Augmenting Cross-Domain Knowledge Bases Using Web Tables

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

vorgelegt von

Yaser Oulabi
aus Frankfurt am Main

Mannheim, 2020

Dekan: Dr. Bernd Lübcke, Universität Mannheim
Referent: Professor Dr. Christian Bizer, Universität Mannheim
Korreferent: Professor Dr. Heiner Stuckenschmidt, Universität Mannheim

Tag der mündlichen Prüfung: 03. August 2020

Abstract

Cross-domain knowledge bases are increasingly used for a large variety of applications. As the usefulness of a knowledge base for many of these applications increases with its completeness, augmenting knowledge bases with new knowledge is an important task. A source for this new knowledge could be in the form of web tables, which are relational HTML tables extracted from the Web.

This thesis researches data integration methods for cross-domain knowledge base augmentation from web tables. Existing methods have focused on the task of slot filling static data. We research methods that additionally enable augmentation in the form of slot filling time-dependent data and entity expansion.

When augmenting knowledge bases using time-dependent web table data, we require time-aware fusion methods. They identify from a set of conflicting web table values the one that is valid given a certain temporal scope. A primary concern of time-aware fusion is therefore the estimation of temporal scope annotations, which web table data lacks. We introduce two time-aware fusion approaches. In the first, we extract timestamps from the table and its context to exploit as temporal scopes, additionally introducing approaches to reduce the sparsity and noisiness of these timestamps. We introduce a second time-aware fusion method that exploits a temporal knowledge base to propagate temporal scopes to web table data, reducing the dependence on noisy and sparse timestamps.

Entity expansion enriches a knowledge base with previously unknown long-tail entities. It is a task that to our knowledge has not been researched before. We introduce the *Long-Tail Entity Extraction Pipeline*, the first system that can perform entity expansion from web table data. The pipeline works by employing identity resolution twice, once to disambiguate between entity occurrences within web tables, and once between entities created from web tables and existing entities in the knowledge base. In addition to identifying new long-tail entities, the pipeline also creates their descriptions according to the knowledge base schema.

By running the pipeline on a large-scale web table corpus, we profile the potential of web tables for the task of entity expansion. We find, that given certain classes, we can enrich a knowledge base with tens and even hundreds of thousands new entities and corresponding facts.

Finally, we introduce a weak supervision approach for long-tail entity extraction, where supervision in the form of a large number of manually labeled matching and non-matching pairs is substituted with a small set of bold matching rules build using the knowledge base schema. Using this, we can reduce the supervision effort required to train our pipeline to enable cross-domain entity expansion at web-scale.

In the context of this research, we created and published two datasets. The *Time-Dependent Ground Truth* contains time-dependent knowledge with more than one million temporal facts and corresponding temporal scope annotations. It could potentially be employed for a large variety of tasks that consider the temporal aspect of data. We also built the *Web Tables for Long-Tail Entity Extraction* gold standard, the first benchmark for the task of entity expansion from web tables.

Zusammenfassung

Domänenübergreifende Wissensbasen genießen eine stetig steigende Anwendung. Da für viele Anwendungen der Nutzen einer Wissensbasis mit der Vollständigkeit steigt, ist die Vervollständigung von Wissensbasen mit neuem Wissen eine wichtige Problemstellung. Eine mögliche Quelle für solch neues Wissen sind sogenannte Webtabellen, relationale HTML-Tabellen, die aus dem Web gewonnen werden.

Diese Arbeit handelt um Datenintegrationsmethoden für die Erweiterung von Wissensbasen mittels Webtabellen. Bestehende Methoden schränken sich auf Slot-Filling statischer Daten ein. Wir erforschen in dieser Arbeit Methoden, welche zusätzlich Slot-Filling zeitabhängiger Daten und Entity-Expansion ermöglichen.

Für das Slot-Filling zeitabhängiger Daten benötigen wir zeitbewusste Fusionsmethoden. Diese erkennen zwischen widersprüchlichen Werten den Wert, welcher für einen bestimmten Zeitrahmen gültig ist. Zeitbewusste Fusionsmethoden arbeiten deshalb primär daran, Zeitrahmen zu erkennen, welche in Webtabellen fehlen. Wir führen zwei Methoden zur zeitbewussten Fusion ein. In der ersten extrahieren wir Zeitstempel von Webtabellen und deren Webseiten, um diese als Zeitrahmen einzusetzen. Ebenfalls erforschen wir Herangehensweisen, welche die Unvollständigkeit und die Ungenauigkeit dieser Zeitstempel reduzieren. Die zweite Methode propagiert Zeitrahmen von einer temporalen Wissensbasis zu Webtabellen, und reduziert dabei die Abhängigkeit von unvollständigen und ungenauen Zeitstempeln.

Entity-Expansion-Methoden erweitern eine Wissensbasis mit neuen noch unbekannten Long-Tail-Entitäten. Wir führen die *Long-Tail Entity Extraction Pipeline* ein, das erste System, welches Entity-Expansion aus Webtabellen ermöglicht. Die Pipeline identifiziert neuartige Entitäten indem sie Duplikatenerkennung zweifach durchführt, zuerst, um zwischen Erwähnungen von Entitäten in den Webtabellen zu unterscheiden, und dann, um zwischen aus Webtabellen geschaffenen Entitäten und bestehenden Entitäten in der Wissensbasis zu unterscheiden. Die Pipeline erkennt nicht nur neue Long-Tail-Entitäten, sondern verfasst auch ihre Beschreibungen gemäß dem Schema der Wissensbasis.

Wir messen durch das Ausführen der Pipeline auf einen umfassenden Webtabellenkorpus die Eignung von Webtabellen für Entity-Expansion. Wir stellen fest, dass wir je nach Klasse die Wissensbasis mit zehn- oder gar hunderttausenden neuen Entitäten erweitern können.

Zuletzt erarbeiten wir einen Ansatz zur schwachen Überwachung für Long-Tail-Entitäten-Extraktion. Dieser ersetzt Überwachung durch eine hohe Anzahl manuell annotierter Paare mit einer kleinen Anzahl grober Matching-Regeln, die auf das Schema der Wissensbasis aufbauen. Dies reduziert den Überwachungsaufwand und ermöglicht automatisiertes und domänenunabhängiges Entity-Expansion.

Im Rahmen dieser Arbeit haben wir zwei Datensätze erstellt und veröffentlicht. Die *Time-Dependent Ground Truth* enthält zeitabhängige Daten mit mehr als eine Million zeitabhängiger Fakten und Zeitrahmen. Auch haben wir den *Web Tables for Long-Tail Entity Extraction* Goldstandard veröffentlicht, welcher als erster Benchmark für Entity-Expansion aus Webtabellen dient.

Contents

Chapter 1

Introduction

Cross-domain knowledge bases cover a large amount of structured data about entities from varying topical domains within a homogeneous data model. One example of such a knowledge base is DBpedia [Lehmann et al., 2015]. It contains more than 735 thousand geographical places, including cities, mountains, and landmarks, 241 thousand organizations, including companies, universities and sports teams, 1.445 million living and historic persons, 87 thousand movies, 123 thousand music albums and 251 thousand animal species. It describes these entities with hundreds of millions of statements and facts.[1]

Yet, these knowledge bases are in no way complete. Many knowledge bases like DBpedia and YAGO [Hoffart et al., 2013] are extracted from Wikipedia. Their coverage is therefore limited to entities covered by the Wikipedia notability criteria [Oulabi and Bizer, 2019a]. Other knowledge bases, like Freebase and Wikidata, are manually curated. Their coverage might be limited to head knowledge, i.e. such knowledge that is popular and well-known, whereas their coverage of less common knowledge, i.e. from the long-tail, might be low [Dong et al., 2014a].

This thesis is concerned with completing an existing knowledge base, like DBpedia, with long-tail knowledge in the form of new entities and facts. In the context of knowledge base completion, we term any knowledge covered by the knowledge base as head knowledge, whereas any knowledge that is not covered as long-tail knowledge. This definition is based on the assumption that knowledge bases, like DBpedia, already cover notable entities and facts.

One approach to knowledge base completion is the integration of knowledge from external data sources [Paulheim, 2017]. Web tables [Cafarella et al., 2008b, Cafarella et al., 2008a,Lehmberg et al., 2016] are relational HTML tables extracted from the Web. They contain large amounts of structured information, covering a wide range of topics, and potentially describing long-tail knowledge. Web tables are being employed for an increasing number of applications, including set expansion [Wang et al., 2015], question answering [Sun et al., 2016], table extension [Zhang and Balog, 2017], and also knowledge base construction and comple-

[1] https://wiki.dbpedia.org/data-set-2014, accessed 2020-01-16

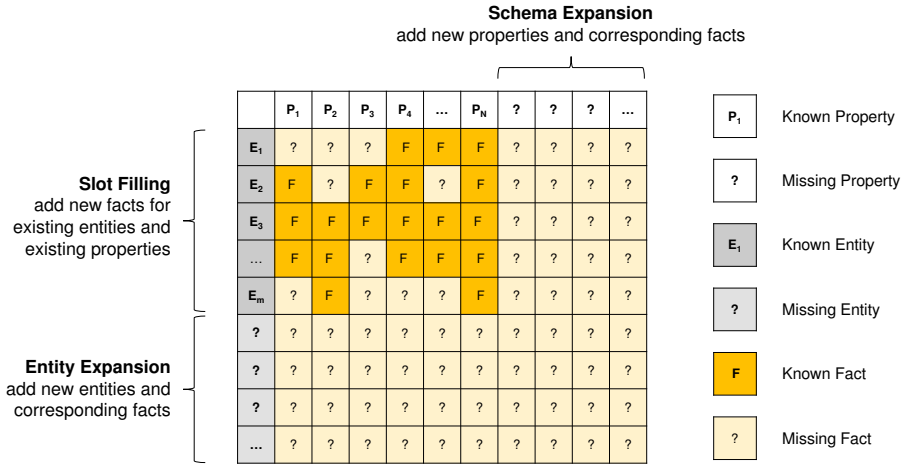

Figure 1.1: Three types of knowledge base augmentation tasks.

tion [Dong et al., 2014a, Sekhavat et al., 2014]. Web tables are thus a promising external source for the task of augmenting cross-domain knowledge bases.

Figure 1.1 outlines three possible knowledge base augmentation tasks through the view of a single topical class of the knowledge base. The rows E_1 to E_m describe existing entities of that class in the knowledge base using facts for the existing properties P_1 to P_n of the schema of that class. A knowledge base does not necessarily cover facts for all combinations of entities and properties [Cao et al., 2020]. The task of augmenting the knowledge base with those facts is termed *slot filling* [Surdeanu, 2013]. Adding new and previously unknown long-tail entities to the knowledge base is termed *entity expansion*, while adding new and previously unknown properties is termed *schema expansion*.

Table 1.1 shows for a selection of DBpedia classes the densities of head properties, which we define as properties that have at least a density of 30%. We find that overall fact density for existing entities is limited, ranging between 60% and 75%. This shows that a large number of new facts can be added to the knowledge base through slot filling.

Table 1.2 additionally compares for a set of classes the number of entities described in DBpedia with the number of entities described in topic-specific data sources. We show numbers for musical artists, musical recordings and American football players, and compare the numbers in DBpedia to the numbers in MusicBrainz[2] and The Football Database[3]. We find that the number of entities covered in DBpedia is much lower, which shows that we can potentially augment DBpedia with new previously unknown entities through entity expansion.

In slot filling, a primary problem is truth discovery, i.e. finding the correct

[2]https://musicbrainz.org/

[3]https://www.footballdb.com/

Table 1.1: Average density of head properties for three classes within DBpedia.

Types	# of Entities	Head Properties	Avg. Density
GridironFootballPlayer	20,751	11	60.16%
Song / Single	52,533	8	75.05%
Settlement	468,986	5	61.59%

Table 1.2: Number of entities in DBpedia compared to domain-specific datasets.

Source	Entity Type	# of Entities
DBpedia	MusicalArtist	45,107
MusicBrainz	Artists	1,501,435
DBpedia	Song / Single	52,533
MusicBrainz	Recordings	20,520,333
DBpedia	AmericanFootballPlayer	13,907
The Football Database	Players	32,560

value given a set of conflicting inputs. This problem can be handled by using data fusion methods, which have been studied extensively in the related work [Dong and Srivastava, 2015b, Dong et al., 2013, Yin et al., 2008]. However, in knowledge bases there exists time-dependent data, where the validity of a fact is dependent on a certain temporal scope, i.e. a point in time or a time range. Slot filling time-dependent data requires fusion methods that are time-aware [Dong et al., 2016]. We find that there is a lack of time-aware fusion methods for web table data.

Entity expansion requires long-tail entity extraction methods, which identify new entities not yet part of the knowledge base by exploiting an external source. They also create for these entities descriptions according to the knowledge base schema. We find that there is a lack of methods for long-tail entity extraction.

This thesis investigates and attempts to overcome the challenges of augmenting a cross-domain knowledge base using web table data. The focus hereby lies on the two tasks of time-aware fusion and long-tail entity extraction.

For time-aware fusion, we first investigate the use of timestamps found in a table and its context. We develop *TT-Weighting*, a time-aware fusion method that considers the locations from which timestamps are extracted to propagate timestamps along them and to exploit their relationship to the slot being filled. We then introduce *Timed-KBT*, a time-aware fusion method that instead of using timestamps, derives temporal scopes by exploiting the overlap of web table data with a temporal knowledge base, as such, overcoming the dependence on timestamps.

In regard to the second task, we introduce the *Long-Tail Entity Extraction Pipeline*, the first system that can find new entities and compile their descriptions from a large corpus of heterogeneous web tables. The pipeline consists of various

components, including schema matching, row clustering, entity creation, and new detection. We run the pipeline on a large-scale web table corpus and find that, given the class, we can extract tens and even hundreds of thousands new entities and corresponding facts. The pipeline however requires manually labeled training data for each class of the knowledge base. We therefore introduce a weak supervision approach that requires per class only a small set of easy-to-create bold matching rules, which allows us to run entity expansion from web tables at web-scale.

1.1 Motivation

In recent years, the importance and relevance of knowledge bases has notably increased. This is shown by the fact, that large tech companies, such as Google, Microsoft, Yahoo!, and Facebook, have created their own knowledge bases and that the "use of these knowledge graphs [i.e. bases] is now the norm rather than the exception" [Mika et al., 2014]. The primary reason for the importance of cross-domain knowledge bases is the large range of applications they can be employed for [Lehmann et al., 2015, Mendes et al., 2012]. Among them are:

- **Web search:** knowledge bases can be used by search engines to enhance search results. Google for example uses the Google Knowledge Graph to understand search queries more effectively, provide summaries in addition to search results, and link to information and entities related to the search.[4]
- **Data mining:** knowledge bases are increasingly being used as background knowledge for data mining. An example of this is the RapidMiner Linked Open Data Extension. It automatically connects local datasets with knowledge bases like DBpedia to gain better insights into certain anomalies in the data [Ristoski et al., 2015].
- **Natural language processing:** knowledge bases can be used to support natural language processing tasks. These include for example:
  - **Question answering:** knowledge bases can be used in answering natural language questions to serve information needs [Ferrucci et al., 2010, Bao et al., 2014, Bordes et al., 2014, Yao and Van Durme, 2014, Wang et al., 2014].
  - **Named entity recognition and disambiguation:** a knowledge base can be used to detect named entities within text, disambiguate them and annotate them with a corresponding entity in the knowledge base. An example of such a system is DBpedia Spotlight [Daiber et al., 2013].
- **Distant supervision:** knowledge bases can be used to reduce supervision effort for learning tasks. They have e.g. been used to distantly supervise

[4] https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html, accessed 2019-08-26.

relation extraction from text [Mintz et al., 2009] and knowledge extraction from semi-structured web data [Lockard et al., 2018, Lockard et al., 2019].

- **Recommender systems:** knowledge bases can also be used as background knowledge to increase the effectiveness of recommender systems. They have e.g. been used to provide semantic representations for items, allowing the exploitation of implicit in addition to explicit relationships in collaborative filtering [Zhang et al., 2016] and to attach semantics to hidden layers of neural networks used for recommendations [Bellini et al., 2017].
- **Integrating data from e-shops:** a knowledge base that covers products can be used to match offers from different online shops, e.g. for finding the offer with the lowest price [Petrovski et al., 2017, Primpeli and Bizer, 2019].

The usefulness of a knowledge base is improved by refining it, which can be done by either correcting errors within the knowledge base, or by extending it [Paulheim, 2017]. In this thesis, we are concerned with refining knowledge bases through augmentation from web table data.

1.2 Alternative Augmentation Approaches

This work is concerned with augmenting a cross-domain knowledge base using external data in the form of web tables. However, there exist ways of extending a knowledge base without needing an external data source. This section will briefly outline these approaches and compare them to external knowledge base augmentation. Primarily, these are internal knowledge base completion methods. We will however also briefly discuss missing data imputation.

Internal Knowledge Base Completion and Relation Prediction

There exist many approaches for internal knowledge base completion. Unlike external approaches, they rely only on the knowledge already contained in the knowledge base. There exist for example symbolic approaches, most prominently logical rule learning [Galárraga et al., 2013, Galárraga et al., 2015, Meilicke et al., 2019b, Meilicke et al., 2019a]. There are also subsymbolic or latent approaches, i.e. using regression [Popescul et al., 2003], knowledge graph embeddings [Bordes et al., 2013, Wang et al., 2017] or neural tensor networks [Socher et al., 2013].

However, these works suffer from one common disadvantage. They can be only used to predict knowledge regarding relations. They can not be used to complete a knowledge base with literal facts, nor can they add new long-tail entities to a class of the knowledge base. While some authors have used literal facts in their methodology, they still can only predict relations [Wang and Huang, 2019].

Additionally, not all types of relations can be predicted. Socher et al. [Socher et al., 2013] introduce a popular approach for internal knowledge base completion using neural tensor networks. In their evaluation set, they make use of the topical

class People from Freebase, from which they however exclude six relation properties, which they deem too difficult for relation prediction. Among the removed properties are for example place of birth, spouse and parents.

For enriching a knowledge base with literal facts, difficult relations, or new entities, we can therefore not rely solely on the knowledge already present within the knowledge base. This is likely because internal knowledge base completion depends on the knowledge to have some form of regularity, which is possibly not the case for these tasks. As such, we require an external source, e.g. in the form of web table data, that explicitly states the knowledge to be added.

Ho et al. introduce a rule-based internal knowledge base completion approach, where they exploit external sources during rule learning and prediction [Ho et al., 2018]. However, their method is still only able to perform relation prediction. Classifying methods solely on whether they use an external source or not is therefore insufficient to fully differentiate between knowledge base completion approaches. We believe that a differentiation based on methodology is more appropriate.

In this thesis, we use data integration methods, whereas the approaches described above make use of prediction. Prediction has its advantages. It does not need new knowledge to be explicitly stated, which could allow it to achieve a high coverage in some enrichment tasks. Data integration on the other hand requires new knowledge to be explicitly stated in a source. However, it could potentially enrich a knowledge bases with literals, difficult relations, and, as we show in this work, long-tail entities.

Missing Data Imputation

There exist in the areas of data mining and statistical analysis approaches to impute missing data. They can add missing values e.g. by using a measure of central tendency, i.e. the mean or the median [Han et al., 2011], machine learning, i.e. regression for continuous values and classification for categorical values, [Larose, 2014], or Bayesian inference [Little, 2020].

While these approaches could potentially be used to enrich knowledge bases with literal values, they have not been applied in the area of knowledge completion. They are used in data mining as a pre-processing step [Han et al., 2011], while in statistical analysis they are employed to reduce the negative effect of missing data, e.g. in the form of bias [Kang, 2013]. Their use might be limited for knowledge base augmentation, as imputed values are by design only estimates. This conflicts with the objective of knowledge bases to contain accurate knowledge.

1.3 Contributions

The contributions of this thesis are the following:

1. We profile extensively the topical domains within the publicly available Web Data Commons 2012 web table corpus. We additionally evaluate the poten-

tial of this corpus for slot filling a knowledge base using Knowledge-Based Trust (KBT), a state-of-the-art fusion method. Existing works profiled either small and not representative corpora or non-public corpora owned by large search engine companies. This research is a *joint contribution*, as it was undertaken with other researchers from the Data and Web Science Group at the University of Mannheim.

2. *TT-Weighting*, a time-aware fusion method that exploits timestamps found in and around a web table. The approach learns relationships between the locations from which timestamps were extracted and a knowledge base property. Using timestamp locations, it also propagates timestamps between web table values. We combine these approaches with KBT to yield an effective time-aware fusion method. Previous works do not consider the effect of the locations of timestamps. They also assign and propagate timestamps solely by web table columns, instead of by web table values.

3. *Timed-KBT*, a time-aware fusion method that uses KBT to estimate the temporal scopes of web table data given its overlap with a temporal knowledge base. Unlike previous works on time-aware fusion, Timed-KBT is independent from timestamps extracted from the web table or its context.

