru, R., and Decker, S. (2006). Extending faceted navigation for RDF data. In Cruz, I. F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., and Aroyo, L., editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 559–572. Springer.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120.

Paton, N. W., Belhajjame, K., Embury, S. M., Fernandes, A. A. A., and Maskat, R. (2016). Pay-as-you-go data integration: Experiences and recurring themes. In *SOFSEM 2016*, pages 81–92.

Paton, N. W., Christodoulou, K., Fernandes, A. A. A., Parsia, B., and Hedeler, C. (2012). Pay-as-you-go data integration for linked data: opportunities, challenges and architectures. In *Proceedings of the 4th International Workshop on Semantic Web Information Management, SWIM 2012, Scottsdale, AZ, USA, May 20, 2012*, page 3.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Poon, H. and Domingos, P. M. (2006). Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications* *of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 458–463.

Pujara, J., Miao, H., Getoor, L., and Cohen, W. (2013). Knowledge graph identification. In *The Semantic Web–ISWC 2013*, pages 542–557. Springer.

Quilitz, B. and Leser, U. (2008). Querying Distributed RDF Data Sources with SPARQL. In Bechhofer, S., Hauswirth, M., Hoffmann, J., and Koubarakis, M., editors, *The Semantic Web: Research and Applications*, volume 5021 of *Lecture Notes in Computer Science*, pages 524–538. Springer Berlin Heidelberg.

Refaeilzadeh, P., Tang, L., and Liu, H. (2009). Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer.

Sabou, M., d'Aquin, M., and Motta, E. (2008). *The Semantic Web: Research and Applications: 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008 Proceedings*, chapter SCARLET: SemantiC RelAtion DiscoveRy by Harvesting OnLinE OnTologies, pages 854–858. Springer Berlin Heidelberg, Berlin, Heidelberg.

Sabou, M., Dzbor, M., Baldassarre, C., Angeletou, S., and Motta, E. (2007). Watson: A gateway for the semantic web. In *Poster session of the European Semantic Web Conference, ESWC*.

Saïs, F., Pernelle, N., and Rousset, M. (2009). Combining a logical and a numerical method for data reconciliation. *J. Data Semantics*, 12:66–94.

Sarasua, C., Simperl, E., and Noy, N. F. (2012). Crowdmap: Crowdsourcing ontology alignment with microtasks. In *The Semantic Web–ISWC 2012*, pages 525–541. Springer.

Satpal, S., Bhadra, S., Sellamanickam, S., Rastogi, R., and Sen, P. (2011). Web information extraction using markov logic networks. In *Proceedings of the 17th*

ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1406–1414. ACM.

Scharffe, F., Liu, Y., and Zhou, C. (2009). RDF-AI: an architecture for RDF datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US).*

Schraefel, M. and Karger, D. (2006). The pathetic fallacy of RDF. In *International workshop on the semantic web and user interaction (SWUI)*, volume 2006.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI Magazine*, 29(3):93–106.

Shangguan, Z. and McGuinness, D. L. (2010). Towards faceted browsing over linked data. In *Linked Data Meets Artificial Intelligence, Papers from the 2010 AAAI Spring Symposium, Technical Report SS-10-07, Stanford, California, USA, March 22-24, 2010.*

Shi, F., Li, J., Tang, J., Xie, G., and Li, H. (2009). *The Semantic Web - ISWC 2009: 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, chapter Actively Learning Ontology Matching via User Interaction, pages 585–600. Springer Berlin Heidelberg, Berlin, Heidelberg.

Shin, J., Wu, S., Wang, F., De Sa, C., Zhang, C., and Ré, C. (2015). Incremental knowledge base construction using DeepDive. *Proceedings of the VLDB Endowment*, 8(11):1310–1321.

Sieg, A., Mobasher, B., and Burke, R. (2007). Web search personalization with ontological user profiles. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 525–534, New York, NY, USA. ACM.

Singhal, A. (2012). Introducing the knowledge graph: things, not strings. `https://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html`. Accessed: 2015-08-14.

Singla, P. and Domingos, P. M. (2005). Discriminative training of markov logic networks. In Veloso, M. M. and Kambhampati, S., editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 868–873. AAAI Press / The MIT Press.

Singla, P. and Domingos, P. M. (2006). Entity resolution with markov logic. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pages 572–582.

Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., and Lindström, N. (2014). JSON-LD 1.0: A JSON-based Serialization for Linked Data. `https://www.w3.org/TR/json-ld/`. Accessed: 2015-09-30.

Spragins, J. D. (1965). A note on the iterative application of bayes' rule. *IEEE Trans. Information Theory*, 11(4):544–549.

Teevan, J., Dumais, S. T., and Gutt, Z. (2008). Challenges for supporting faceted search in large, heterogeneous corpora like the web. *Proceedings of HCIR*, 2008:87.

ter Horst, H. J. (2005a). Combining RDF and part of OWL with rules: Semantics, decidability, complexity. In *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings*, pages 668–684.

ter Horst, H. J. (2005b). Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary. *J. Web Sem.*, 3(2-3):79–115.

Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., and Decker, S. (2010). Sig.ma: Live views on the web of data. *J. Web Sem.*, 8(4):355–364.

Tunkelang, D. (2009). *Faceted Search*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers.

Umbrich, J., Karnstedt, M., Parreira, J. X., Polleres, A., and Hauswirth, M. (2012). Linked data and live querying for enabling support platforms for web dataspaces. In *Workshops Proceedings of the IEEE 28th International Conference on Data Engineering, ICDE 2012, Arlington, VA, USA, April 1-5, 2012*, pages 23–28.

Umemoto, K., Yamamoto, T., Nakamura, S., and Tanaka, K. (2012). Search intent estimation from user's eye movements for supporting information seeking. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 349–356, New York, NY, USA. ACM.

Völker, J. and Niepert, M. (2011). Statistical schema induction. In *The Semantic Web*, pages 124–138. Springer.

Volz, J., Bizer, C., Gaedke, M., and Kobilarov, G. (2009). Silk - A link discovery framework for the web of data. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009.*

Wang, H., Liu, Q., Penin, T., Fu, L., Zhang, L., Tran, T., Yu, Y., and Pan, Y. (2009). Semplore: A scalable IR approach to search the web of data. *J. Web Sem.*, 7(3):177–188.

Wang, J., Wang, H., Wang, Z., and Zhu, K. Q. (2012). Understanding tables on the web. In *Conceptual Modeling - 31st International Conference ER 2012, Florence, Italy, October 15-18, 2012. Proceedings*, pages 141–155.

Xu, Y., Gao, Z., Wilson, C., Zhang, Z., Zhu, M., and Ji, Q. (2013). Entity correspondence with second-order markov logic. In Lin, X., Manolopoulos, Y., Srivastava, D., and Huang, G., editors, *Web Information Systems Engineering - WISE 2013 - 14th International Conference, Nanjing, China, October 13-15, 2013, Proceedings, Part I*, volume 8180 of *Lecture Notes in Computer Science*, pages 1–14. Springer.

Zapilko, B. and Mathiak, B. (2014). Object property matching utilizing the overlap between imported ontologies. In *The Semantic Web*, pages 737–751. Springer.

Zhang, Z., Gentile, A. L., Augenstein, I., Blomqvist, E., and Ciravegna, F. (2013). Mining Equivalent Relations from Linked Data. In *ACL (2)*, pages 289–293.

Zhu, M., Gao, Z., Pan, J. Z., Zhao, Y., Xu, Y., and Quan, Z. (2013). Ontology learning from incomplete semantic web data by belnet. In *25th ICTAI*, pages 761–768. IEEE.

Zong, N., Im, D.-H., Yang, S., Namgoon, H., and Kim, H.-G. (2012). Dynamic generation of concepts hierarchies for knowledge discovering in bio-medical linked data sets. In *Proc. 6th ICUIMC*, page 12. ACM.

Appendix A

Computing Area Under Precision-Recall Curve

In the work presented in this thesis, we use the area under precision-recall (AUC) curve to evaluate the performance of our MLNs and PSL models. AUC PR is a summary measure that computed on the basis of two information retrieval (IR) metrics: precision and recall. Precision is a measure of result relevancy, while recall is measure of many truly relevant results are retuned. The AUC is common evaluation metric that is used by the SRL community (e.g., employed by [Singla and Domingos, 2006; Poon and Domingos, 2006; Bach et al., 2013a; Neville and Jensen, 2007; Fakhraei et al., 2015; Pujara et al., 2013]. The AUC is a single point summary of resulting curve. In machine learning, the AUC us used as a heuristic for optimizing machine learning algorithms and for comparing between the performance of different classifiers [Davis and Goadrich, 2006]. The use of AUC is common in settings the involve highly skewed datasets where the number of false positives is exceeds the number of true positives. For example, in our problem there are more things that not similar (e.g., via `SimEntity`) than things that are similar.

To compute the AUC, the PR curve need to be plotted first. This is done by varying the probability thresholds of precision and recall of a probabilistic classifier.

Query Predicate	**Probabiliy**	**Ground Truth**
HasProperty(Person, birthYear)	0.973	1
HasProperty(Person, birthDate)	0.973	1
HasProperty(Work,runtime)	0.967	1
HasProperty(Work, musicComposer)	0.811	0
HasProperty(SpatialThing, lat)	0.645	1
HasProperty(Q728937, numberOfStations)	0.645	0
HasProperty(SportsTeam, manager)	0.645	1
HasProperty(Film, producer)	0.583	1
HasProperty(Film, director)	0.573	1
HasProperty(Person, abstract)	0.44	0
HasProperty(Work, distributor)	0.368	0
HasProperty(MusicalWork, previousWork)	0.335	1
HasProperty(Location, populationMetro)	0.322	0
HasProperty(Organization, season)	0.322	0
HasProperty(Place, speedLimit)	0.322	0
HasProperty(SoccerPlayer, surname)	0.071	1

Table A.1: An excerpt of `HasProperty` results.