4. The *Long-Tail Entity Extraction (LTEE) Pipeline*, the first system that uses web tables to extract previously unknown long-tail entities along with their descriptions. The pipeline is made up of various components, which include schema matching, row clustering, entity creation and new detection. Related tasks presented in existing research are unsuitable for entity expansion, as they were neither concerned with creating descriptions for new entities, nor with fully disambiguating new entities among each other.

5. Using the LTEE Pipeline, we provide the first large-scale profiling of the potential of web tables for the task of entity expansion. We investigate the number of new entities and corresponding facts we can add to a cross-domain knowledge base and additionally provide an extensive summary of lessons learned for long-tail entity extraction from web table data.

6. A weak supervision approach for long-tail entity extraction, where class-specific manually labeled data in the form of positive and negative entity matches is substituted with a set of class-specific bold matching rules. Compared to previous weak supervision approaches, our rules are easy to create, and we only require a small set of rules for each class. This is important, given the large number of classes within a cross-domain knowledge base.

7. We create and publish two datasets. The *Time-Dependent Ground Truth* (TDGT) is a dataset that is useful for tasks that consider the time-aspect of data. The *Web Tables for Long-Tail Entity Extraction* (T4LTE) dataset is the first gold standard for the evaluation of long-tail entity extraction from web tables. It acts as a benchmark when used with our experimental setup.

1.4 Outline

This section summarizes the content of each chapter within this thesis.

Chapter 2: Cross-Domain Knowledge Bases. This chapter introduces cross-domain knowledge bases, what they are, and what allows them to be useful for many tasks. It then describes existing knowledge bases, focusing especially on how they were constructed. Finally, this chapter describes in-depth DBpedia, highlighting its usefulness for research purposes.

Chapter 3: Web Tables. This chapter provides an introduction to web tables. It first describes web tables within the broader context of unstructured and structured data found on the Web. It then highlights why web tables are especially interesting for cross-domain knowledge base augmentation, while also elaborating the potential challenges of working with web tables. Furthermore, it summarizes the applications that web tables have already been used for. It finally discusses methods of web table corpus extraction and describes the web table corpora made publicly available by the Web Data Commons (WDC) project.

Chapter 4: Data Integration Methods. Augmenting a knowledge base with data extracted from web tables is a data integration task. This chapter investigates the data integration methods required for this task. It first introduces traditional data integration methods, and then describes state-of-the-art methods used for integrating web data and web tables. The chapter then highlights how existing data integration methods are not sufficient for slot filling time-dependent data and entity expansion from web tables. Finally, the chapter introduces a collection of frameworks and methods for data integration that we use throughout this thesis.

Chapter 5: Web Table Profiling. This chapter profiles the potential of web tables for augmenting a cross-domain knowledge base. For this, we match the publicly available 2012 WDC web table corpus to DBpedia. Based on the matching results, we report detailed statistics about classes, properties, and entities. We focus in this chapter on profiling the number of slots in DBpedia that could be filled using web table data. In order to estimate the quality of newly filled slots, we use the Local Closed-World Assumption, which we first confirm empirically. We additionally compare three data fusion strategies and conclude that Knowledge-Based Trust outperforms PageRank and voting-based fusion.

Chapter 6: Exploiting Timestamps for Time-Aware Fusion. This chapter introduces TT-Weighting, a time-aware fusion method that exploits timestamps that appear in different locations in the table and its context. These timestamps are however potentially sparse and noisy. As such, we introduce an approach to propagate timestamps based on their location between values to reduce sparsity, and train machine learning models that weight the importance of timestamp locations

given a certain property of the knowledge base to counteract their noisiness. We investigate the extent to which the performance of static fusion strategies that rely on voting, PageRank, and Knowledge-Based Trust can be improved by incorporating our approaches to exploiting timestamp information. We evaluate the data fusion strategies using a large public corpus of web tables and a ground truth that covers time-dependent properties. We find that our methods effectively propagate timestamps and judge the importance of timestamp locations, as we are able to increase average fusion F1 by 5 percentage points when compared to static fusion strategies.

Chapter 7: Estimating Temporal Scopes Using Knowledge-Based Trust. This chapter introduces Timed-KBT, a time-aware fusion approach that overcomes the dependence on potentially sparse and noisy timestamps by propagating temporal scopes from a temporal knowledge base to web table data using Knowledge-Based Trust. It also derives a trust score that estimates both, the correctness of the data and the validity of the assigned temporal scope. When evaluated on fusing data from a large corpus of web tables for filling missing facts in a temporal knowledge base, we achieve an increase in average $F_{0.25}$ of 19 percentage points when compared to Knowledge-Based Trust and 9 when compared to TT-Weighting. This chapter also introduces and describes the Time-Dependent Ground Truth.

Chapter 8: The Long-Tail Entity Extraction Pipeline. This chapter introduces the LTEE Pipeline, the first system for completing knowledge bases with formerly unknown entities and their descriptions. The chapter outlines the individual components of the pipeline, and investigates alternative methods for schema matching, row clustering and new detection. We additionally introduce and describe the T4LTE dataset, using it for evaluation and training. We achieve an average F1 of 0.83 for both, finding new entities and compiling their descriptions.

Chapter 10: Profiling Web Tables for Entity Expansion. This chapter uses the LTEE Pipeline on the 2012 WDC web table corpus to profile the potential of web tables for the task of entity expansion. We additionally evaluate how well our pipeline performs and discuss a variety of lessons learned. We describe for instance the relationship between the Wikipedia notability criteria and the potential of adding new entities to DBpedia from web table data. We find that we can augment DBpedia with about 14 thousand new football players as well as 187 thousand new songs, respectively described with about 44 and 394 thousand facts.

Chapter 9: Weak Supervision for Long-Tail Entity Extraction. The LTEE Pipeline requires class-specific training data in the form of positive and negative entity matches, which are laborious to create manually. This chapter investigates reducing labeling effort by using weak supervision. In our approach, class-specific supervision is provided in the form of bold matching rules, built using the schema of the knowledge base. By ensembling these rules with a class-agnostic unsu-

pervised matching model, we create a class-specific weakly supervised labeling function. We use this function to label matching and non-matching pairs from randomly chosen web table data and use these pairs to train random forests to finally use within the pipeline. We find that using weak supervision, we perform only slightly worse when compared to classifiers trained using traditional supervision.

Chapter 11: Conclusion. This chapter summarizes the thesis and provides our concluding remarks. It also discusses open questions, future work, and the research impact of this thesis.

1.5 Published Work

Research presented in this thesis has largely been published in the following works:

- **Profiling the 2012 WDC web table corpus**
  Ritze, D., Lehmberg, O., Oulabi, Y., and Bizer, C. (2016). Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 251–261, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.
- **Time-aware fusion using timestamps**
  Oulabi, Y., Meusel, R., and Bizer, C. (2016). Fusing time-dependent web table data. In *Proceedings of the 19th International Workshop on Web and Databases*, WebDB '16, pages 3:1–3:7, New York, NY, USA. Association for Computing Machinery.
- **Time-aware fusion using Timed-KBT**
  Oulabi, Y. and Bizer, C. (2017). Estimating missing temporal meta-information using knowledge-based-trust. In *Proceedings of the 3rd International Workshop on Knowledge Discovery on the WEB*, KDWEB '17, Aachen, Germany. CEUR Workshop Proceedings, RWTH.
- **The Long-Tail Entity Extraction Pipeline and profiling the 2012 WDC web table corpus for entity expansion**
  Oulabi, Y. and Bizer, C. (2019). Extending cross-domain knowledge bases with long tail entities using web table data. In *Proceedings of the 22nd International Conference on Extending Database Technology*, EDBT '19, pages 385–396, Konstanz, Germany. OpenProceedings.org.
- **Weak supervision for long-tail entity extraction**
  Oulabi, Y. and Bizer, C. (2019b). Using weak supervision to identify long-tail entities for knowledge base completion. In *Semantic Systems. The Power of AI and Knowledge Graphs*, proceedings of the 15th International Conference on Semantic Systems, SEMANTiCS '19, pages 83–98, Cham, Switzerland. Springer International Publishing.

Part I

Foundations

Chapter 2

Cross-Domain Knowledge Bases

The primary motivation behind this work is the increasing usefulness and relevance of cross-domain knowledge bases. The number and diversity of applications (see Section 1.1) that can exploit such a knowledge base is a strong advocate for understanding, constructing, maintaining, and augmenting knowledge bases.

While all knowledge bases are possibly useful, not all are equal, nor are they constructed equally. For example, some knowledge bases are openly available and are the result of open, non-profit, and collaborative effort. These include for example DBpedia [Lehmann et al., 2015] and Wikidata [Vrandečić and Krötzsch, 2014]. In strong contrast are knowledge bases that are created, owned, and used by private companies, e.g. the Google Knowledge Graph.

This chapter provides an introduction to cross-domain knowledge bases, outlining first the general concept behind them. It also explores existing knowledge bases and how they were constructed. Finally, it describes in depth DBpedia, a cross-domain knowledge base.

The findings of this chapter are:

- The defining aspects of a knowledge base are interrelations between entities, a graph-based structure, the presence of a well-defined schema or ontology, and the coverage of multiple and versatile topical domains.
- Knowledge bases cover time-dependent data in two ways. They either try to reflect only the most current information (snapshot-based knowledge bases) or they provide a history of temporal facts (temporal knowledge bases). We also argue that, when it comes to time-dependent data, knowledge bases generally cover temporal knowledge, in contrast to listing data.
- Open knowledge bases are primarily constructed through extraction from Wikipedia or collaborative curation. More recently, approaches of constructing knowledge bases from web data have shown to be promising.
- DBpedia is a highly useful and interesting knowledge base due to it being the central hub of linked open data, its manually created high-quality ontology and type hierarchy, and its broad application in related research.

This chapter is structured as follows. The first section defines knowledge bases, demonstrating why they are useful and explaining how they cover time-dependent data. Section 2.2 describes existing knowledge base, while Section 2.3 explores the methods used for their construction. In Section 2.4, we take an in-depth look at DBpedia. The final section summarizes the chapter.

2.1 Understanding Cross-Domain Knowledge Bases

Each knowledge base is different, possibly exhibiting strong suitability for some tasks, and less for others [Ringler and Paulheim, 2017]. Nonetheless this section tries to provide a general introduction to knowledge bases. This is done by first defining what a knowledge base is, and what its most common aspects are. Using an example from DBpedia, this section also attempts to illustrate why knowledge bases are useful for a broad range of tasks. Finally, this section highlights how knowledge bases represent time-dependent data.

2.1.1 Defining a Knowledge Base

Many knowledge bases, e.g. DBpedia [Auer et al., 2007] and YAGO [Suchanek et al., 2007], represent knowledge in the form of triples. A triple consists of subject, predicate, and object, where the subject is an entity with an instance in the knowledge base, and the predicate is a property of the knowledge base schema. The object can either be an entity with an instance, in which case the triple represents a relation between entities, or it can be a literal value, e.g. a date or a number.

However, how a knowledge base represents knowledge, is not what makes it a knowledge base. There are generally four aspects that are used to define a knowledge base [Paulheim, 2017, Ehrlinger and Wöß, 2016]:

1. **Entity Interrelations:** when Google introduced their Knowledge Graph in 2012, they did so with the subtitle "Things, Not Strings".[1] Triples in knowledge bases, where the object is an entity, point to the actual instance of that entity, instead of simply to its name as a literal [Pujara et al., 2013]. Each entity is, barring any mistakes, present only once in the whole knowledge base, and all relations concerning this entity, reference its one instance.
2. **Graph Structure:** due to interrelations of entities, knowledge bases naturally tend to be organized as graphs, with entities as nodes, and relations between entities as edges.
3. **Schema and Ontology:** knowledge bases often employ a schema or an ontology. The latter can be defined as a "formal, explicit specification of a shared conceptualization that is characterized by high semantic expressiveness required for increased complexity" [Ehrlinger and Wöß, 2016].

[1] https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html, accessed 2019-08-26.

4. **Cross-domain:** knowledge bases cover general knowledge, not limited to a single topical domain [Paulheim, 2017, Färber et al., 2018]. As a result, datasets that cover only one domain, are not considered a knowledge base, regardless of how large they are [Paulheim, 2017].

While these four aspects tend to be universal, they are not necessarily complete. Some authors for example include in the defining factors of a knowledge base the representation using the Resource Description Framework (RDF) [Färber et al., 2018]. Being linked to the Linked Open Data cloud as part of the Semantic Web, is also typical for many knowledge bases [Paulheim, 2017].

2.1.2 Conceptualizing a Knowledge Base

To illustrate the concepts behind and the usefulness of a knowledge base, we built Figure 2.1 from actual data in DBpedia. The figure shows a small sample of entities and relations, which we selected on their general proximity in the knowledge base to the German city of Stuttgart.

From the figure, we can first of all see that the knowledge base is structured as a graph. Entities are nodes, while the relations between entities represent edges. More importantly, entity interrelations are prominently visible throughout. For example, all the entities <dbr:Daimler_AG>, <dbr:University_of_Hohenheim>, <dbr:Baden-Württemberg>, <dbr:Mercedes-Benz_Arena_(Stuttgart)>, and additionally <dbr:Die_Fantastischen_Vier> all have something in common with the city of Stuttgart. This relationship is therefore represented in the graph through a relation to the same unique entity <dbr:Stuttgart>.

Some edges point to so-called literals, where the target is not an entity, but a more primitive data type, like a date or a quantity. This is illustrated for example in the properties <dbo:numberOfEmployees>, <dbo:numberOfStudents>, <dbo:birthDate> and <dbo:releaseDate>.

Finally, the figure shows the cross-domain nature of the knowledge base. This small sample contains at least 7 different and unique domains. Starting from the top and moving clockwise we have domains about businesses and companies, universities and higher education, politics and political parties, sports, and music. In the middle the domain about geographical and political divisions is covered. Finally, there are two relations, <dbo:birthDate> and <dbo:birthPlace> about persons. This cross-domain nature of knowledge bases is what sets them apart from specific single-domain datasets like MusicBrainz[2] or GeoNames[3].

The final aspect that defines a knowledge base is the presence of an ontology or a schema. DBpedia has an extensive manually built and maintained ontology. This ontology can also be reflected as a graph, but it does not give itself naturally into a graph structure. We will therefore illustrate the ontology of DBpedia using

[2]https://musicbrainz.org/

[3]http://www.geonames.org/

N-Triple statements [Beckett, 2014], shown in the listing below. We include statements regarding the two entities about persons <dbr:Winfried_Kretschmann> and <dbr:Mario_Gómez>, and the two properties <dbo:birthPlace> and <dbo:birthDate> from Figure 2.1. Any concepts related to the ontology are prefix by <dbo:>, while any actual entities are prefixed by <dbr:>.

```
<dbo:birthDate> <rdfs:domain> <dbo:Person> .
<dbo:birthDate> <rdfs:range> <xsd:date> .

<dbo:birthPlace> <rdfs:domain> <dbo:Person> .
<dbo:birthPlace> <rdfs:range> <dbo:Place> .

<dbr:Winfried_Kretschmann> <rdf:type> <dbo:Politician> .
<dbo:Politician> <rdfs:subClassOf> <dbo:Person> .

<dbr:Mario_Gómez> <rdf:type> <dbo:SoccerPlayer> .
<dbo:SoccerPlayer> <rdfs:subClassOf> <dbo:Person> .

<dbr:Riedlingen> <rdf:type> <dbo:Settlement> .
<dbo:Settlement> <rdfs:subClassOf> <dbo:PopulatedPlace> .
<dbo:PopulatedPlace> <rdfs:subClassOf> <dbo:Place> .
```

The triples with the predicates <rdfs:domain> and <rdfs:range> set the domains and ranges of properties. The domain of both properties <dbo:birthDate> and <dbo:birthPlace> is the class <dbo:Person>, as they both describe properties of natural persons. However, their ranges differ. The range of <dbo:birthDate> is a literal date, while the range of <dbo:birthPlace> is the class <dbo:Place>.

Based on this, both properties, <dbo:birthDate> and <dbo:birthPlace>, should only be used in triples, where the subject is of type <dbo:Person>. However, the types of <dbr:Winfried_Kretschmann> and <dbr:Mario_Gómez> respectively are <dbo:Politician> and <dbo:SoccerPlayer>. It is those types, that are then subclasses of <dbo:Person>. These ontological specifications are denoted in the listing with the properties <rdf:type> and <rdfs:subClassOf>. The city <dbr:Riedlingen> is similarly of type <dbo:Place>, however over two hops in the class hierarchy.

Additional ontological specifications are present for all classes and properties in Figure 2.1. Having this kind of an ontology imparts additional knowledge to the raw relationships between entities, increasing the usefulness of a knowledge base.

From the small sample illustrated in this section, it becomes clear how the four defining aspects of a knowledge base facilitate the representation of cross-domain knowledge in a highly comprehensive yet practical and useful format.

2.1.3 Time-Dependent Data in Knowledge Bases

We differentiate between two types of data, *static* and *time-dependent* [Pal et al., 2012, Dong and Srivastava, 2015b]. For time-dependent data, a fact is only valid given a certain *temporal scope*, i.e. a point in time or a certain time range. This yields a *temporal fact*, which is a fact within the knowledge base that is additionally annotated with a temporal scope [Kuzey and Weikum, 2012].

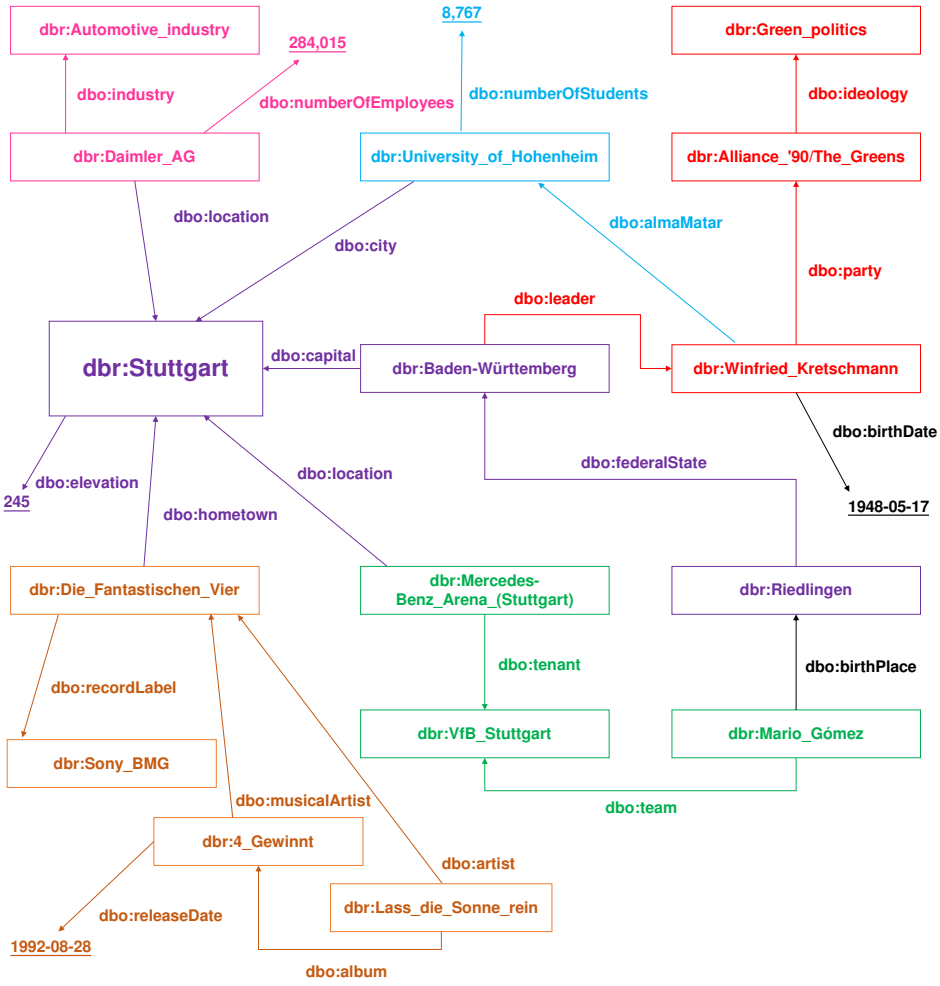

Figure 2.1: Example of entities and their relations in DBpedia for entities related to <dbr:Stuttgart>.

Listing Data versus Temporal Knowledge

When considering time-dependent data, especially in the context of web data, we can additionally differentiate between two types. *Listing data* [Rekatsinas et al., 2014] contains for example information about what the current price of a product in an online store is, whether a certain apartment is still available for rent on a real estate website, or whether there are still tickets for a certain upcoming event. *Temporal knowledge* [Hoffart et al., 2013] contains e.g. the population numbers of a country by year, the teams an athlete played for by season, or a listing of head of states of a certain country along with the time periods in which they held office.

Knowledge bases in general tend to cover temporal knowledge, and not listing data. The two differ primarily by the nature of the entities they describe. Entities

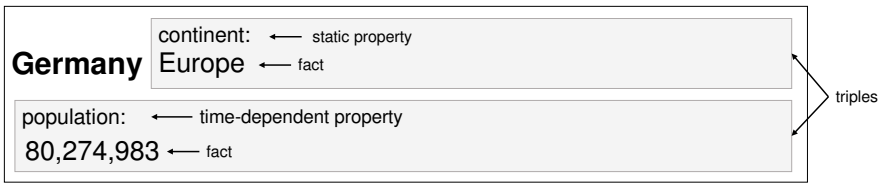

Figure 2.2: Illustration of an entity with a time-dependent property in a *snapshot-based knowledge base*.

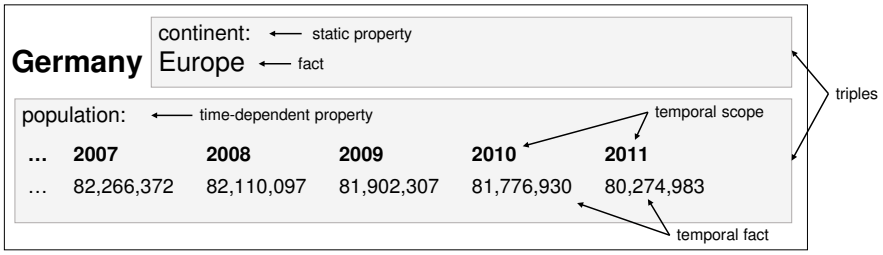

Figure 2.3: Illustration of an entity with a time-dependent property in a *temporal knowledge base*.

in listing data are more temporary in nature. On a real estate site for example, an entity equals the listing for an apartment. While the apartment itself is permanent, the listing exists only temporarily. The same can be said about a product listing for a smartphone, where the offer is likely temporary in its nature. This is not the case for the actual product behind the offer. A knowledge base is likely to cover the smartphone, with properties related to the smartphone itself. However, a knowledge base is unlikely to cover concrete offers for that smartphone.

In the remainder of this work, we focus on temporal knowledge when talking about time-dependent data in the context of knowledge base augmentation.

Snapshot-based versus Temporal Knowledge Bases

Based on how they represent time-dependent data, knowledge bases can be categorized into broadly two types. *Snapshot-based knowledge bases*, e.g. DBpedia, try to reflect only the most recent facts, while on the other hand, *temporal knowledge bases* store time-dependent data as series of temporal facts.

Figures 2.2 and 2.3 illustrate how a snapshot-based and a temporal knowledge base differ in storing time-dependent data. Both figures show population numbers for Germany and illustrate the case where both knowledge bases were released in 2011. We can see how in a temporal knowledge base, all historic facts up to the most current are reflected. Each fact is additionally annotated with a temporal scope. On the other hand, in the snapshot-based knowledge base, only the most current fact is reflected. While this fact is not annotated with any temporal scope, we are aware that the knowledge bases was created in a certain year.

2.2 Existing Cross-Domain Knowledge Bases

There exist many knowledge bases, some of which are openly available, while others are commercial or owned and used solely by one organization. Additionally, knowledge bases differ in the way they are constructed. This section introduces in chronological order existing cross-domain knowledge bases.

Incepted in 1984, **Cyc** [Lenat, 1995] is a commercial and closed knowledge base built using manual curation. It contains a large number of general-knowledge and common-sense axioms, and its main purpose is to enable artificial intelligence tasks. An open subset, OpenCyc, was released in 2001, covering 120 thousand entities and 2.5 million facts, with a schema containing 45 thousand classes and 19 thousand properties [Paulheim, 2017]; it was however discontinued in 2017.

Freebase, DBpedia and YAGO were all launched in 2007. Like Cyc, **Freebase** was maintained by manual curation, but through open collaboration by the crowd. This allowed it to grow fast, from its initial size of 125 million facts [Bollacker et al., 2008], to 1.9 billion in 2015[4]. Freebase initially also imported other datasets including Wikipedia, the Notable Names Database[5] and MusicBrainz [Färber et al., 2018].[6] Google acquired and then shut down Freebase, using it for the creation of the Google Knowledge Graph [Tanon et al., 2016].

DBpedia [Auer et al., 2007, Bizer et al., 2009, Mendes et al., 2012, Lehmann et al., 2015], is automatically constructed from structured data within Wikipedia infoboxes. The extracted knowledge is mapped using handwritten mappings to a manually constructed ontology. Both, the mappings and the ontology, are created collaboratively by the crowd. The latest DBpedia release (2016-10) has 760 classes and 2,865 properties, covering 4.9 million entities and 124 million facts.[7] We describe DBpedia in more details below.

Similarly to DBpedia, **YAGO** [Suchanek et al., 2007, Suchanek et al., 2008] is constructed automatically from Wikipedia and additionally integrates WordNet[8]. YAGO2 considers spatial and temporal dimensions, integrating additionally the GeoNames dataset [Hoffart et al., 2013]. YAGO3 [Mahdisoltani et al., 2015] furthermore uses and fuses Wikipedia from multiple languages for its construction. YAGO3 covers 4.6 million entities and 8.9 million facts.

NELL (Never-Ending Language Learning) [Mitchell et al., 2018,Carlson et al., 2010b, Carlson et al., 2010a] is a knowledge base automatically constructed from unstructured web data. It is the result of a long-running system that uses coupled semi-supervised learning to continuously learn new facts, discover new entities, and extend an ontology. One of the latest iterations of NELL (1115) has an ontology with 335 classes and 937 properties and describes about 2 million entities with

[4] https://developers.google.com/freebase/, accessed 2019-06-17
[5] https://www.nndb.com/
[6] http://radar.oreilly.com/2007/03/freebase-will-prove-addictive.html, accessed 2019-06-20
[7] http://downloads.dbpedia.org/2016-10/statistics/, accessed 2020-01-04
[8] https://wordnet.princeton.edu/

2.8 million high confidence facts.[9] This is a sharp increase from a 2016 release, where NELL had an ontology of 285 classes and 425 properties and described only 0.43 million high confidence facts [Paulheim, 2017]. Unfortunately, NELL seemed to have stopped learning, as the last learned fact is from September 2018.[10]

Incepted in 2012, **Wikidata** [Vrandečić and Krötzsch, 2014] employs, like Freebase, open collaborative curation for growth and maintenance. It is owned by the Wikimedia Foundation, which also owns Wikipedia. One unique characteristic of Wikidata is, that it allows conflicting data to be included, which can then be ranked by its quality or other qualifiers, e.g. how recent it is. Additionally, any statement should be supported using references to external sources, assuring credibility. Wikidata has grown fast immensely. While in 2013 it had 14 million entities and 26 million statements, in 2019 it has 57 million entities and 724 million statements.[11] 74.82% of the statements in 2019 were referenced successfully to an external source, and while Wikidata was initially based heavily on Wikipedia, only 7.25% of all statements in 2019 reference Wikipedia as a source. Wikidata also makes extensive use of bots, which are programs that can make edits without requiring human decision-making. Between May 2017 and May 2019 about 448 million edits were made in total. Of these, 271 million (60%) were made by bots.[12]

While the number of statements within Wikidata seems impressive, we have looked at the number of statements per property to gain insights.[13] Out of 6189 properties, the top 68, i.e. any property with more than one million statements, make up 85% of all statements. Considering, within the top 68, only the properties about the domain *publications*, they contain more than 36% of all statements in Wikidata. Properties linking to external ids and those that contain entity type statements both individually make up 9% of all statements in Wikidata. Properties that reference the source, from which a statement is imported, make up 15% of all statements in Wikidata. By looking at the top 68 properties, we found that 69% of all statements in Wikidata either describe meta-information or one single domain. From the 85% of statements covered in the top 68 properties, only 16 percentage points describe other knowledge. These are about 116 million statements.

Recently, knowledge bases have come to exist that are created, owned and used by commercial companies. This includes for example the **Google Knowledge Graph**, first announced in 2012. Afterwards, it was announced that Microsoft[14] and Yahoo! [Blanco et al., 2013] also own such knowledge bases. Not much is known about how these knowledge bases are constructed, but they likely rely on existing public data, e.g. from Wikipedia or Freebase, and other sources [Paulheim, 2017]. They are larger than public knowledge bases, with the Google Knowledge Graph having about 570 million entities and 18 billion facts, Microsoft's Satori 300

[9] Numbers derived from datasets available here: http://rtw.ml.cmu.edu/rtw/resources

[10] As of 2019-11-15, retrieved from: http://rtw.ml.cmu.edu/rtw/

[11] https://tools.wmflabs.org/wikidata-todo/stats.php, accessed 2019-06-17

[12] https://stats.wikimedia.org/v2/, queried on 2019-06-17

[13] https://wikidata.org/wiki/Wikidata:Database_reports/List_of_properties, accessed 2019-06-18

[14] https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/, acc. 2019-08-26.

million entities and 800 million facts, and Yahoos!'s knowledge base 3.5 million entities and 1.4 billion facts [Paulheim, 2017, Dong et al., 2014a].

Finally, **Knowledge Vault**, a non-open knowledge base presented in 2014 by Google, is automatically constructed from web content by exploiting prior knowledge derived from Freebase [Dong et al., 2014a]. The web content used for its creation includes text, web tables, the DOM structure of websites and schema.org annotations within websites. The authors are able to extract 271 million facts for 45 million entities. This is less than the knowledge bases owned by companies like Google, Microsoft and Yahoo!, and also less than the knowledge bases created using collaboration like Freebase and Wikidata, but far larger than other automatically constructed knowledge bases like DBpedia, YAGO and NELL.

In their paper on Knowledge Vault, the authors write on the weaknesses of their approach. They are e.g. unable to model temporal facts. They are also unable to find new entities, and any fact extracted must be about an entity that already exists in Freebase. Considering that one of the promising aspects of using web data for knowledge base construction is the coverage of long-tail knowledge, limiting the entities to those that already exist within a knowledge base significantly reduces the amount of long-tail knowledge that can be extracted from the Web.

Resolving these two weaknesses of the Knowledge Vault approach is the focus of this thesis. We attempt, similarly to Knowledge Vault, to extract knowledge from the Web, using an existing knowledge base as prior knowledge, however with the intent of augmenting that knowledge base. We focus in this thesis first on researching fusion methods for time-dependent data, and then on methods that identify and compile previously unknown long-tail entities from web table data.

2.3 Methods of Knowledge Base Construction

In the previous section we introduced existing knowledge bases chronologically. In this subsection, we identify and describe the specific approaches used to construct those knowledge bases. Among them are:

- **Non-open manual creation and curation.** The knowledge base is created mainly using manual curation without external collaboration, e.g. Cyc. The viability of this approach for fast large-scale construction of cross-domain knowledge bases is however likely limited.
- **Collaborative crowdsourced curation.** Open collaborative curation by the crowd has proven to be able to construct and maintain large-scale cross-domain knowledge bases like Freebase and Wikidata. These bots can also be contributed by the crowd. In addition to exploiting the input from a large number of users, Wikidata also allows the use of bots, which automatically curate the knowledge base without human intervention.
- **Extraction from Wikipedia.** There are two approaches to constructing a knowledge base from Wikipedia. In the first approach, the knowledge base

is regularly completely reconstructed from Wikipedia, which applies to DBpedia and YAGO. Alternatively, the knowledge base is only initially constructed from Wikipedia but then introduces other methods to grow and curate the knowledge base. This applies to Freebase and Wikidata.

- **Integration with large public domain-specific datasets.** This has been performed in the case of Freebase, where datasets like MusicBrainz were imported early on. This is also used by Wikidata, where other topic-specific datasets, even ones that regularly update, are continuously fed into Wikidata using bots. However, this possibly skews the content of the knowledge base into specific domains, as is the case in Wikidata and the domain publications.
- **Automatically constructed from web data.** NELL and the Knowledge Vault are both large-scale cross-domain knowledge bases constructed from web data. There exist other works which aim to extract knowledge from web data [Etzioni et al., 2004, Bühmann et al., 2014, Cafarella et al., 2009b], including open information extraction systems [Niklaus et al., 2018], and works that focus on web tables [Crestan and Pantel, 2010, Gatterbauer et al., 2007, Sekhavat et al., 2014, Cafarella et al., 2009b]. The benefits of this approach are, that it can be automated, and unlike bots, is domain-independent and can curate a knowledge base with cross-domain knowledge.

2.4 Introducing DBpedia

DBpedia is a cross-domain knowledge base extracted from Wikipedia. It contains knowledge about 4.58 million entities, and describes them using 153 million facts, given its 2014 English language release.[15] Throughout our research, we focus mostly on DBpedia as the reference knowledge base to be extended, and otherwise also base ground truths on the DBpedia schema or ontology. This section briefly outlines why DBpedia was chosen, how DBpedia is constructed, and how DBpedia is structured.

DBpedia as Reference Knowledge Base

DBpedia was chosen for this research for multiple reasons. It is one of the first open cross-domain knowledge bases that is still maintained today. In addition to providing releases every couple of years, the DBpedia community is now providing regular snapshots in the form of DBpedia live.[16]

It is also the hub of the Linked Open Data cloud and has links to many other datasets [Färber et al., 2018]. As such, it has also been used for research in the area of Semantic Web [Färber et al., 2018]. More importantly, it has been used extensively for research on web tables [Limaye et al., 2010, Hassanzadeh et al.,

[15] https://wiki.dbpedia.org/services-resources/datasets/dataset-statistics, accessed 2020-01-04
[16] https://wiki.dbpedia.org/online-access/DBpediaLive

Figure 2.4: Example of possibly important facts in DBpedia extracted from Wikipedia using raw infobox extraction, but not covered by the mapping-based extraction.

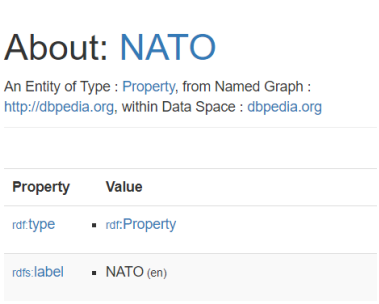

Figure 2.5: Ontological knowledge in DBpedia for the raw infobox extraction property <dbr:nato>.

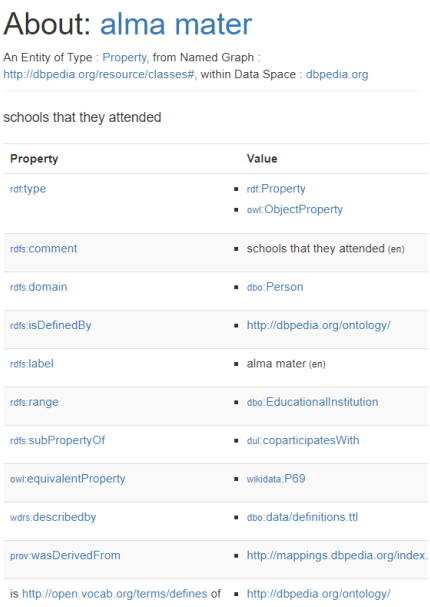

Figure 2.6: Ontological knowledge in DBpedia for the mapping-based extraction property <dbo:almaMatar>.

2015,Ritze et al., 2015,Ritze and Bizer, 2017,Lehmberg and Bizer, 2016,Lehmberg and Bizer, 2017, Lehmberg and Bizer, 2019a, Lehmberg and Bizer, 2019b].

Finally, DBpedia has a manually curated high-quality ontology. The presence of an ontology to which e.g. constructed knowledge can be mapped increases the usefulness of the knowledge base significantly [Suchanek et al., 2008]. As such, the high-quality ontology facilitates the use of DBpedia as a target knowledge base, and possibly reduces the impact of noise and inaccuracies within web table data during knowledge base augmentation.

Wikipedia Infobox Extraction

DBpedia [Lehmann et al., 2015] is constructed by extracting knowledge from Wikipedia. This is done from Wikipedia categories, geographical coordinates, disambiguation pages, and redirects, but primarily from Wikipedia infoboxes.

The infobox extraction is performed in two ways. The *raw extraction* parses the infoboxes of individual Wikipedia pages to create relations with minimal mapping and transformation. This could result in data within the knowledge base that is inconsistent and heterogeneous. The *mapping-based extraction* on the other hand

uses an underlying ontology and mappings from Wikipedia infobox properties to that ontology, to homogenize the knowledge extracted from Wikipedia infoboxes. The ontology and the mappings are both manually created and curated. Properties within DBpedia that were created based on the raw extraction (raw properties) are prefixed with <dbp:>, whereas properties that are part of the ontology (mapping-based properties) are prefixed with <dbo:>.

As the mapping-based extraction requires a manual mapping for each property, some knowledge might only be covered by raw properties. Figure 2.4 shows for a given entity in DBpedia (<dbr:A>) a number of facts for raw properties, that are not covered by the mapping-based extraction, but which however might still be important. Figures 2.5 and 2.6 compare what ontological knowledge is provided for a raw property (<dbr:nato>) and a mapping-based property (<dbo:almaMatar>). We find that we know nothing besides the label for the raw property, but much more for the mapping-based property. Data for raw properties can still be highly useful and it is part of DBpedia, albeit with no ontological knowledge.

Within the English 2014 release of DBpedia, there are 68 million facts about raw properties, 57 million about mapping-based properties, an additional 29 million facts regarding types of entities. It also contains an additional 50 million links to external datasets, e.g. YAGO categories.

Structure and Content

The DBpedia ontology structures classes within a hierarchy, where classes higher in the hierarchy are broader and classes deeper more specific. For example, the class GridironFootballPlayer is a subclass of Athlete, itself a subclass of Person, which itself is a subclass of Agent. Overall, the DBpedia ontology contains more than 680 classes, with the four largest top-level classes within the ontology being Agent, Place, Work and Species.

Table 2.1 shows for a selection of classes the number of entities and facts for the 2014 release of DBpedia. The classes are organized by hierarchy, where classes within a certain level are sorted in descending order by their number of entities. We include for each of the four largest top-level classes one to two subclasses, and for each of those further one to two subclasses. For additional comparison, we also include the next two top-level classes, MeanOfTransportation and Event.[17]

From the table we can first of all see, that DBpedia is quite comprehensive. While the class Agent seems to be somewhat dominant, we can see from its subclasses that in fact it itself is quite versatile. Looking e.g. at its subclasses Person and Organisation, we find that their largest two subclasses cover only 25% and 44% respectively. The class Athlete, the largest subclass within Person, is further divided into a large number of subclasses.

[17] Technically, the next largest top-level classes would be PersonFunction and TimePeriod, however these include intermediary and helper entities like <dbr:Bill_Gates__1> or <dbr:Andrew_Jackson__3>.

Table 2.1: Number of entities and facts for selected DBpedia classes. In the table + denotes a second-level class, while | − denotes a third-level class.

Class	**Entities (thousands)**	**Facts (thousands)**
Agent	1,688	14,318
+ Person	1,445	11,947
| − Athlete	269	3,799
| − Artist	96	1,259
+ Organisation	241	2,372
| − Company	58	518
| − EducationalInstitution	49	549
Place	735	7,436
+ PopulatedPlace	478	5,448
| − Settlement	449	5,236
+ ArchitecturalStructure	150	1,233
Work	411	3,915
+ MusicalWork	180	1,780
| − Album	123	1,102
+ Film	87	891
Species	252	2,021
+ Eukaryote	247	1,990
| − Animal	187	1,503
| − Plant	50	414
MeanOfTransportation	51	467
+ Ship	27	260
+ Aircraft	10	37
+ Automobile	8	135
Event	45	322
+ SocietalEvent	45	322
| − SportsEvent	24	123
| − MilitaryConflict	12	167

We also notice from the table, that the semantic meaning of each level is ambiguous. If we consider the usefulness of a class to be determined by the fact that almost all entities within the class are described with one common set of properties, then for athlete classes at the fourth level, i.e. GridironFootballPlayer, is likely to be the most useful. In contrast, for MeanOfTransportation, the second level might be the most useful, and in fact both Ship and Automobile have no further subclasses. However, this might again be different for PopulatedPlace, also a

second-level class, where Settlement is in regard to number of entities and facts, a highly dominant subclass. This might overshadow other subclasses of PopulatedPlace, i.e. the likely interesting class Country.

DBpedia differentiates between two types of properties. *Object properties* reference other entities, i.e. they represent relations between entities. On the other hand, *data-type properties* reference literals of various types. There are 1079 object properties within the DBpedia ontology, and 1600 data-type properties.

Finally, DBpedia also contains links between entities in DBpedia and entities in 36 external datasets.[18] The number of links to external datasets is about 50 million, of which 41.2 million are links to YAGO categories.[19]

Within DBpedia all entities, properties and classes are referenced using an URI. Entity URIs all start with the prefix <dbr:>[20], while classes and properties within the ontology use the prefix <dbo:>[21].

2.5 Summary

This chapter introduced cross-domain knowledge bases. It first defined knowledge bases, showing how they function and why they can potentially be useful for a large variety of tasks. It additionally outlined the importance of a high-quality ontology or schema within a knowledge base.

This chapter also described time-dependent data, and how it is stored in knowledge bases. We outlined the differences between listing data and temporal knowledge, the latter being the main type of time-dependent data we consider throughout this research. We also showed, that, based on how they store time-dependent data, knowledge bases can be described as snapshot-based or temporal.