The *threshold* (t) determines which propositions in the inference results are labelled positive and which negative. The ones whose probability of being greater than or equal the threshold are positive and the rest are negative. The *precision* (P) and *recall* (R) are computing using the standard IR formulas as follows:

$$Precision = \frac{T_P}{T_P + F_P} \qquad Recall = \frac{T_P}{T_P + F_n}$$

where T_P is the number of *true positives*, F_P is the number of *false positives*, and F_n is the number of *false negatives.*

Example 4 (Computing AUC PR)**.** To illustrate how the AUC is computed, consider the results shown in Table A.1 which shows the some of the results produced by our PSL model for the `HasProperty` query predicate. Given that the distribution of probability scores varies greatly among different query predicates in the interval $[0, 1]$, the thresholds for computing the AUC are determined by the positive atoms of a query predicate. For instance, in this example, the thresholds for which the

Threshold	0.071	0.335	0.573	0.583	0.645	0.967	0.973	*1.0*
Precision	0.562	0.667	0.778	0.750	0.714	1.0	1.0	*1.0*
Recall	1.0	0.889	0.778	0.667	0.556	0.333	0.222	*0.0*

Table A.2: Obtained PR scores for the results shown in Table A.1

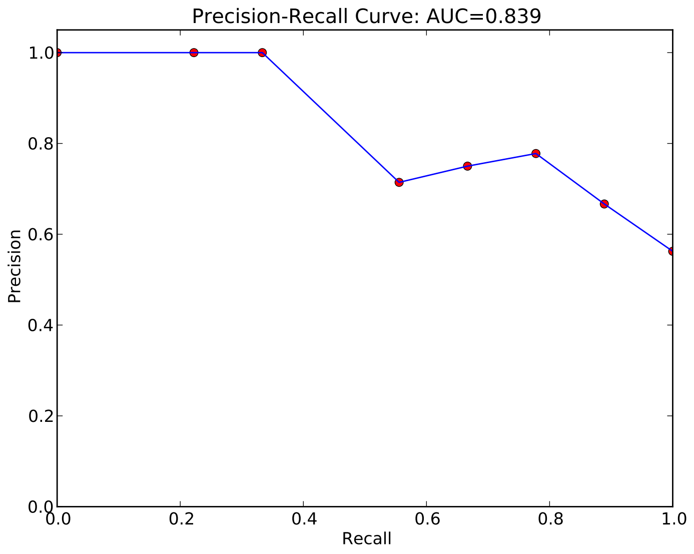


Figure A.1: PR curve for the scores in Table A.2

precision and recall are computed are 0.071, 0.335, 0.573, 0.583, 0.645, 0.967 and 0.973. At $t = 0.071$, $T_P = 9$, $F_P = 7$, $F_n = 0$, thus $P = 0.562$ and $R = 1$. Similarly, At $t = 0.335$, $T_P = 8$, $F_P = 4$, $F_n = 1$, so we get for $P = 0.667$ and for $R = 0.889$. Repeating this calculation for the remaining thresholds we obtain the scores shown in Table A.2. These obtained scores produce the curve shown in Figure A.1. The area under a curve between the upper-left and lower-right points can be found by estimating a definite integral between the two points. We use *scikit-learn* [1] tool kit to estimate the find the value of the area.

[1]scikit-learn.org/

Appendix B

A Methodology for Empirical Study on LD Vocabularies

In Chapter 4 we presented the results of survey for a number of LD vocabularies which showed exiting LOD vocabularies lack the use of expressive meta-vocabulary constructs (such as disjointness, e.g., `owl:disjointWith`, and equivalence, e.g., `owl:equivalentClass` axioms). In here we present the methodology used in our survey of the vocabulary. Note that this methodology is an adaptation of similar method presented in [Christodoulou, 2015].

To conduct the survey, we used LDSpider [Isele et al., 2010], an open-source crawling framework designed for LD. The crawler was seeded with a list of 338 vocabularies that was fetched the Linked Open Vocabularies[1] repository.

The crawler was configured as follows:

- **Syntax**: The crawling was restricted to consider vocabularies that were serialised RDF/XML or Turtle syntax due to the widespread use of these serialisations.

- **Link filter**: Since the goal was to survey the conceptual level of the WoD, the crawler was restricted to follow the T-Box links which ensures that the

[1]lov.okfn.org/dataset/lov/

crawler remains at the conceptual level.

- **Load-balancing**: To ensure that the fetched URIs are distributed across domains, the crawler was configured to fetch 500K URIs from the seed list.
- **Output**: The out of the crawler were stored in a triple store. The Jean TDB [2] store was used in our case.

We ran the crawler with the above configurations for 1 week, in May 2016. The crawling process resulted in retrieving about 200K N-Quads representing ontological statements in the WoD

[2]jena.apache.org/documentation/tdb/