Moreover, the chapter described existing knowledge bases and what methods were typically used in their construction. We found that using web data is a promising approach to automatic knowledge bases construction. As such, web data can also be an interesting source for automatic knowledge base augmentation.

Finally, this chapter introduced DBpedia, a cross-domain knowledge base extracted from Wikipedia. We use DBpedia throughout this research due to its popularity, high-quality ontology, and it being used for related research.

Augmenting a knowledge base with additional knowledge is an important task. Web data is a promising source for knowledge base augmentation. It includes web tables, which are relational HTML tables extracted from the Web. These web tables could potentially be used for the large-scale enrichment of cross-domain knowledge bases with long-tail knowledge. We will introduce web tables in the following chapter.

[18] https://wiki.dbpedia.org/Downloads2014, accessed 2020-01-16
[19] https://wiki.dbpedia.org/data-set-2014, accessed 2020-01-16
[20] Unshortened prefix: http://dbpedia.org/resource/
[21] Unshortened prefix: http://dbpedia.org/ontology/

Chapter 3

Web Tables

The primary motivation behind this research is the augmentation of a cross-domain knowledge base with long-tail knowledge. To achieve this, we first need an external source that provides long-tail knowledge for a large variety of domains.

One potential source is the Web. Its size is immense, and it potentially contains large amounts of long-tail knowledge that we can use to augment a cross-domain knowledge base. Harnessing the knowledge found on the Web however is challenging, especially because the traditional Web was made for humans to consume, and not for machines to process automatically.

However, on the Web there also exists some structure. Structured web data, even if only semi-structured, facilitates automatic extraction and processing. Considering for example DBpedia. While it is extracted from Wikipedia, it is not in fact extracted from the main component of Wikipedia, the unstructured text within articles, but from semi-structured knowledge e.g. in the form infoboxes.

Web tables are a type of semi-structured data found on the Web. They are relational HTML tables, which unlike other HTML tables, e.g. those used for layout, contain structured data potentially describing interesting long-tail knowledge. In fact, among the applications of web tables are many knowledge completion tasks, e.g. set expansion [Wang et al., 2015], attribute expansion [Yakout et al., 2012], or table expansion [Zhang and Balog, 2017].

This chapter introduces web tables, their characteristics, and the challenges in and the benefits of using web tables. The findings of this chapter are:

- Among various structured web data sources, web tables are especially interesting as they are not limited to a certain topical domain. Additionally, web tables are easily accessible, as they can be crawled from the surface of the Web. Finally, web tables have a universal relational structure that facilitates parsing and understanding. Methods that perform knowledge base augmentation from web tables are potentially useful for other types of web data, if this data can be transformed into a relational table format.
- Extracting knowledge from web tables however is challenging because web tables are small in size, numerous in number, heterogeneous in their data

model, and noisy in their data quality.

- While extracting web tables from HTML pages is a non-trivial task, there exists a large body of research on this topic. There also exist large-scale public corpora of web tables provided e.g. by the Web Data Commons project.

This chapter is structured as follows. We will first contextualize web tables within the broader context of data on the Web. Section 3.2 then describes in depth the characteristics of web tables and for what tasks they have been used. It also makes the case for choosing web tables as a source of for knowledge base augmentation. Finally, Section 3.3 briefly introduces methods of web table corpus extraction and then describes the publicly available Web Data Commons web table corpora, which we use in this research. The final section summarizes the chapter.

Cafarella et al. [Cafarella et al., 2018] and Zhang and Balog [Zhang and Balog, 2020] provide recent survey papers on web tables in general. This chapter focuses on introducing web tables in the context of knowledge base augmentation.

Datasets / Repository	**Source**	**Statistics**
DBpedia	Extracted from Wikipedia	4.5m entities 0.54b facts
WDC RDFa, Microdata, and Microformats (Nov 2018)[1]	Semantic Annotations	7.1b entities 31.5b facts
WDC Web Table Corpus 2012[2]	Tables in websites	147m tables ⌀ 3.49 columns per table ⌀ 12,41 rows per table
Data.gov[3]	US government	261k datasets (Dec 2019)
EU Open Data Portal[4]	European Union	14k datasets (Dec 2019)
GOVDATA[5]	German federal and state governments	36k datasets (Dec 2019)
Deep Web Databases	Estimation by [He et al., 2007]	43 to 96k databases 7.5k TB of data

Table 3.1: Sources and statistics about a selection of structured web data datasets and repositories.

[1] http://webdatacommons.org/structureddata/2018-12/
[2] http://webdatacommons.org/webtables/
[3] http://www.data.gov/
[4] https://data.europa.eu/
[5] https://www.govdata.de/

3.1 Structured Data on the Web

There are numerous types of data found on the Web. They can first be categorized broadly speaking into two types: unstructured and structured data.

Unstructured data on the Web comes in the form of text extracted from websites. There exist many methods for extracting knowledge from text, e.g. in the form of open information extraction systems [Niklaus et al., 2018]. There also exist approaches to construct knowledge bases from text, including NELL (Never-Ending Learning) [Carlson et al., 2010a,Carlson et al., 2010b,Mitchell et al., 2018] and Knowledge Vault [Dong et al., 2014a].

Structured data on the Web includes for example, in the order of their structuredness, the DOM of websites, web tables, microdata annotations and linked data. These data sources have unlike text some structure and possibly even meta-data that can be exploited. Web tables fall in the middle on the spectrum of structuredness. They can be considered as semi-structured data [Abiteboul, 1997], generally describes structured data that does not follow strict rules and conventions common among database systems. They also lack any meta-data.

To augment a knowledge base from structured web data, we generally require data integration methods in the form of schema matching, identity resolution and data fusion (see Chapter 4). In the case of unstructured web data however, the primary challenge lies in extracting structured facts from natural language text. These facts can then be used to augment a knowledge base, e.g. by semantifying and then integrating them with the knowledge base [Dutta et al., 2015,Dutta, 2016].

The sources of structured and semi-structured data on the Web are numerous and diverse. Among them are:

- **Semantic Annotations in Websites.** Many websites, including shops and travel portals, have started using semantic annotations within their HTML markup [Meusel, 2017]. These annotations provide structured knowledge about a large variety of topics, including product data, and could potentially be extracted and used to enrich knowledge bases [Yu et al., 2019].
- **Web Tables.** There exist hundreds of millions of HTML tables on the Web that contain relational data, potentially covering long-tail knowledge about a large variety of domains. Web tables are the focus of this thesis, and we will introduce them more extensively below.
- **Deep-Web Databases.** A large amount of data on the Web is potentially stored in databases hidden behind query forms [He et al., 2007]. Unlike the surface web, the deep web can not be reached by search engine, so that it is usually neither crawled nor indexed.
- **Public Datasets.** Many governments and other organizations, like research institutes and libraries, publish structured datasets about a variety of domains
- **Linked Open Data Cloud.** There exists a large amount of structured data

on the Web, which is published and interlinked using linked open technologies [Bizer, 2009].

Table 3.1 shows datasets extracted from different types of structured web data. For each dataset, we provide numbers in regard to the size of the data within. We also include statistics about DBpedia for reference. From the table we can see, that regarding their size, many of the structured web data types potentially provide large enough number entities, facts, or values to augment DBpedia. We also see the benefits of the WDC project, as it handles extraction and other pre-processing tasks, enabling others to use the datasets directly for a range of applications.

3.2 Introducing Web Tables

This section introduces web tables, what they are, and what types of web tables exist. We will also elaborate on why we believe that web tables are especially useful for knowledge base augmentation, while also elaborating on the challenges faced when using web tables. Finally, this section summarizes common uses and applications of web tables.

3.2.1 Definition and Structure

Web tables are relational HTML tables extracted from the Web. Unlike tables extracted from deep web databases, web tables do not have to queried using forms, but can be crawled from websites directly, i.e. from the surface of the Web [Cafarella et al., 2008a, Dong and Srivastava, 2015a].

It is important to differentiate between HTML tables that describe relational data, in contrast to those that are used for other purposes. For example, HTML tables are commonly used for layout purposes in websites. Genuine tables can be defined as those which contain structured data and where the "two dimensional grid is semantically significant in conveying the logical relations among the cells" [Wang and Hu, 2002].

Genuine web tables can further be sub-divided into relational, entity and matrix types [Eberius et al., 2015, Ritze, 2017, Balakrishnan et al., 2015]:

- **Relational** tables are those, where each row corresponds to a real-world entity, and each column describes one specific property for the entities within the table. We focus solely on relational tables in of this research.
- **Entity** tables typically describe one entity, where each row represents a different property of that entity. As such, entity tables often have two columns, one for the property name, and one for the corresponding facts [Balakrishnan et al., 2015]. Entity tables are also referred to as attribute-value tables [Yin et al., 2011].
- **Matrix** tables typically describe one single property, and each cell in the table is qualified by both its row and column [Eberius et al., 2015]. An

example of a matrix table is one, where for a row of cities, and columns of years, the cells of the table contain population numbers.

Figures 3.1 to 3.3 show real-world examples for each type of table.

In the 2015 WDC web table corpus (see below), relational tables number more than 90 million, and make up 39% of all genuine tables, whereas entity and matrix tables respectively make up 60% and 1%. The 2012 WDC web table corpus only includes relational tables, which number more than 147 million.

Los Angeles Chargers Roster

Alphabetical | Numerical | Position — 2019

#	Player	Pos	Ht	Wt	Age	College
1	Ty Long	K	6-2	205	26	Alabama-Birmingham
2	Easton Stick	QB	6-1	224	24	North Dakota State
4	Michael Badgley	K	5-10	183	24	Miami
5	Tyrod Taylor	QB	6-1	217	30	Virginia Tech
13	Keenan Allen	WR	6-2	211	27	California
15	Jalen Guyton	WR	6-1	212	22	North Texas
16	Andre Patton	WR	6-2	200	25	Rutgers
17	Philip Rivers	QB	6-5	228	38	North Carolina State
20	Desmond King	DB	5-10	200	25	Iowa
22	Justin Jackson	RB	6-0	199	24	Northwestern
23	Rayshawn Jenkins	DB	6-1	220	25	Miami
25	Melvin Gordon	RB	6-1	215	26	Wisconsin
26	Casey Hayward	DB	5-11	192	30	Vanderbilt

Figure 3.1: An example of a relational web table describing football athletes.[6]

Other Technical Details

Brand Name	Microsoft
Series	Microsoft Surface Pro 6
Item model number	KJT-00016
Operating System	Windows
Item Weight	1.7 pounds
Product Dimensions	11.5 x 7.9 x 0.3 inches
Item Dimensions L x W x H	11.5 x 7.9 x 0.33 inches
Color	Black
Rear Webcam Resolution	8 MP
Processor Brand	Intel
Processor Count	4
Computer Memory Type	DDR3 SDRAM
Flash Memory Size	256 GB
Power Source	Battery
Batteries	1 Lithium Polymer batteries required. (included)

Figure 3.2: An example of an entity table describing a personal computer product.[7]

Gemeindegebiet, Bevölkerung und Bevölkerungsdichte seit 1961
Universitätsstadt Mannheim

Jahr[1)]	Gemeindegebiet[2)]	Bevölkerung insgesamt[3)]	Bevölkerungsdichte	
	ha	Anzahl	EW/km²	Landeswert
1961*	14.494	313.890	2.166	217
1961	14.494	316.007	2.180	219
1962	14.495	318.919	2.200	224
1963	14.495	321.075	2.215	227
1964	14.495	323.444	2.231	231
1965	14.495	328.156	2.264	236
1966	14.495	329.301	2.272	239
1967	14.495	323.744	2.233	240
1968	14.495	326.302	2.251	244
1969	14.495	330.920	2.283	249
1970*	14.495	332.163	2.292	249
1970	14.495	332.378	2.293	250
1971	14.495	330.635	2.281	253
1972	14.495	328.411	2.266	256
1973	14.495	325.386	2.245	258
1974	14.495	320.508	2.211	258
1975	14.495	314.086	2.167	256
1976	14.495	309.059	2.132	255
1977	14.495	305.741	2.109	255
1978	14.495	302.794	2.089	256
1979	14.495	303.247	2.092	257
1980	14.495	304.303	2.099	259
1981	14.495	304.219	2.099	260

Figure 3.3: An example of a matrix table describing statistics about a German city.[8]

[6] https://www.footballdb.com/teams/nfl/los-angeles-chargers/roster/2019?sort=num
[7] https://www.amazon.com/Microsoft-Surface-Pro-Intel-Core/dp/B07K4FMSS8/
[8] https://www.statistik-bw.de/BevoelkGebiet/Bevoelkerung/01515020.tab?R=GS222000

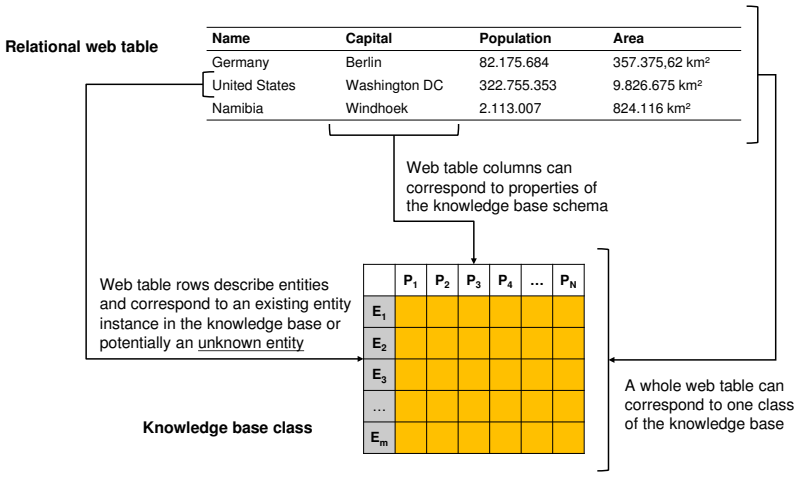

Figure 3.4: Illustration of how knowledge within a web table matches knowledge within a knowledge base.

In this work, we will focus only on relational web tables. Figure 3.4 shows how a relational web table can be matched to a knowledge base. First, all entities and properties described in one web table are assumed to describe knowledge about only one class of the knowledge base. Attribute columns in the web table, can be matched to properties of that class. If the column does not describe structured data, or does so for a property not present in the knowledge base, it can not be matched and therefore not be used to extract knowledge for knowledge base augmentation. Similarly, rows correspond to entities in the knowledge base. However, if a row is not matched to an existing knowledge base entity, it could contain knowledge that could be used to extend a knowledge base with long-tail entities.

3.2.2 Potential and Challenges of Using Web Tables

Web tables have the potential to be especially useful for cross-domain knowledge base augmentation. First, they are not limited a specific topical domain, and could possibly cover any class present in the knowledge base. Other data sources are more likely to be limited to a topical domain. Semantic annotations e.g. have to make use of vocabularies [Meusel et al., 2014], which could potentially limit the domains they cover. More importantly, semantic annotations are often included on websites for a specific purpose, e.g. to highlight information in search engines. This could further limit what kind of knowledge is annotated. Datasets offered by public institutions can be vast in number and size, but often cover information that is in the public interest, containing for example statistics. These datasets, while useful, might not cover the diverse topical domains within a knowledge base. In Chapter 5, we will profile a large public web table corpus and investigate which topical domains are potentially covered by web tables.

When compared to deep web databases, web tables are relative straightforward to extract. They are included on the surface web, and as such they can be extracted from a crawl. Deep web databases, while potentially covering large amounts of data, need to be accessed through query forms.

Relational web tables also provide knowledge in a semi-structured nature that is useful for a large variety of purposes. The relational structure facilitates the parsing, processing and understanding of tables. As we know that, given a row, all cells of that row describe the same entity, we have information that we can exploit to disambiguate this entity. Similarly, as a column always describes one property, this gives us more than one value that we can exploit to detect the column data type and match it to a property in the knowledge base schema. This is not the case for entity tables, where for each property at most one value exists.

Finally, it is quite common to represent structured data in a relational format. Methods developed for relational data, e.g. those we suggest and evaluate in this thesis, could potentially be applied to any web data source, as long as this source is converted into a relational format. Given for example a website that provides a large number of individual entity tables that share a common format, these entity tables could all be merged into one large relational table. Methods built for relational data could then be used to enrich a knowledge base using this table.

Integrating web tables for the purpose of extracting knowledge for knowledge base augmentation however also has its challenges:

- **Size**: web tables are usually small. Further below we describe the WDC web table corpora, which contain relational tables that have on average between 3 and 5 columns, and 12 to 14 rows. The less knowledge covered by a source, the potentially more difficult it becomes is to integrate and extract knowledge from that source.

- **Heterogeneity**: web tables are heterogeneous and inconsistent. Each web table has its own schema and data model, and must therefore be parsed, processed and matched individually. This is made especially difficult by the fact that web tables usually lack meta-data.

- **Amount**: there are billions of HTML tables that can be extracted from the surface web. After applying filtering and classification (see Subsection 3.3.1 below), we are still left with more than hundreds of millions of small relational web table that need to be integrated. As such, working with web tables requires scalable methods.

- **Quality**: web tables, as many web data sources, can generally be considered noisy. We are dealing with tables that were potentially incorrectly extracted, classified and parsed. We could also be dealing with inconsistent and incorrect data. Some knowledge on the Web is purposely fake, but might appear to be genuine, e.g. websites of fantasy sports leagues.

3.2.3 Uses and Applications of Web Tables

This section outlines the variety of uses and applications of web tables. Prominently among them is using web tables to extract new knowledge, e.g. to perform set, attribute, or table expansion. This indicates that web tables are especially suitable for the task of augmenting cross-domain knowledge bases with long-tail knowledge. However, there are also other uses.

Table Search. Web tables can be used to create a table search engine, which allows users to quickly find versatile and specific structured data in the form of tables. Google Tables is an example of a table search engine built using web tables [Balakrishnan et al., 2015]. The authors implemented another version as part of the Google Research Tool, which allows the users of Google Docs and Presentation to search for information while working on a document to integrate structured tables. Another system is OCTOPUS, which uses a search query to find clusters of web tables, relevant to the query [Cafarella et al., 2009a].

Enhancing Web Search. Web tables can be used to enhance web search, providing additional structured information that can be displayed for the user in addition or as part of the search results. This has been implemented for example at Google, where subsets or snippets from web tables are extracted based on a user search query to be shown along with search results [Balakrishnan et al., 2015]. A team at Microsoft developed FACTO, a fact lookup engine that also uses web tables to enhance search results [Yin et al., 2011].

Set Expansion. Set expansion refers to the task of extending an incomplete set of entities with more entities of that set. For this task, various sources can be exploited, among which are e.g. web tables [Wang et al., 2015].

Attribute Expansion. Attribute expansion is the task of finding more attributes, or attribute values, for a given set of entities. This task can also be performed using web tables, as done by the InfoGather [Yakout et al., 2012] and InfoGather+ [Zhang and Chakrabarti, 2013] systems, by the Mannheim Search Join Engine [Lehmann et al., 2015], and the work by Kopliku et al. [Kopliku et al., 2011].

Table Expansion. Table expansion refers to the task of finding both, more entities and properties given a certain input query. There exist works that perform this task using web tables, for example EntiTables [Zhang and Balog, 2017].

Knowledge Base Construction and Augmentation. Given that web tables are useful for set, attribute and table expansion, they naturally lend themselves to the task of knowledge base construction and augmentation. There has been a large corpus of work on matching and interpreting web tables using knowledge bases (see Subsection 4.2.2 of the following Chapter). Some of these works match web tables

specifically to enable knowledge base augmentation [Ritze et al., 2015, Sekhavat et al., 2014, Lehmberg, 2019], however, few works exist that actually use web tables for cross-domain knowledge base augmentation. There exists the work by Dong et al. [Dong et al., 2014a], who use web tables, among other web sources, for constructing the knowledge base Knowledge Vault. However, they only extract knowledge for head entities that already exist within Freebase. Hassanzadeh et al. profile the class overlap between the 2012 WDC web table corpus and classes in multiple knowledge bases, by matching web table columns to classes in knowledge bases [Hassanzadeh et al., 2015]. In Chapter 5, we provide the first large scale profiling of a public web table corpus for the specific task of slot filling a cross-domain knowledge base.

3.3 Web Table Corpora

To make use of web tables for knowledge base augmentation, we need to extract web tables at large-scale from the Web. This extraction is non-trivial. It requires extensive crawling of the Web, and a combination of heuristics and machine-learned methods to filter out non-relational HTML tables.

The Web Data Commons (WDC) project[9] extracts and provides large-scale structured web data datasets to enable research and other purposes. Among the datasets are two web table corpora, which we also use in this thesis.

This section first introduces the task of web table corpus extraction. We will then briefly describe the 2012 and 2015 WDC web table corpora, providing statistics and describing what kind of information they contain.

3.3.1 Web Table Corpus Extraction

Extracting web tables from web pages consists of two steps. First, pages are parsed and HTML tables are extracted based on the `<table>` tag [Wang and Hu, 2002, Cafarella et al., 2008b], which is generally a trivial process. In a second non-trivial step, HTML tables that do not contain structured data need to be filtered out. These tables, also denoted as non-genuine [Wang and Hu, 2002] can for example be used for layout purposes. In contrast, tables that contain structured data are termed genuine. However, we first require a large-scale crawl of the Web, from which can actually extract web tables.

Crawling. Due to the historic lack of available large-scale public crawls of the Web, work with large-scale web table corpora was initially limited to big web companies. For example, the first works using a large number of web tables were conducted at Google using non-public corpora [Cafarella et al., 2008b, Cafarella et al., 2008a]. This however changed with the availability of the Common Crawl[10] [Cafarella et al., 2018, Eberius et al., 2015]. The Common Crawl is the largest pub-

[9] http://webdatacommons.org/

[10] https://commoncrawl.org/

licly available crawl of the Web. The December 2019 release contains 2.45 billion web pages. Based on this, public web table corpora in the form of the Web Data Commons web table corpora or the Dresden Web Table Corpus[11] have become available.

Filtering. Filtering out non-genuine tables can first be done using heuristics or rule-based approaches. For example, tables can be filtered out when they are too small, or because they are part of forms [Cafarella et al., 2008b]. HTML tables can also be filtered out when they contain nested tables, i.e. keeping only the innermost tables [Lehmberg et al., 2015]. In addition, supervised machine learning approaches are used to disambiguate genuine and non-genuine tables [Wang and Hu, 2002, Crestan and Pantel, 2011, Cafarella et al., 2008b]. A machine-learning approach suggested by Eberius et al. [Eberius et al., 2015] classifies tables into a taxonomy. First, tables are classified into genuine, i.e. those with actual structured data, or non-genuine tables, e.g. those used for layout purposes. The genuine tables are further divided into relational, entity, and matrix types.

Table 3.2 demonstrates how considerable the overall filtering of non-relational table is. From the total of all HTML tables extracted, works report that only between 0.9 and 3.4% of tables are genuine relational tables. In the case of the 2015 WDC corpus there are additionally about 140 million and 3 million entity and matrix tables respectively, which together with the relational tables still constitute only about 2.25% of all extracted web tables.

Table 3.2: Number of relational web tables left after filtering, compared to the total number of HTML tables extracted.

Web table corpus	Total Tables	Relational Tables	Ratio
[Wang et al., 2012]	1.95b	66m	3.4%
[Cafarella et al., 2008b]	14.1b	154m	1.1%
WDC Web Table Corpus 2012	11.25b	148m	1.3%
WDC Web Table Corpus 2015	10.24b	90m	0.9%

3.3.2 Web Data Commons Web Table Corpora

The Web Data Commons (WDC) project has published two large-scale web table corpora. Table 3.3 provides general statistics on both of them. The 2012 corpus was extracted using the 2012 Common Crawl dataset, from which 3.3 billion pages were parsed, while the 2015 corpus was extracted using the smaller July 2015 Common Crawl dataset, from which 1.78 billion pages were parsed. Correspondingly, the 2015 corpus is smaller, containing only 90 million relational web tables, while the 2012 corpus contains 147 million relational web tables.

[11] https://wwwdb.inf.tu-dresden.de/misc/dwtc/

Table 3.3: General statistics about the WDC web table corpora.

	2012 corpus	**2015 corpus**
Common Crawl version	2012	July 2015
Parsed pages	3.30b	1.78b
All relational tables	147m	90m
English-language relational tables	92m	51m
Entity tables	-	140m
Matrix tables	-	3m

Table 3.4: Column and row statistics for the WDC web table corpora.

	Min.	**Max.**	**Average**	**Median**
2012 corpus				
columns	2	2,368	3.5	3
rows	1	70,068	12.4	6
2015 corpus				
columns	2	18,106	5.2	4
rows	2	17,033	14.5	6

For the extraction of the 2012 WDC web table corpus, classification-based filtering methods based on those suggested by [Cafarella et al., 2008b], [Crestan and Pantel, 2011], and [Wang and Hu, 2002] were used. For the 2015 WDC web table corpus, the classification methods were based on [Eberius et al., 2015]. In both cases, the authors first also filtered out any tables that contain nested tables, keeping only the innermost tables.

Table 3.3 shows the number of tables within the WDC web table corpora. In addition to the full set of relational tables extracted from the Web, the WDC project also provides a subset of English-language web tables. For the 2012 corpus, any web table from the following top-level domains was considered as English-language: com, org, net, eu and uk. For the 2015 corpus, the language detection was performed using language profiles learned from Wikipedia abstracts. The 2012 English-language subset contains 92 million, while the 2015 subset contains 51 million relational tables.

In case of the 2015 corpus, entity tables and matrix tables were additionally detected. Overall, there are 140 million entity tables, i.e. more than relational tables, and 3 million matrix tables.

Table 3.4 provides statistics about the size of the relational tables within both corpora. On average, the tables are rather small, with only an average number of columns between 3 and 5, and an average number of rows between 12 and 14. However, this also means that each entity in a relational table is described on

average by more than three properties, and each property has values for more than 12 entities. This is likely sufficient to parse, understand and match web tables, and to use them for knowledge base augmentation.

In Chapter 5, we profile the topical domains covered within the 2012 web table corpus and its potential for the task of slot filling missing knowledge in DBpedia. We then use the 2012 web table corpus for our work on entity expansion in Chapters 8 to 10.

The 2015 corpus also contains time and contextual data [Lehmberg et al., 2016], among which are the page title, the table caption, 200 words before as well as after the table, the last modified date on the HTTP header, and sentences around the table that contain timestamps. We use this data in our works regarding time-aware fusion methods in Chapters 6 and 7.

3.4 Summary

Web tables are relational HTML tables extracted from the Web. While they make up only a subset of data found on the Web, they are a highly interesting source for knowledge base augmentation. First, they cover knowledge in a relational format. This potentially makes it easy to parse and understand tables and match them to a knowledge base. More importantly, web tables are not limited to a certain domain, making them especially useful for cross-domain knowledge base augmentation. Finally, methods that use web tables for knowledge base enrichment, could potentially be used to enrich knowledge bases from any other type of structured web data, as long as this data is transformed into a relational format.

However, using web tables for knowledge base augmentation is non-trivial. First of all, each web table has a heterogeneous data model. Moreover, individual web tables are relatively small, while the number of overall web tables is very large. Additionally, web tables are likely to contain noise and low-quality data.

Additionally, to acquire web table corpora, we require a large-scale public crawl of the Web and methods for web table extraction. Not all HTML tables cover relational data, so that extraction methods need to distinguish web tables from other HTML tables, e.g. those used for layout.

However, there exist publicly available web table corpora published as part of the Web Data Commons project. We use these corpora throughout this research, so that we do not need to extract or identify web tables ourselves. The WDC Web Table Corpus 2012 consists of 147 million, while the 2015 consists of 90 million web tables. The 2015 corpus additionally contains contextual information from which we could extract timestamp information to exploit for time-aware fusion.

Augmenting a knowledge base from web tables is a data integration task. It requires therefore data integration methods in the form of schema matching, identity resolution and data fusion. We will introduce and discuss data integration methods for web tables in the next chapter.

Chapter 4

Data Integration Methods

Augmenting a knowledge base from a large corpus of web tables constitutes a data integration task, consisting therefore of schema matching, identity resolution and data fusion. However, due to the nature of web data, traditional data integration methods, e.g. those used for integrating enterprise data sources, are not sufficient [Dong and Srivastava, 2015a]. As each web table has a heterogeneous data model, the data in a table is likely to be noisy, the number of tables is very large, and we aim to augment a cross-domain knowledge base, we need data integration methods that are automated, domain-independent, noise-resistant and scalable.

This chapter introduces the topic of integrating web tables with a knowledge base for the purpose of augmentation. This will be done by first describing existing approaches for web table data integration, discussing which augmentation tasks they enable, and where they fall short. Filling these gaps in integrating web tables with a knowledge base is the primary objective of this thesis. We also introduce a set of frameworks of data integration methods that we use throughout this thesis.

The findings of this chapter are:

- For schema matching, there exist as part of the T2K framework methods that are able to successfully match the schemata of a large number of heterogeneous web tables using a central cross-domain knowledge base. We make use of these methods throughout this thesis.
- For identity resolution, there also exist methods that match entities using a central knowledge base. While these methods allow us to perform slot filling, they are insufficient for performing entity expansion.
- To fuse static data, we can make use of existing fusion techniques that rely on source reliability estimation to determine the correctness of values for fusion. For time-dependent data however, an approach that ensures correctness alone is not sufficient, as we require time-aware fusion methods that additionally ensure validity given a temporal scope. We find that there is little research in regard to time-aware fusion methods for web table data.

This chapter is structured as follows. The next section illustrates the challenges

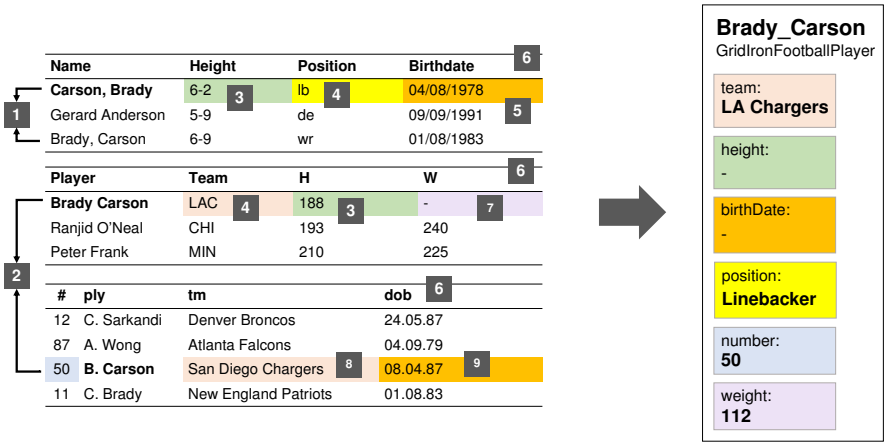

Figure 4.1: Illustration of a slot filling task from web tables.

faced when integrating web tables with a knowledge base. Section 4.2 introduces both, traditional data integration methods and the state of the art in web tables integration. It also highlights how existing methods are insufficient for both, slot filling time-dependent data and entity expansion. In Section 4.3, we finally introduce a set of frameworks with data integration methods that we rely on throughout this thesis. The last section provides a chapter summary.

4.1 Challenges of Integrating Web Table Data

Section 3.2.2 describes the challenges faced when using web tables. In summary, we are dealing with a large number of small web tables of varying quality and with heterogeneous data models. To illustrate more explicitly the challenges in integrating web tables, consider the example of a simple augmentation task in Figure 4.1. The shown task is slot filling for the static, non-time-dependent, properties (`height` and `birthDate`), and the existing entity `Brady_Carson`. As such this task, illustrates a rather simple augmentation task. It involves neither adding new entities to the knowledge base, nor fusing time-dependent data.

The augmentation problem shown in the figure demonstrates some of the challenges faced when integrating web data, of which we have highlighted eight. The highlighted challenges are:

1. Homonym problem for entity labels
2. Synonym problem for entity labels
3. Non-normalized quantities with unknown units
4. Synonym problem for reference data types (lb instead of Linebacker and LAC instead of LA Chargers)

5. Ambiguous date format (day / month unclear)
6. Heterogeneous (Birthdate and dob) and ambiguous (H / W / #) column labels
7. Missing information
8. Outdated label used for reference data types (San Diego Chargers being a historic name of the LA Chargers)
9. Incorrect data

Some of these challenges are not present in traditional data integration tasks or can otherwise be resolved. First, traditional sources cover more meta-information and provide data in a more structured format. They usually cover units of numeric quantities and formats of dates. Reference types are not represented using strings, but usually refer to other entities within the source or to explicitly maintained taxonomies, so that is not necessary to disambiguate which entity is referenced by a certain string literal. While heterogeneous and ambiguous schema labels also exist in traditional data integration methods, there is usually more meta-information present to allow schema matching, i.e. the data type of a property. More importantly, we need to integrate millions of web tables, i.e. match millions of schemata. These numbers are not common in traditional data integration methods. Finally, missing data is also typically explicitly marked as missing in enterprise sources.

Enterprise data is also easier to integrate because it generally covers more data. Enterprise sources first of all cover more information per property, which would allow us to more easily determine date formats or units of quantities. E.g., in the first table in Figure 4.1, we only need one birthdate that occurs on a day of the month beyond the 12th to determine the date format, however there is not enough data in the web table to do this. Traditional sources also tend to cover more properties per entity, allowing us to disambiguate synonyms and homonyms more easily.

4.2 Data Integration for Web Table Data

Data integration consists of three components. In schema matching, the schemata of the sources to be integrated are first aligned and the properties that describe semantically the same knowledge are linked. In identity resolution, also called record linkage or entity matching, the records between sources that describe the same entity are linked. Finally, in data fusion, when conflicting values exist for the same property and entity, the true value is found.

This section outlines for all three components existing methods for web table integration. We will show, where these existing are lacking, and identify the gaps that we aim to close with this research. However,

4.2.1 Traditional Data Integration Methods

Before introducing data integration methods for web table data, we will first briefly introduce traditional approaches to data integration methods.

Traditional Schema Matching Methods

The purpose of schema matching is to align schemata of sources by first, identifying which properties, or columns in the case of web tables, describe semantically equal knowledge, and secondly create a mapping that specifies the semantic relationship between the properties [Madhavan et al., 2001, Dong and Srivastava, 2015a]. Using for example the illustration in Figure 4.1, a schema mapping between the first and second table would relate the columns Height and H. It would also specify that the units of both columns differ, including how they need to be converted for integration.

For schema matching, a mediated schema can be used. The mediated schema is a schema built for the data integration application and consists of the aspects only relevant to the application [Doan et al., 2012a]. As this mediated schema has to be created and refined by domain experts, it can pose a bottleneck for data integration applications [Dong and Srivastava, 2015a].

Within the schema matching task, one can additionally differentiate between the subtasks of finding semantic property matches and creating the actual schema mapping. Semantic property matching refers to the task of finding which properties from different source describe semantically the same knowledge, without explicitly stating how the data between the two is translated [Dong and Srivastava, 2015a]. On the other hand, creating the actual mapping, involves creating query expressions that relate one schema to the other explicitly, allowing for example the reformulation of a user query on the mediated schema into a set of queries on underlying schemata [Dong and Srivastava, 2015a, Doan et al., 2012a]. Works on schema matching however can focus on deriving a mapping that includes only semantic property matches [Madhavan et al., 2001, Bernstein et al., 2011].

Matching approaches can generally be categorized into two types. Schema-based matching exploits only information from the schema of a data source, without making use of content within a data source. It e.g. makes use of linguistic similarities between the names of properties, or exploits information about property data types for matching. On the other hand, instance-based matchers make use of the actual data within sources to match the schemata of the sources. This can include for example linguistic comparison of the data, but also constraint-based comparison using value overlap and value ranges [Rahm and Bernstein, 2001].

One instance-based method is duplicate-based matching [Bilke and Naumann, 2005]. It first requires deriving correspondences between the entities of two data sources, i.e. duplicates, which are then used to better identify semantic property matches. As finding duplicates generally requires identity resolution, which could themselves rely on an existing schema mapping, an iterative approach can be used.

Traditional Identity Resolution Methods

Identity resolution refers to the task, where it is determined which records from different sources match, i.e. describe the same real-world entity [Christen, 2012, Dong

and Srivastava, 2015c]. This is mainly done through pairwise matching, where record pairs are compared individually, and a local matching decision for each pair is made. As comparing every possible pair does not scale for large datasets, blocking methods are used to reduce the number of comparisons. To refine the matching decision and to ensure global consistency, a clustering approach can be used.

Pairwise Matching. Pairwise matching is the basic step of identity resolution. A candidate record pair is first compared to then make a decision whether the pair is a match or a non-match. This is done by comparing the properties of both records, either exactly, or using an approximate comparison that returns a similarity value. In any case, the use of data pre-processing is required to ensure that records from different sources are cleaned and standardized.

To perform the actual matching decision, rule-based or supervised classification approaches can be used. *Rule-based approaches* can be tailored to complex matching scenarios, however, they might require significant domain knowledge to formulate, while being ineffective when records contain errors and noise. For *classification approaches*, machine learning techniques are used along with a set of labeled training pairs of matching and non-matching records to train a binary classifier that decides whether a compared pair matches or not. The classifier can additionally provide a confidence score for its matching decision.

Blocking. Employing pairwise matching directly has a quadratic complexity, as each record from one source needs to be compared with each record from another source. This does not scale when the number of sources is large. Using blocking, the number of comparisons can be reduced by exploiting the fact that the majority of record pairs are likely non-matches. Using a blocking function, the records from each data source are first mapped to blocks. Only record pairs within the same block are then compared using pairwise matching.

Clustering. As pairwise matching produces local matching decision for pairs, the outcome might be globally inconsistent. Let's consider a case with three records a_x, b_y, and c_z from three different sources a, b, and c and a pairwise matching outcome where pair a_x and b_y and pair a_x and c_z are matches, while pair b_y and c_z is a non-match. This is a conflicting outcome, as b_y and c_z themselves are determined to be non-matches, while they both match the same record a_x.

By performing identity resolution using clustering, we can create a globally consistent outcome. Each resulting cluster of records refers to one unique entity, where records in a cluster have a high intra-cluster similarity, and records between clusters have a high inter-cluster dissimilarity. These similarities can e.g. be based on the confidence scores returned by a supervised pairwise matching classifier.

Clustering can be performed as a post-processing step on the output of the pairwise matching, e.g. by building a graph where each edge corresponds to a match, and transforming the graph using clustering into pairwise disjoint subsets. Alternatively, clustering can be performed on pairwise similarities or dissimilarities com-

puted between all pairs. In this case, pairwise classification would be substituted by pairwise similarity computation.

As the number of clusters corresponds to the number of unique entities described by all sources, the number of clusters can not be determined beforehand. As a result, a clustering algorithm is required, that itself determines the optimal number of clusters in the output. A possible candidate for clustering is for example correlation clustering, which is robust, however also computationally expensive.

Traditional Data Fusion Methods

Data fusion methods [Dong and Srivastava, 2015b, Li et al., 2016, Dong et al., 2014b] have become more important with the emergence of data extracted from the Web. Traditional data fusion methods were rule based, i.e. by using *voting* to take the value that is provided by the highest number of sources, by taking the most recent value, by averaging or by taking the minimum or maximum value.

4.2.2 Schema Matching and Identity Resolution for Web Tables

The majority of schema matching and identity resolution methods for web table data work on the same basic principle: a knowledge base, catalog or repository are employed, towards which the schemata of the tables are matched, and towards which the identity of entities described in the tables are resolved.

Limaye et al. [Limaye et al., 2010] use a graphical model to annotate columns and cells in a table with references to a type hierarchy and an entity catalog extracted from YAGO. TabEL [Bhagavatula et al., 2015] is a system that performs relation extraction between columns and entity linking between entity mentions in web tables and a reference knowledge base, in their case also YAGO. Mulwad et al. [Mulwad et al., 2010] link cells to entities in linked data datasets and assign DBpedia classes to web table columns. TableMiner+ [Zhang, 2017] is a system that links cells in tables to entities in Freebase and assigns specific semantic relations from Freebase to pairs of columns. Finally, T2K [Ritze et al., 2015] is a framework that uses an iterative approach to match web tables to DBpedia. It matches web tables to classes and web table columns to properties in the DBpedia ontology, while web table rows are matched to existing entities in DBpedia.

The focus of many works on linking tables to one central catalog or knowledge base can be explained by the fact that the majority of works attempt to understand and semantically interpret tables [Bhagavatula et al., 2015, Mulwad et al., 2010, Zhang, 2017], in lieu of using them for knowledge base augmentation.

This usage of a central knowledge base towards which the schemata of the tables are matched, causes the schema of the knowledge base to take the role of the mediated schema. As such, in the context of knowledge base augmentation, the schema and data model are predetermined by the knowledge base and are themselves not extended using web table data. This is in line with the research in

this thesis. However, schema expansion from web table data has also been studied [Lehmberg and Hassanzadeh, 2018, Lehmberg, 2019].

In the case of identity resolution however, this means, that if a row describes an entity that does not have a corresponding instance in the knowledge base, the row will remain unmatched. While this is an indication that the row likely describes a new long-tail entity that could be added to the knowledge base, we do not have any information about which other rows also describe this same entity. Performing identity resolution by matching rows or cells to entities in the knowledge base, therefore only enable slot filling missing facts for existing entities, but does not allow us to perform entity expansion.

To enable entity expansion, we require an approach to identity resolution that does not only link web table rows to instances of entities in the knowledge base, but also creates correspondences between the rows, that describe entities not present in the knowledge base. We solve this using our Long-Tail Entity Extraction Pipeline described in Chapter 8.

4.2.3 Data Fusion Methods for Web Data

The task of data fusion [Dong and Srivastava, 2015b, Li et al., 2016, Dong et al., 2014b] is to find true value given a set of conflicting values from different sources. Given that web data is noisy, traditional data fusion methods are likely not sufficient. Choosing for example the most common value, i.e. voting, is not useful when the majority of sources provide a wrong value. Similarly, averaging correct with incorrect values does not yield an accurate output.

In this section we will introduce data fusion methods for web data. We will differentiate between fusion methods for static non-time-dependent web data, and time-aware fusion methods for fusing time-dependent web data.

Fusing Static Web Data

To fuse static data, fusion methods must determine from the set of conflicting and noisy values, the value that is correct. Fusion methods for static data generally consist of two concepts: source reliability and truth table. *Source reliability*, also referred to as source accuracy [Dong et al., 2015, Dong et al., 2013] or website trustworthiness [Yin et al., 2008], refers to the reliability of a source in providing accurate data. The reliability of a source in the context of the Web is sometimes contrasted with its popularity based e.g. on the hyperlink structure of the Web [Yin et al., 2008, Dong et al., 2015]. *Truth table* on the other hand refers to the true values or likely true values given certain data items. The truth table is used to estimate source reliability.

The two concepts are related by the following principle: "The sources that provide true information more often will be assigned higher reliability degrees, and the information that is supported by reliable sources will be regarded as truths" [Li et al., 2016]. As the truth table is generally unknown beforehand, truth discovery

is therefore a "chicken-and-egg problem" [Dong and Srivastava, 2015b], which is why many methods for truth discovery are iterative.

Some works [Pasternack and Roth, 2010, Yin and Tan, 2011, Dong et al., 2015] use a ground truth as an a priori truth table to initialize source reliability. For example, the Knowledge-Based Trust approach [Dong et al., 2015], where Freebase is used as a ground truth to bootstrap truth finding and source reliability estimation.

Dong et al. [Dong et al., 2015] also do not only estimate source trustworthiness, but simultaneously also the trustworthiness of extractors. This approach is useful, as conflicting data and noise are not just caused by incorrect data within the source itself, but also by incorrect extraction or possibly incorrect schema matching and identity resolution. In the fusion methods we employ in this thesis (see Subsection 4.3.3), we estimate source reliability using Knowledge-Based Trust per web table column. As web table columns share similar extraction, matching and normalization, estimating the trustworthiness per web table column potentially captures extraction, matching and normalization quality, in addition to source quality.

Fusing Time-Dependent Web Data

The above described fusion approaches assume that data is static [Pal et al., 2012, Dong and Srivastava, 2015b], so that a certain truth does not change over time. However, there also exists time-dependent data, where a value is only valid given a certain temporal scope [Kuzey and Weikum, 2012], i.e. a point in time or a time range. For this type of data, we require fusion methods that are time-aware [Dong et al., 2016].

This section summarizes existing approaches for time-aware fusion. As in this thesis we research data integration methods, we will only consider time-aware approaches that are in the area of data integration. There exist approaches for slot-filling time-dependent data without data integration, e.g. through relation extraction from unstructured text [Surdeanu, 2013].

Dong et al. [Dong et al., 2016] provide an extensive survey on time-aware data integration. They describe as the main challenges of time-aware integration the identification and assignments of temporal scopes to data. Here they find rule-based approaches, e.g. HeidelTime [Strötgen and Gertz, 2010], or approaches that use a combination of combination of machine learning and syntactic analysis [Ling and Weld, 2010]. In both cases, the temporal scopes are extracted from temporal expressions found within the input data. This could for example be in the form of timestamps, which are found in and around a web table [Zhang and Chakrabarti, 2013]. Other works exploit e.g. the crawl date [Dong et al., 2009]. Limiting the source of temporal scopes solely to being explicitly stated in the input is however problematic. The crawl date of a website has likely no direct relationship to the temporal scope of the data and web tables extracted from the website. Timestamps found in and around the table could be more relevant, however they are potentially sparse and noisy [Zhang and Chakrabarti, 2013]. Chapter 6 elaborates more on the issue of sparse and noisy timestamps, and presents approaches to deal with

both. Chapter 7 introduces an approach that propagates temporal scopes from a temporal knowledge base to web table data, thereby eliminating the dependence on explicitly stated temporal scopes.

Time-ware methods also differ in the way they formulate the time-aware fusion problem. Some works consider only finding the most current value from a set of conflicting values [Dong et al., 2009, Alexe et al., 2014, Fan et al., 2014]. In the work by Zhang and Chakrabarti however, a target temporal scope is provided as part of the input query, so that the time-aware method is tasked with finding the value that is valid given that scope [Zhang and Chakrabarti, 2013]. In this research, we formulate the fusion problem similarly, such that a fusion method is aware of the target temporal scope. We outline this further in Chapters 6 and 7.

In regard to their actual approaches, the works also differ strongly. Dong et al. [Dong et al., 2009] estimate source reliability using a probabilistic model and source quality indicators that consider the time-dependence aspect of data. To measure these indicators however, the same source needs to change over time and to be crawled repeatedly, which is not applicable to web table data. Zhang and Chakrabarti [Zhang and Chakrabarti, 2013] employ probabilistic graphical models to match web table columns with each other semantically, so that matched columns also describe data for the same temporal scope. Alexe et al. [Alexe et al., 2014] use a set of class-specific preference rules to determine the most recent value given a set of conflicting alternatives, while Fan et al. [Fan et al., 2014] use class-specific constraints, that are manually refined.

Only Zhang and Chakrabarti [Zhang and Chakrabarti, 2013] actually integrate data from web tables, which they use to perform attribute expansion. Given a set of input entities and a target property, they extract from a web table corpus corresponding values for that property. For time-dependent data, the target property is given a target temporal scope as part of the input query.

In summary, Zhang and Chakrabarti provide the only viable approach that enables time-aware fusion of web table data. Other approaches additionally either only enable finding the most recent value [Dong et al., 2009, Alexe et al., 2014, Fan et al., 2014], require domain-specific supervision [Alexe et al., 2014, Fan et al., 2014], or require a continuously crawled source that regularly updates [Dong et al., 2009]. We provide an extensive comparison to the related work in Chapters 6 and 7, with focus on the work by Zhang and Chakrabarti.

4.3 Data Integration Frameworks

This thesis focuses on investigating two augmentation tasks: slot filling for time-dependent data, and entity expansion. For our research, we exploit a set of web data integration frameworks, with methods based on the current state of the art. This section introduces these frameworks.

First, we make use of the T2K matching framework. We use it throughout all methods presented in this thesis for performing column data type detection, entity

label column detection, and table-to-class matching. Additionally, we use it for schema matching and identity resolution in Chapters 5 to 7. In Chapter 5, we use it to profile a web table corpus, while in Chapters 6 and 7, we use it in the context of researching and evaluating time-aware fusion methods. However, in regard to our research on entity expansion, we require novel identity resolution methods. For this purpose, we do not make use of T2K for schema matching and identity resolution and introduce and evaluate an alternative approach in Chapter 8.

Secondly, we introduce a framework for data fusion that we use throughout this thesis. This framework can make use of any fusion strategy by allowing it to score candidate values given a certain target triple. Our fusion framework also allows the use of time-aware fusion strategies, as it provides a target temporal scope in addition to the target triple. This section also introduces a set of static fusion strategies that we use throughout this thesis.

Moreover, we describe the Local Closed-World Assumption (LCWA) [Dong et al., 2015], which we use in Chapters 5, 6, and 7 to evaluate slot filling performance. For entity expansion we do not make use of this assumption, as we evaluate using a gold standard, which we introduce in Chapter 8.

Finally, we use throughout this thesis a set of data types, and corresponding data-type-specific similarity and equivalence functions. We will introduce these at the end of this section.

4.3.1 T2K Matching Framework

The T2K matching framework [Ritze et al., 2015] has a broad range of functionality when it comes to matching web tables to a knowledge base. This includes column data type detection, entity label column detection, table-to-class matching, schema matching, and identity resolution in the form of row-to-instance matching.

Column Data Type Detection

T2K performs column data type detection by assigning to each table column one of the following basic types: string, date and numeric. Before detecting the data type, it pre-processes tables by cleaning them, e.g. from HTML characters, but also by normalizing their values. The data type detection is performed using about 100 manually defined regular expressions. The data type of a column is chosen based on the majority data type among its values. If necessary, the values of a column are normalized based on the result of the data type detection.

Entity Label Column Detection

The entity label column contains natural language labels for the entities described in the table. It acts as the key of the table and is necessary in determining the main topical content of a table. T2K considers for the entity label column, only the columns with the data type string, and chooses the one with the highest number of

unique values. In case there is a tie between multiple columns, T2K chooses the column that is furthest to the left.

Table-To-Class, Schema and Entity Matching

T2K performs table-to-class, attribute-to-property and row-to-instance matching simultaneously. It first extracts from the entity label column a label for each row and uses the label to find candidate entity instances from the knowledge base. A class, for which many rows of a table have a candidate instance, is chosen as a possible candidate class of that table.

T2K then performs value-based row-to-instance matching, scoring candidate instances of a row based on the overlap of the values in the row, with the values of the instance in the knowledge base. Only values of equal data type are compared.

Given a table's candidate classes, T2K performs for each class attribute-to-property matching, i.e. matching the table's columns, besides the entity label column, to properties in the knowledge base schema. T2K first compares for each property the values in the rows with facts of that property in the knowledge base. This is done only for facts that are part of the candidate instances of a row, and only for the properties from the knowledge base where the data types of the column and the property match. These comparisons yield value-based similarities, which are used by T2K to perform duplicate-based attribute-to-property matching [Bilke and Naumann, 2005], additionally scoring the property candidates.

For each table, a score is computed for each of its candidate classes by aggregating the instance and property matching scores for each class. We then choose the class with the highest score and assign it to the table, which gives us table-to-class correspondences.

Row-to-instance and attribute-to-property correspondences of the chosen class are however further refined using an iterative approach. This is done by recomputing the scores of candidate instances by weighting properties using their preliminary assigned matching scores and recomputing the scores of candidate properties by weighting instances using their preliminary assigned scores. This is repeated until scores converge, so that final sets of attribute-to-property and row-to-instance correspondences are chosen.

The authors evaluated T2K using a gold standard with correspondences to DBpedia.[1] It achieved an F1 score of 0.82 for row-to-instance, 0.70 for attribute-to-property and 0.94 for table-to-class matching [Ritze et al., 2015].

4.3.2 Data Fusion Framework

All fusion strategies used in this thesis are based on one underlying fusion framework. Within this framework, we assume as input a set of web table values matched to a single triple, i.e. a combination of a certain entity and a certain property. In

[1] http://webdatacommons.org/webtables/goldstandard.html

this context, fusion should return from all candidates the value that is correct. In the case of time-dependent data, time-aware fusion strategies must find the value that is also valid given a certain target temporal scope.

The fusion framework consists of four steps, which we describe below. Within our data fusion methodology, fusion strategies influence the outcome of fusion solely through scoring.

1. **Scoring.** A fusion strategy computes a score for each candidate web tables value. In case of time-dependent data, fusion strategies are also given a target temporal scope as input for scoring. Scores are provided for a range from 0.0 to 1.0.

2. **Filtering.** Based on their assigned scores, values are filtered using a learned threshold. The filtering is an important step as it strongly influences a possible precision/recall trade-off. A filter with a higher threshold could increase precision at a possible cost of recall and vice-versa. To address this trade-off, we optimize our thresholds, usually for the $F1$ score. Thresholds are learned per fusion strategy and per property-class.

3. **Grouping.** For a given target slot, i.e. a triple, of the knowledge base, we group all matched values that are *equal* into one group. To determine if values are equal, we employ the data-type-specific equivalence functions we describe further below. We sum the assigned scores of all values in a group to calculate a group score.

4. **Selection.** We select the group with the highest summed score and determine a value that represents that group. As values in a group can deviate slightly, while still determined as equal by an equivalence function, we use *data-type-specific fusers* to determine one value that represents a group. For string and reference values, we use the value that has the highest sum of scores within the group. For date and numeric values, we use the weighted median, where values are weighted by the score assigned to them by the fusion strategy. For nominal string and integer values, all values in a group are exactly equal, as the equivalence functions do not allow for deviations. Data types are described in more details in Subsection 4.3.5 below.

4.3.3 Static Fusion Strategies

This section introduces a set of fusion strategies for static data. Among them, are simple baseline strategies like voting, and strategies that make use of source reliability estimation, like Knowledge-Based Trust.

Voting. Voting is a common baseline fusion strategy [Dong et al., 2014b]. It assigns all values matched to a triple a score of 1.0 and is therefore effectively a simple count of the number of web tables from which a specific candidate value

was extracted. As all candidate values have the same score, filtering with a threshold, i.e. step 2 of our fusion framework, is not applicable.

PageRank. PageRank [Brin and Page, 1998] uses the link structure of the Web to rank websites. It is widely used to assess the quality of Web content and has also previously been used for data fusion [Pasternack and Roth, 2010]. Therefore, it can also be used to generate scores for the fusion of web table values. To implement a fusion strategy based on PageRank, we use a full ranking of all hosts within the 2014 Common Crawl provided by the WDC Hyperlink Graph[2] [Meusel et al., 2015]. The score of a candidate value, is equal to the PageRank score of the host from which the value was extracted.

Knowledge-Based Trust. Knowledge-Based Trust [Dong et al., 2015] is a fusion strategy that makes use of a ground truth to estimate the trustworthiness of sources. It is based on the assumption that neighboring values share similar correctness, so that a value extracted from a source can be considered trustworthy, if other values extracted from that same source are correct. To determine if values extracted from a source are correct, an external ground truth can be used. This ground truth can for example be in the form of the knowledge base that we also aim to extend.

In their original work, the authors compute trust scores per source and extractor, as the trustworthiness of an extracted value depends on both, the quality of the original source and the accuracy of the extractor. We employ Knowledge-Based Trust by computing scores per web table column. This is done, because we believe that a column possibly captures source, extraction, normalization and matching quality. The trust score of a column is computed using the following formula.

$$\text{KBT(column)} = \frac{\text{\# of correct overlapping values}}{\text{\# of overlapping values}} \tag{4.1}$$

Overlapping values are those for which we can find an existing triple in the knowledge base, i.e., where the column is matched to a property of the knowledge base schema, and the row to an instance of an existing entity in the knowledge base, that contains a value for the property. An overlapping value is counted as correct, if it is equal to the value in the knowledge base triple it was matched to. This is determined using the data-type-specific equivalence function we describe below.

During fusion, the KBT score of a column is then assigned to the candidate values extracted from that column as part of step 1 of the fusion framework. We use this Knowledge-Based Trust approach as the primary fusion strategy for static data throughout this thesis.

4.3.4 Evaluating Fusion Using the Local Closed-World Assumption

As suggested by Dong et al. [Dong et al., 2014a], we use the Local Closed-World Assumption (LCWA) to enable the large-scale automatic evaluation of fusion strate-

[2] http://webdatacommons.org/hyperlinkgraph/

gies. According to the LCWA, if for a given triple there exist one or more values in the knowledge base, we assume that the correct value is among them. If a value returned by a fusion strategy is among the existing values in the knowledge base, it is assumed to be correct, otherwise it is assumed to be incorrect. On the other hand, if for a triple, there exist no values in the knowledge base, we exclude the triple from evaluation completely, as we can neither say if it is correct nor incorrect. Dong et al. show that this assumption is a valid approximation. We manually test and empirically confirm the LCWA in Section 5.4.1 of the next chapter.

To compare the values fused from web table data with the values in the knowledge base, we use the data-type-specific equivalence functions that we describe below. As we cannot expect data from web tables to be perfectly clean, the equivalence functions allow for minor deviations when comparing fused triples to values of overlapping triples from the knowledge base.

Using the LCWA, we can estimate the precision of a fusion strategy using the following formula:

$$Precision = \frac{fused_{correct}}{fused_{total}} \tag{4.2}$$

However, we are also interested in estimating the recall of a fusion strategy. To do this, we must find the maximum number of triples that exist in the knowledge base for which a correct triple could potentially be fused from the web table corpus. We determine this number by comparing for each triple its values from the knowledge base with each individual value matched to it from web table data using the equivalence functions described value. If an equal value exists among its matches, we count the triple to the maximum number of triples for which a correct value can be fused. Recall is then estimated using this formula:

$$Recall = \frac{fused_{correct}}{correct_{max}} \tag{4.3}$$

In the formula, $correct_{max}$ is the number of triples in the knowledge base, for which a correct value exists among all the matched candidate web table values.

Finally, we use the harmonic of both precision and recall, the F1 score, to measure fusion performance:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.4}$$

In Chapter 8, where we introduce methods for entity expansion, we do not evaluate fusion performance using the LCWA, but using a gold standard. This is because we are evaluating performance for long-tail entities, which by definition do not exist in the knowledge base. Additionally, evaluating performance using a gold standard is likely more accurate, especially in regard to recall.

4.3.5 Data Types and Similarity and Equivalence Functions

Throughout methods in this thesis, we employ data types. The data types are used to type individual values, facts, web table columns or knowledge base properties. Each data type has a corresponding similarity function, which compares values and produce similarity scores normalized from 0.0 to 1.0. We additionally make use of equivalence thresholds on the similarity functions. This gives us equivalence functions, which we use to determine if two values are equal.

Overall, we consider six different data types, however not all are used throughout the thesis. T2K itself supports column data type detection for string, numeric and date. Furthermore, in Chapters 5 to 7, we make use of the entity reference data type. In Chapter 8, we additionally make use of nominal string and integer types. The six types, including their similarity and equivalence functions are:

- **String:** a string value, where two strings do not have to be exactly equal to be similar. Given for example two entity labels 'Frankfurt am Main' and 'Francfort-sur-le-Main', these labels are not equal, however still very similar, and they should be scored as such. The similarity metric for this data type first normalizes the two input strings, which includes the removal of special characters, conversion to lowercase, and conversion accented characters to non-accented characters. Afterwards, the two input strings are tokenized, and compared using the Generalized Jaccard similarity metric, using Levenshtein as the inner similarity [Doan et al., 2012b]. For this function we learn two thresholds, one for the inner function, to determine if two tokens are equal, and one for the outer, to determine if the two values are equal. The thresholds are learned, using a manually annotated external dataset.
- **Numeric:** a numeric quantity, where numeric closeness has a semantic relevance, e.g. population of a settlement, or height of an athlete. The similarity between numbers equals their deviation subtracted from one. The deviation equals the ratio of the absolute distance between the two numbers relative to the larger magnitude of either number. The numbers are considered equal if their deviation is smaller than 5% (Chapter 5 to 7) or 3% (from Chapter 8).
- **Date:** point in time with three possible granularities: year, year-month or a specific date. This data type is used for example for the release date of a song or the birth date of a person. When comparing dates, they must perfectly match, in which case they achieve a similarity of 1.0, otherwise 0.0. However, they are always compared at the lowest available precision. E.g. a date value with just the year equals a date specific to the day, if the years are equal in both dates.
- **Entity Reference:** references to other entities, whether those exist in the knowledge base or not, e.g. team of an athlete or musical artist of a song. As values with type entity reference are simple string literals in web table data, in Chapter 5 to 7 we match these literals to actual instances of exiting

entities in the knowledge base before comparing them. They are then compared using an exact similarity. However, this approach was error prone, as the disambiguation was not always successful, and, it assumed that all referenced entities are covered in the knowledge base. From Chapter 8, we do not match literal values to existing entities in the knowledge base, but instead use the literals extracted from web tables in our similarity and equivalence functions. We use overlap similarity as the similarity function, with Levenshtein as the inner similarity. If one of the compared values is already disambiguated (i.e. the value is from the knowledge base, and for example in the form of an URI), we utilize the labels and synonyms of its instance present in the knowledge base, and choose the highest achieved similarity score. The equivalence threshold is set at 1.0.

- **Nominal String:** a literal string, where the two strings are only similar if they are exactly equal. e.g. postal code of a settlement[3], or IATA code of an airport. The similarity function used for this data type is an exact similarity, where it is only checked if the two values are exactly equal or not.
- **Nominal Integer:** an integer data type, where numbers close to each other are not semantically related. This include for example numbers of athlete, where two numbers are close, have the same semantic relevance to each other as two numbers that are numerically distant, i.e. none. The similarity function used for this data type is also an exact similarity measure.

4.4 Summary

Knowledge base augmentation from web tables is a data integration task. This chapter summarized existing methods for integrating web tables with a knowledge base and investigated how far these methods enable knowledge base augmentation.

We showed that existing methods for identity resolution focus on slot filling and are insufficient for performing entity expansion. We also found that there are few viable time-aware fusion methods for slot filling time-dependent data.

This chapter also introduced a set of frameworks containing data integration methods from the current state of the art. It includes web table schema matching methods in the form of the T2K Framework, static data fusion method, e.g. Knowledge-Based Trust, an evaluation approach for slot filling performance using the Local Closed-World Assumption, and data-type-specific similarity and equivalence functions for value comparison.

We use these frameworks throughout this thesis. This includes e.g. the next chapter, where we use them to profile the information described by web tables and assess the potential of web tables for the task of slot filling.

[3]While postal codes in Germany are numeric, they are not in all countries, e.g. in the UK, where they are alphanumeric.

Chapter 5

Web Table Profiling

Existing works on using web tables for augmenting knowledge bases [Limaye et al., 2010, Zhang et al., 2013, Dong et al., 2014a] are evaluated on either small and thus not representative web table corpora or on large web table corpora owned by search engine companies, which do not allow information about the content and coverage of their crawls to be published. Further, none of the existing publications answers the question of which topical areas of the knowledge bases can be complemented using web table data. For these reasons, we believe that a large publicly available corpus, such as the 2012 WDC web table corpus[1], along with an in-depth profiling of its contents can provide significant insights into the topical domains covered by web tables and their potential for knowledge base augmentation.

This chapter reports about the results of matching 33 million web tables from the 2012 WDC web table corpus to DBpedia. Based on the matching results, we profile the potential of web tables for augmenting different parts of the knowledge base and report detailed statistics about correspondences between web tables and classes, properties, and entities of the knowledge base. In order to explore the degree of overlap between the web tables, we group values extracted from web tables by their matched triple. We are thus able to report the size distribution of the resulting groups with alternative values extracted from different tables. Finally, we estimate fusion performance and profile the potential for slot filling new facts in the knowledge base using web tables.

The contributions of this chapter are:

- An in-depth profiling of a publicly available web table corpus, providing insights into its topical contents.
- A confirmation that the Local Closed-World Assumption is valid and that its results are transferable to new facts fused from web table data.
- A verification that Knowledge-Based Trust outperforms PageRank- and voting-based data fusion strategies.

[1] http://webdatacommons.org/webtables/#toc3

This chapter is organized as following. First, we briefly introduce our experimental setup. Section 5.2 then discusses statistics about class, property, and entity correspondences created by web table matching, while Section 5.3 analyzes the internal overlap of web table data using triple groups. Section 5.4 verifies the Local Closed-World Assumption, compares various data fusion strategies, and provides statistics about fused triples and their slot filling potential. Section 5.5 discusses related work, while the final section summarizes this chapter.

The work presented in this chapter has previously been published in [Ritze et al., 2016]. It is a joint contribution, as the research has been conducted together with Dominique Ritze [Ritze, 2017] and Oliver Lehmberg [Lehmberg, 2019]. Our primary contribution within this research is in regard to fusion, i.e. Section 5.4.

5.1 Experimental Setup

This section outlines the experimental setup of this chapter. We first introduce and provide statistics about the 2012 WDC web table corpus. We then briefly introduce the reference knowledge base, in our case DBpedia, and quickly introduce the matching methodology used to match web tables to DBpedia.

5.1.1 Web Table Corpus

We profile the 2012 WDC web table corpus, which we have previously introduced in Section 3.3. We only consider web tables from the English-language subset, i.e. from the top-level domains (TLDs) *com*, *org*, *net*, *eu*, and *uk*. As shown in Table 5.1, the majority (83%) of these tables originate from *com*-domains.

We further exclude all tables without an entity label column, which we define as the column that contains the names of the entities described within a table. Without such a column, we cannot determine the topical content of a table. We also exclude very small tables with less than five rows or three columns. The resulting subset of the corpus consists of 33,403,411 tables.

The entity label column is detected using T2K (see Section 4.3), along with the data types of the other columns. This assigns each column one of the three data types string, numeric or date. Table 5.2 shows statistics about columns, rows and values in general and per data type, excluding columns without obvious data types. We find that most values are of data type string, followed by numeric.

The tables in our corpus originate from 97,932 different websites. Here, we use the term website for each pay-level-domain (PLD), that is, the part of an URL's host that is paid for. Table 5.3 shows the most frequent PLDs. The most prominent PLD is *apple.com* (iTunes Music), while the other PLDs often refer to sport websites, such as *baseball-reference.com*, or retailers, such as *amazon.com*.

Table 5.4 shows the most frequent column headers. These are derived from the first non-empty row of a table. The headers give us a first impression about the topics within the web tables. Frequently used headers are for example "5 star" and

"price", indicating that the corpus contains a large number of tables about products. Headers like "replies" or "latest post" point to the fact that the corpus contains data from blogs or forums. About 8.5% of all columns have an empty header.

Table 5.1: Number of tables per top-level domain (TLD) in the profiled corpus.

TLD	**.com**	**.org**	**.net**	**.eu**	**.uk**	**Σ**
Tables	26.7m	3m	3m	216k	6k	**33.4m**

Table 5.2: Number of columns, rows and values in the profiled corpus.

			Per data type		
	Σ	**μ per table**	**Numeric**	**Date**	**String**
Columns	137m	4.1	46m	4m	86m
Rows	716.6m	21.5	-	-	-
Values	2.95b	88.6	995m	101m	1.9b

Table 5.3: Most frequent pay-level-domains (PLDs) in the profiled corpus.

PLDs	**Tables**
apple.com	50,910
patrickoborn.com	455,005
baseballreference.com	25,647
latestf1news.com	17,726
nascar.com	17,465
amazon.com	16,551
baseballprospectus.com	16,244
wikipedia.org	13,993
inkjetsuperstore.com	12,282
flightmemory.com	8,044
sportfanatic.net	7,596
tennisguru.net	7,504
windshieldguy.com	7,305
donbergelectronique.com	6,734
citytowninfo.com	6,293
juggle.com	5,752
deadline.com	5,274
blogspot.com	4,762
7digital.com	4,462
electronicspare-parts.com	4,421

Table 5.4: Most frequent column headers in the profiled corpus.

Headers	**Tables**
no header	14,495,456
5 star:	2,402,376
name	1,813,064
price	1,771,361
date	1,603,938
amazon price	1,178,559
formats	1,066,836
title	9,132,60
time	856,401
description	773,883
size	692,251
replies	605,075
used from	589,278
new from	589,259
year	579,726
location	546,856
album	526,375
type	501,747
latest post	421,737
discussion	412,672

5.1.2 Reference Knowledge Base

As the reference knowledge base, we use the 2014 release of DBpedia. We have described DBpedia and specifically profiled the 2014 release in Section 2.4. It contains 4.58 million entities, describing them with 153 million facts using more than 2600 different properties and 680 classes. Table 2.1 profiles a selection of classes with their number of entities and facts from the DBpedia hierarchy.

5.1.3 Matching Methodology

To profile the content of web tables, we must first match each individual table to the reference knowledge base. This includes finding correspondences between web tables and classes of the knowledge base, web table columns and knowledge base properties, and, finally, web table rows and entities within the knowledge base. To match the tables from the web table corpus to our reference knowledge base, we make use of the T2K framework, which we have already described in Section 4.3.

5.2 Matching and Correspondences Profiling

This section profiles and provides statistics on the correspondences returned by the matching. Table 5.5 summarizes these statistics by a selection of DBpedia classes. T_0 is the overall number of tables matched to the class, while T_c is the number of tables which have at least one property correspondence besides the entity label column. V_c is the number of cells (values) contained in the tables T_c that have both an entity and a property correspondence. These numbers are further divided according to their data type in the last four columns of the table.

Matched Tables. Altogether, 949,970 of 33.3 million web tables have correspondences to DBpedia entities (T_0), distributed over 361 different classes from the DBpedia ontology. Such tables describe entities that already exist in DBpedia, but potentially also describe previously unknown entities, making them useful for entity expansion. 301,450 tables (T_c) additionally have at least one property correspondence. They match altogether 274 different DBpedia classes. In these tables, 8 million values (V_c) were successfully matched to a triple, i.e. a combination of a property and an entity. If this triple is missing in DBpedia, these values could potentially be used for slot filling. The fact that only 2.85% of all web tables can be matched to DBpedia indicates that the topical overlap between the tables and DBpedia is rather low. This is also in line with the frequent column headers as shown in Table 5.4, as the most frequent headers are centered around product offerings, which is knowledge not typically covered in DBpedia.

Class Overlap. To profile the topical overlap between the web tables and DBpedia, we provide statistics about the most frequently matched DBpedia classes from the higher levels of the DBpedia class hierarchy in Table 5.5. Almost 50% of the web

Table 5.5: Correspondence statistics for matched tables.

DBpedia Class	Number of Tables/Values			V_c Data Type			
	T_0	T_c	V_c	**Numeric**	**Date**	**String**	**Reference**
+ Person	265,685	103,801	4,176,370	2,117,793	1,588,475	266,628	203,474
| − Athlete	243,322	95,916	3,861,641	2,084,017	1,435,775	163,771	178,078
| − Artist	9,981	2,356	18,886	3	11,527	3,499	3,857
| − Politician	3,701	1,388	18,505	10	7,725	3,393	7,377
| − Office Holder	2,178	1,435	131,633	30	66,762	59,332	5,509
+ Organisation	194,317	36,402	573,633	99,714	187,370	100,710	185,839
| − Company	97,891	6,943	203,899	58,621	83,001	34,665	27,612
| − SportsTeam	50,043	2,722	31,866	2,206	22,368	43	7,249
| − Educational Institution	25,737	14,415	238,365	38,056	64,578	13,334	122,397
| − Broadcaster	14,515	11,315	93,042	564	13,095	52,186	27,197
Work	269,570	127,677	2,284,916	109,265	1,354,923	33,091	787,637
+ MusicalWork	138,676	80,880	1,131,167	64,545	396,940	7,610	662,072
+ Film	43,163	9,725	256,425	10,844	198,913	14,382	32,286
+ Software	39,382	23,829	486,868	418	414,092	9,194	63,164
Place	133,141	24,341	859,995	413,375	273,510	84,111	88,999
+ PopulatedPlace	119,361	21,486	787,854	405,406	257,780	57,064	67,604
| − Country	36,009	6,556	208,886	93,107	66,492	31,793	17,494
| − Settlement	17,388	2,672	17,585	4,492	6,662	2,444	3,987
| − Region	12,109	427	5,625	3,097	897	292	1,339
+ ArchitecturalStructure	10,136	1,815	46,067	3,976	7,387	23,110	11,594
+ NaturalPlace	1,704	254	2,568	866	696	340	666
Species	14,247	4,893	83,359	-	7,902	38,682	36,775
Σ	949,970	301,450	8,037,562	2,751,105	3,437,420	536,526	1,312,511

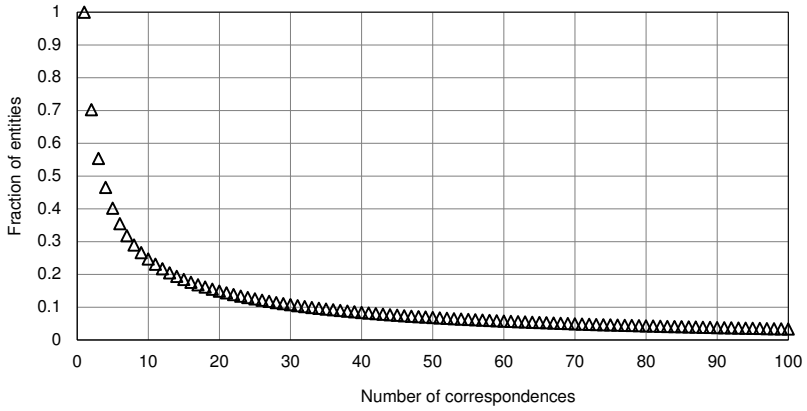

Figure 5.1: Complementary cumulative distribution of the number of correspondences per entity.

tables describe *Person* and *Organisation* entities, followed by tables covering *Work*. It is no surprise that we find a majority of correspondences for these classes, as they are also the three most frequent classes in DBpedia (see Table 2.1 in Section 2.4). More interesting is the fact that the second most frequent class in DBpedia, *Place*, is much less frequently matched in the web tables, although it is twice as large as *Work*. This either indicates that places are underrepresented in the web table corpus or that the matching framework has trouble detecting this class. We can further see that only 18% of the tables about places have a property correspondence. Thus, besides being underrepresented, we also find signs for a schema mismatch between DBpedia and the web tables for *Place* and its subclasses.

Data Types Distribution. In the full corpus, the majority of values had the data type string, followed by numeric. Among the matched tables however, date is the data type with the most values, followed by numeric and reference[2], while string has the least values. The change in distribution of data types could indicate that the actual schema overlap between the tables and DBpedia consists of properties with these data types. In DBpedia, the data types among triples are approximately distributed as following: 40% for reference, 39% for numeric, 15% for string, and 7% for date. The change in distribution for the string data type between all and the matched values could be explained by the smaller prevalence of string triples in DBpedia. However, the opposite seems to be true for date, where there is a disproportionately high overlap in the schemata of web tables and DBpedia.

Row-To-Instance Correspondence Distribution. In total, we find 13,726,582 row-to-instance correspondences for 717,174 unique entities, which is 15.6% of all entities in DBpedia. Figure 5.1 shows the complementary cumulative distri-

[2]As the reference type requires a matching step, these values appear as string in the statistics about the full corpus in Table 5.2.

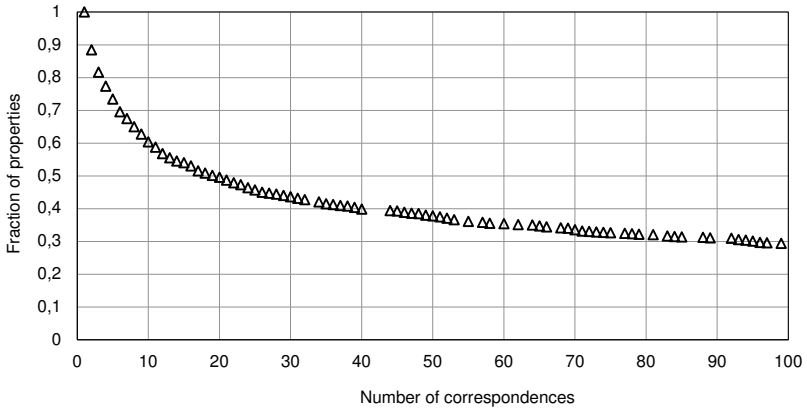

Figure 5.2: Complementary cumulative distribution of the number of correspondences per property.

Table 5.6: Entities and properties with the highest number of correspondences.

Class	Entity	#	Property	#
Athlete	Jeff Gordon	15,826	team	7,982
	Fernando Alonso	14,870	championships	4,464
Country	China	13,515	capital	965
	France	13,300	currency	508
Office-Holder	John McCain	329	religion	74
	Barack Obama	328	vicePresident	66
Company	Toshiba	59,112	formationDate	1,016
	Nortel	45,573	iataAirlineCode	714
Musical-Work	Can't Help Falling in Love	1,403	releaseDate	60,473
	Hold It Against Me	1,801	musicalBand	27,832
Educational-Institution	University of Phoenix	2,486	state	998
	Purdue University	2,325	numberOfStudents	707
Species	Great Egret	541	genus	3,706
	Rainbow trout	329	sire	207

bution function (or tail distribution) of the fraction of entities (y-axis) that have correspondences in a given number of web tables (x-axis). We can see that 70% of all entities with a correspondence can be found in more than one web table. 55% have three or more sources, while 25% have at least ten sources. 3% of entities are described within more than 100 tables.

Table 5.6 lists some examples of frequently matched entities (and properties) for selected classes. All of the frequently matched entities are more or less com-

monly known, which is in line with the intuitive expectation that more popular entities are likely to be described more often in web tables.

Attribute-To-Property Correspondence Distribution. Aggregated over all tables, we find a total of 562,445 column attribute-to-property correspondences for 721 unique properties of the knowledge base. Figure 5.2 shows the tail distribution of the fraction of properties (y-axis) that have correspondences in a given number of web tables (x-axis). 88% of all properties have correspondences from at least two web tables. 81% can be found in three or more web tables and 60% of all properties have correspondences from at least ten web tables. About 30% of all properties have more than 100 correspondences.

5.3 Triple Groups Profiling

Through the row-to-instance and attribute-to-property correspondences, extracted web table values (V_c) are matched to a specific triple, i.e. a combination of an entity and a property. Based on this, we group values matched to the same triple together. E.g., all values from rows mapped to <dbr:Germany> and columns mapped to <dbo:populationTotal> are grouped. The remainder of this section profiles these groups and provides insights into the internal overlap in the web table corpus.

Group Size Distribution. Out of the 8 million values (V_c) extracted from web tables, 929,170 triple groups can be formed. Figure 5.3 shows the tail distribution of group sizes. 58% of all groups contain values from at least two sources, 39% from at least three sources. Values from ten or more sources can be found in 13% of all groups. Very large groups with values from at least 100 sources constitute 1% of all groups. Triples described by less sources are more likely to be new to the knowledge base, as we expect frequently described, i.e. popular, triples to already exist in the knowledge base. But these new triples come with a drawback: as they are only supported by few sources, it will be difficult for a fusion strategy to find the correct values in the groups. For 42% of groups only a single value is present, meaning that a fusion strategy cannot 'fuse' values, but only determine if it wants to accept or discard a single value.

Classes. Table 5.7 shows the number of groups (G) by class. The table also states the ratio of the number of groups to the total number of extracted values. This ratio is high if we cannot group many values per triple, i.e. most groups are small and contain fewer values. If the ratio is low, this means that more values are grouped together, i.e. more groups are larger in size. The ratio is potentially lower for classes with less entities. An example is the class *Country*, which has the lowest ratio. Given that the number of countries is limited, so is the number of potential triple groups, causing the groups to be relatively large. We find generally that this ratio differs per individual class of the knowledge base.

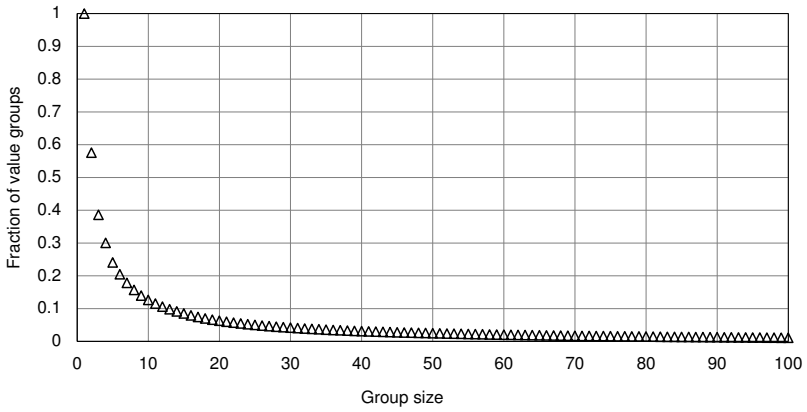

Figure 5.3: Complementary cumulative distribution of group sizes.

Table 5.7: Triple group statistics by DBpedia class.

Class	**G**	$^{G}/_{V_c}$
+ Person	366,048	.088
| − Athlete	284,213	.074
| − Artist	6,842	.362
| − OfficeHolder	6,559	.354
| − Politician	11,362	.086
+ Organisation	87,527	.153
| − Company	25,164	.123
| − SportsTeam	2,453	.077
| − EducationalInstitution	35,736	.150
| − Broadcaster	21,687	.233
Work	331,071	.145
+ MusicalWork	201,186	.178
+ Film	56,610	.221
+ Software	33,552	.069
Place	100,673	.117
+ PopulatedPlace	71,981	.091
| − Country	5,709	.027
| − Settlement	1,879	.107
| − Region	1,193	.212
+ ArchitecturalStructure	17,697	.384
+ NaturalPlace	12,037	.468
Species	23,809	.286
Σ	**929,170**	**.012**

Table 5.8: Data type distribution for different stages of web tables integration.

Data Type	Corpus	Matched	Grouped	$^{M}/_{G}$
Numeric	995m	2,751,105	202,362	13.59
Date	101m	3,437,420	379,240	9.06
String	1.90b	536,526	86,330	6.21
Reference	-	1,312,511	261,238	5.02
Σ	**2.95b**	**8,037,562**	**929,170**	**8.65**

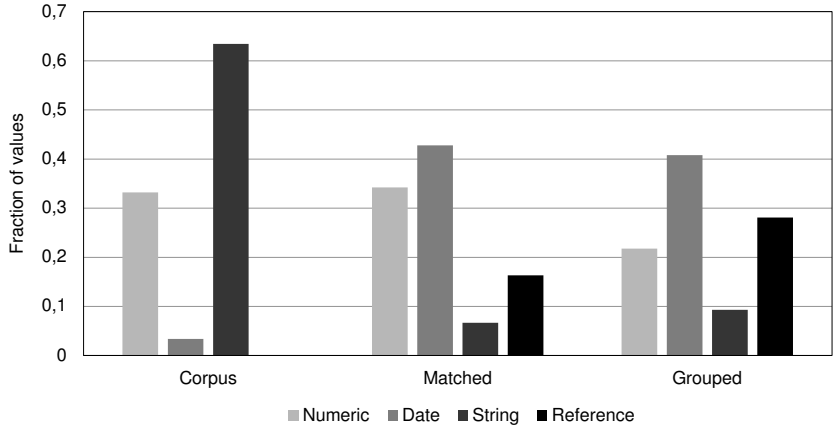

Figure 5.4: Data type distribution for different stages of web tables integration.

Data Type Distribution. Table 5.8 shows the data type distribution at different stages of the data integration process. At first, we have the full web table corpus (Corpus). We then derive correspondences through matching (Matched) and finally group matched values by triple (Grouped). As we already discussed the change in the distribution between the full corpus and the correspondences in Section 5.2 above, we now focus on the transition from correspondences to groups. The last column in Table 5.8 shows the ratio of this grouping process. We see that on average, each group contains 8.65 values. The largest group sizes can be observed for numeric triples, where on average 13.59 values form a group. Date groups are also relatively large with about 9 values. String and reference groups are smaller with only about 5 to 6.2 values. This indicates that more tables describe the same numeric and date triples, whereas in the case of string and reference, there is a stronger diversity among the described triples. Figure 5.4 shows the proportion of data type in each step. Here it becomes obvious how the large fraction of string values in the complete corpus is overtaken by date and numeric in the correspondences. In the grouped stage, we see how the relative size of string and reference increases again, as many date and numeric values are grouped together.

5.4 Data Fusion and Slot Filling

This section profiles the potential of slot filling DBpedia from web table data. This is done by primarily investigating the quality of new triples that can be fused from web table data. For this, we use the evaluation methodology for slot filling described in Section 4.3. In summary, we use the Local Closed-World Assumption (LCWA) to evaluate the correctness of fused triples by comparing them to triples which already exist in the knowledge base, i.e. overlapping triples. Additionally, we test and confirm in this section this assumption through manual evaluation.

To fuse triples, we make use of a fusion framework and a fusion strategy based on Knowledge-Based Trust (KBT), both also described in Section 4.3. We additionally compare fusion performance when using KBT to using PageRank and a voting-based baseline approach. As KBT also uses the knowledge base for learning, we use 5-fold cross-validation in our experiments.

5.4.1 Fusion Performance Evaluation

We evaluate the performance of a fusion strategy using precision and recall. To compute precision, we must determine the fraction of fused triples returned by a fusion strategy that are correct. Based on the LCWA, we can determine if a fused triple is correct by comparing it to a corresponding existing triple in the knowledge base. However, we find that for only 691,622 of the 929,170 groups a corresponding triple exists within DBpedia, i.e. these groups overlap. We run and evaluate our fusion strategies for these overlapping groups only.

To compute the recall, we must determine the upper bound of a fusion strategy. This upper bound equals the number of groups, for which a perfect strategy should be able fuse the correct value. We estimate this upper bound by checking which groups that correspond to an existing triple contain a value that equals the existing value in the knowledge base. This is determined using the data-type-specific equivalence functions (see Subsection 4.3.5). We find that the number of groups containing the correct value is equal to 310,284.

This means that only 45% of groups with an existing triple in the knowledge base actually contain at least one correct value within them. While this seems quite low, to match a value to a knowledge base triple we need to perform class, property and entity matching and, in the case of a reference data type, also string-to-instance matching. Along these matching processes, errors are likely to compound. Additionally, web table values might simply be wrong or outdated.

Fusion Strategy Performance Comparison

In this section, we report the performance of three fusion strategies: Knowledge-Based Trust (KBT), PageRank and voting. This is done to highlight the effect of fusion strategies and examine the claim by Dong et al. [Dong et al., 2015] that KBT outperforms a strategy using PageRank.

Table 5.9: Number of overlapping and non-overlapping triples and fusion performance by fusion strategy.

Strategy	F_O	F_{NO}	P	R	F1
Voting	691,622	237,548	.369	.823	.509
KBT	378,892	64,237	.639	.785	.705
PageRank	691,622	237,548	.365	.814	.504

Table 5.9 shows the number of overlapping (F_O) and non-overlapping fused triples (F_{NO}) that are generated by the different fusion strategies. Note that the non-overlapping triples are the previously unknown triples which could be added to the knowledge base. The last three columns present the performance in terms of precision, recall and F1. The performance values are calculated using the LCWA.

Voting does not apply any filtering. Hence, the precision can maximally be 45%. This is because out of the 691,622 triples groups included in our evaluation, only 310,284 actually contain a correct value). Taking this into account, the achieved precision of 36.9% is at an acceptable level for a simple approach.

KBT, filters out attribute-to-property correspondences with a low trust score and can hence decide not to produce a triple from a given group. This results in a large 27 percentage point increase in precision with only a small decrease of 3.8 percentage points in recall.

The third strategy, PageRank, does not result in any improvement over the voting baseline (not even if we completely filter out values with low PageRank scores). Thus, we can confirm the finding of Dong et al. [Dong et al., 2015] that the quality of a web source is not necessarily determined by its popularity.

In summary, we find that KBT, a strategy that uses source reliability estimation, is able to provide the best performance fusion performance. In the remainder of this section, we will employ KBT for fusion.

Manual Evaluation of Fusion Performance

We perform two manual evaluations in order to verify the fusion results and the use of the LCWA for estimating fusion performance. First, we test how well the LCWA determines the precision of overlapping fused triples. Second, we test how far the precision of overlapping triples can be used to estimate the performance of non-overlapping triples. We evaluate in this section triples fused using KBT.

Precision of Overlapping Triples. We manually determine for a set of 1,000 overlapping fused triples whether they are correct or not. The automatic evaluation of this sample using the LCWA results in a precision of .678, while three human annotators determined a precision of .716. Overall 958 out of 1,000 triples were evaluated correctly by the automatic evaluation, which results in an error rate of 4.2%. This result is a signal for the validity of the LCWA and justifies its appli-

cation for our experiments. However, during the manual evaluation we spot some error categories, which shed light on possible shortcomings of this approach:

- **Time-Dependent Data.** The knowledge base could contain outdated information for time-dependent data, leading to an incorrect evaluation of more up-to-date web tables.
- **Different Granularity.** Objects can have different levels of granularity, e.g. the *city* of the *Emroy university* is *Druid Hills Georgia* in DBpedia. In the web tables, we find a reference to the entity *"Atlanta"*. These labels do not look similar to string comparison functions, but knowing that *Druid Hills Georgia* is a community in the metropolitan area of Atlanta, this triple can be regarded as correct.
- **Missing Objects in Lists.** If a list is incomplete in the knowledge base, the automatic evaluation fails for cases in which a web table contains a correct value, that is not covered by the knowledge base.

Precision of Non-Overlapping Triples. We manually determine the precision for a set of 500 randomly selected non-overlapping fused triples. As the non-overlapping fused triples are candidates for slot filling new facts, we are especially interested in determining whether the performance of overlapping fused triples can be transferred to non-overlapping fused triples.

The sample has a precision of .624.[3] The precision determined using the LCWA for overlapping fused triples for the KBT strategy was .639 (see Table 5.9). The two numbers are very close, which is an indication for the validity of transferring fusion accuracy of overlapping to non-overlapping fused triples.

5.4.2 Slot Filling Potential Profiling

Now that we have tested our methodology, we report details about the data fusion results with respect to the potential of web tables for slot filling. We show separate performance statistics for data types, classes and properties.

Data Types. Table 5.10 shows the fusion performance by data type. The first column F_O contains the number of fused triples that overlap with DBpedia, the second column F_{NO} the number of non-overlapping fused triples. All performance measures are calculated on the overlapping fused triples using the LCWA. While the date, reference and string data types have a comparable performance, the recall of data type numeric is considerably lower. As it seems, some numeric properties tend to be noisier due to conflicting objects, changes over time or different interpretations of certain properties. Thus, even correct triples are filtered out by the KBT

[3] 19 triples were excluded, as the human annotators could not determine the correct facts. This happened for example for rare properties like *bSide* of a record or *upperAge* of colleges.

fusion, as the trust score is not high enough. We further identified the following reoccurring causes of incorrect fusion results:

- **Conversion Issues.** Some conversions, e.g. converting date formats, are not easily resolved. As an example, the *birthDate* of *Jeff Zatkoff* is "6/9/1987" according to DBpedia, however we find the object "9/6/1987" in the web tables. Without knowing which date format is used within the web table, it is hard to parse the date correctly. This problem constitutes a large part of the error for the data type date.
- **Ambiguous Entities.** The identity resolution both for the subjects or values of type reference can be incorrect, especially if the label of the subject or object is ambiguous. This can occur with very common names of people or with musical works like album or single names, e.g. cover versions. A wrongly identified subject can lead to incorrect results for all data types, i.e. all fused triples are mapped to the incorrect knowledge base instance.

Table 5.10: Fusion results by data type.

Data Type	F_O	F_{NO}	P	R	F1
Numeric	28,364	10,613	.644	.452	.531
Date	171,653	23,301	.627	.806	.705
String	34,260	14,285	.755	.811	.783
Reference	144,615	16,038	.629	.871	.730

Classes. Table 5.11 shows the fusion results for the same set of classes also shown in Table 5.5. The second column contains the number of overlapping triples F_O per class while the third column shows the set of non-overlapping triples F_{NO}. All performance measures in the last three columns are computed on F_O using the LCWA. We find the highest amount of non-overlapping fused triples for *Work*, especially *Film*, and for *Person*, especially *Athlete*. This gives another hint for which parts of DBpedia slot filling using web tables can be beneficial. Concerning precision and recall, we achieve the best results for *Species* and *Place*.

Properties. Tables 5.12, 5.13, and 5.14 show performances for three different selection of properties. The first table shows properties with the highest number of overlapping fused triples, while the second table shows properties with the highest number of non-overlapping fused triples. Finally, the third table shows a selection of properties with a high precision and for which at least 50 non-overlapping triples can be fused. The columns titled F_O and F_{NO} show the number of overlapping and non-overlapping fused triples respectively, while the column titled P shows the precision. For the first two tables, we also include ratios relative the existing number of triples in DBpedia (DBP).

Table 5.11: Fusion results by DBpedia class.

Class	F_O	F_{NO}	P	R	F1
+ Person	117,522	15,050	.639	.723	.678
| – Athlete	84,562	9,067	.646	.679	.662
| – Artist	2,019	427	.711	.830	.766
| – OfficeHolder	3,465	510	.698	.849	.766
| – Politician	3,124	1,167	.533	.765	.628
+ Organisation	20,522	7,903	.645	.691	.667
| – Company	6,376	2,547	.700	.834	.761
| – SportsTeam	790	132	.671	.892	.766
| – Educational Institution	8,844	3,132	.638	.714	.674
| – Broadcaster	4,004	1,924	.557	.459	.503
Work	189,131	27,867	.614	.828	.705
+ MusicalWork	118,511	8,427	.599	.830	.695
+ Film	29,903	12,143	.573	.803	.669
+ Software	17,554	2,766	.591	.760	.665
Place	32,855	9,871	.767	.858	.810
+ PopulatedPlace	16,604	6,704	.711	.779	.743
| – Country	2,084	433	.738	.690	.713
| – Settlement	540	224	.583	.669	.623
| – Region	362	70	.587	.784	.671
+ Architectural Structure	10,441	1,775	.834	.940	.884
+ NaturalPlace	743	64	.843	.940	.889
Species	9,016	1,429	.783	.892	.834

Table 5.12: Fusion results for properties with most overlapping triples.

Property	F_O	P	$F_O/_{DBP}$
releaseDate	92,383	.628	.670
birthDate	61,636	.769	.055
artist	25,563	.649	.268
musicalArtist	20,663	.288	.527
musicalBand	18,160	.498	.463
director	8,082	.623	.095
activeYearsStartDate	7,934	.658	.116
activeYearsEndDate	7,861	.710	.140
deathDate	7,448	.625	.015
number	6,160	.383	.103

Looking at the properties with the most overlapping fused triples, we can again see that a large portion of the topical overlap is about *Work* (releaseDate) and *Person* (birthDate). Concerning the precision, most properties are close to our overall average performance, with exceptions being *musicalArtist* and number, which have a lower precision. Likely, this is caused by number (e.g. the number of a baseball player in a certain team) being a time-dependent property. For *musicalArtist*, the identity resolution could be a problem, as this property is applied to songs, which can often have ambiguous labels, e.g. also in the case of cover versions.

For the properties with the most non-overlapping fused triples, we approximate the precision with the precision that was achieved on the overlapping fused triples for the same property. The ratio column shows the potential for slot filling. We can almost double the number of *publicationDate* triples and increase the amount of *releaseDate* triples in the knowledge base by 11%.

Table 5.13: Fusion results for properties with most non-overlapping triples.

Property	$\mathbf{F_{NO}}$	**P**	$\mathbf{F_{NO}/_{DBP}}$
releaseDate	15,836	.628	.115
number	3,557	.383	.059
publicationDate	2,693	.688	.964
alias	1,471	.436	.011
locationCountry	1,304	.564	.089
country	1,242	.667	.002
synonym	1,240	.559	.014
status	1,116	.421	.042
birthDate	1,000	.769	.001
artist	971	.649	.010

Table 5.14: Fusion results for properties with highest fusion precision.

Property	$\mathbf{F_{NO}}$	**P**
numberOfIslands	157	1.00
province	67	1.00
seniority	60	1.00
sire	366	.990
games	247	.973
illustrator	81	.969
iso6391Code	236	.967
throwingSide	2,500	.961
icaoLocationIdentifier	5,459	.941
family	4,760	.846

To illustrate in which cases a slot filling approach would result in very high-quality data, the last table shows properties with high precision. While the properties with the highest precision can only add a rather small amount of non-overlapping fused triples, the properties *throwingSide* (for *BaseballPlayer*), *icaoLocationIdentifier* (for *Place*) and *family* (for *Species*) add thousands of triples with an above average precision.

5.5 Related Work

There exists research on the topic of large-scale web table profiling by Hassanzadeh et al. [Hassanzadeh et al., 2015]. The authors profile the same 2012 WDC web table corpus to analyze the topical distribution of web tables, by matching web table columns to classes in DBpedia, but also other knowledge bases including YAGO, Wikidata and Freebase. They show that the distribution of derived class correspondences is determined by which knowledge base the web table corpus is matched to. We confirm their finding, by showing that the most-matched classes correlate with the largest classes in DBpedia. The authors also find that only a relatively small fraction of the web tables can be matched to a cross-domain knowledge base. This is also a finding of our profiling. In our work however, we go beyond their analysis and do not only consider classes, but also properties and entities. We also examine the potential of web tables for slot filling missing values in the knowledge base.

Dong et al. in their work on Knowledge Vault [Dong et al., 2014a] present a method for automatically constructing a web-scale probabilistic knowledge base by combining data from four types of sources: web texts, DOM trees, web tables and semantic annotations. They extract overall 1.6 billion values matched to existing entities and properties in Freebase. Only about 0.5% originate from web tables, and around 3.8 million of those have an expected accuracy higher than 0.7. We extracted overall 8 million values that were successfully matched to a triple in the knowledge base. When fusing (and filtering) those values using KBT, we end up with about 443 thousand facts with an expected precision of 0.639. Dong et al. do not list the number of unique triples those 3.8 million web table values match to. As such, we believe that our numbers are generally comparable. To evaluate their approach, Dong et al. use the LCWA and show its validity. We also confirm the validity of the LCWA and use it to evaluate performance. However, we also show that the performance approximations based on the LCWA can be transferred to non-overlapping new triples.

5.6 Summary

In this chapter, we profiled the 2012 WDC web tables corpus with regard to its topical distribution and its potential for slot filling missing values in knowledge bases. We do this by first matching the corpus to DBpedia using T2K to then fuse matched values using Knowledge-Based Trust.

Our profiling shows that the majority of web tables does not contain data that can be related to DBpedia. Of the tables within the corpus, only about 2.85% could be matched to a class in DBpedia. We also found that the distribution of matched classes is similar to the overall distribution in DBpedia.

On the other hand, we find that 70% of matched DBpedia entities are described within at least two web tables. For properties this holds for 88%. Overall, 8 million web table values were successfully matched to 930 thousand unique triples, i.e. on average 8.5 values per triple.

We also examined recent results in the area of data fusion. We can first of all experimentally confirm the applicability of the Local Closed-World Assumption. We even show that the performance approximation for overlapping fused triples using the assumption is transferable to fused triples with no overlap in the target knowledge base, i.e. those that can be used for slot filling. Using the LCWA, we compare several fusion strategies and find that KBT outperforms PageRank as well as a voting-based fusion strategy.

Finally, we investigated the outcome of fusion to examine the potential of web tables for slot filling DBpedia in terms of quality as well as quantity. We provide detailed statistics for different classes and properties to get an impression which parts of DBpedia can especially benefit from slot filling with web table data. We can e.g. almost double the number of *publicationDate* triples in DBpedia.

Throughout this chapter, we focused on the task of static slot filling. Adding new entities to the knowledge base was not investigated, nor did we consider the aspect of validity in the case of time-dependent data. The remainder of this thesis is concerned with both, researching time-aware fusion methods for fusing time-dependent data from web tables and long-tail entity extraction methods to add new and formerly unknown entities to the knowledge base from web tables.

Part II

Time-Aware Fusion

Chapter 6

Exploiting Timestamps for Time-Aware Fusion

Consolidating information from a large number of web tables into a single knowledge base for slot filling requires two basic steps. First, the web tables are matched to the knowledge base. Second, data fusion is applied in order to select for a triple from among a set of conflicting web table values, the value that is most likely true.

Existing fusion methods for web data use source reliability estimation, which enables fusion methods to ensure the correctness of a fused value. A specific challenge in data fusion is that a large fraction of web table data is time-dependent. In time-dependent data, the validity of a value is dependent on a certain temporal scope [Kuzey and Weikum, 2012], i.e. a point in time or a time range. This time-dependent data includes for example the population of a city or the teams that an athlete plays for. Fusion methods for time-dependent data need to be time-aware, ensuring in addition to the correctness of the fused value, the validity of the value given a temporal scope.

For this, time-aware fusion methods require temporal scopes. However, in the case of web tables, which generally lack meta-information, explicit temporal scope annotations do not exist. As laid out by Dong et al. [Dong et al., 2016], fusing time-dependent web data therefore generally requires three subtasks: (1) identifying temporal references, (2) mapping them to a certain temporal scope, and then (3) assigning temporal scopes to values. For the first two subtasks, there exist rule-based approaches that are able to extract temporal scopes from timestamps found in web tables and HTML documents [Dong et al., 2016], e.g. in the form of the HeidelTime framework [Strötgen and Gertz, 2010, Strötgen and Gertz, 2015].

Regarding the third task, assigning an extracted temporal scope to a specific value or a set of values within a web table, two major challenges arise in the context of web table data [Zhang and Chakrabarti, 2013]. First, timestamp presence is sparse. Only few timestamps are present within web tables and web table column headers. This means that we are often unable to extract temporal scopes to assign to values in the first place. Second, while more timestamps might be present in

other cells of a web table as well as the page around the web table, it is challenging to determine whether this information applies to a specific web table value or not. As such, temporal scopes extracted from timestamps are also potentially noisy.

We propose TT-Weighting, a time-aware fusion approach that deals with the sparsity and the noisiness of temporal scopes extracted from timestamps. This is done by first categorizing extracted temporal scopes into timestamp types based on the location in the table or on the web page from which the scopes were extracted. To reduce sparsity, we introduce a method to propagate temporal scopes between values by timestamp type. E.g., given a web table value for which a temporal scope could not be extracted from the column header, we estimate this scope by propagating scopes extracted from column headers for other similar values. To deal with the noisiness of timestamp information, we train a regression model that learns weights for individual timestamp types, allowing us to aggregate noisy timestamp information into a single model with a possibly less noisy output. Finally, to ensure the correctness of a fused value in addition its validity, we combine these approaches with static fusion strategies that use source reliability estimation.

The contributions of this chapter are:

- **Taxonomy of timestamp types:** a taxonomy that allows us to exploit timestamps while differentiating between the locations from which they were extracted. We also introduce hierarchy types, that favor a timestamp given a certain predetermined order of locations. While existing works extract timestamps from various locations, timestamps from all location are treated equally and assigned the same weight [Zhang and Chakrabarti, 2013].
- **Temporal scope propagation:** an approach that deals with the sparsity of timestamps by propagation temporal scopes between values based on timestamp types. Existing works [Zhang and Chakrabarti, 2013] do not consider the location of the timestamp when propagating, while additionally, they propagate by whole table column, instead of more fine-grained by value.
- **Weighting timestamp types by property:** an approach that uses weighted-multiple-linear regression to learn weights for timestamp types given a property of the knowledge base schema. Existing works [Zhang and Chakrabarti, 2013] first do not assign weights to timestamps based on their location, and secondly, do not explicitly exploit a relationship between timestamp locations and the given property of the triple for which a value is being fused.
- A time-aware fusion strategy that combines these three approaches with source reliability estimation. We find that this strategy outperforms methods that only exploit source reliability when fusing time-dependent data.

We publish both our code and the datasets used for this research, while using a publicly available web table corpus within our experimental setup. As such, all resources required for the replication of this work are publicly available.[1]

[1] http://data.dws.informatik.uni-mannheim.de/expansion/time-aware-fusion/

Page Title: Country Data 2017

...

The following table provides information about those five countries, including capital, the national day, the current leader and current population. In comparison we provide population numbers from the year 1990

Country	Capital [15]	Current leader [98]	Population	Population 1990	National day
Germany	Berlin	Angela Merkel	81.69 M	79.43 M	3rd October 1990
France	Paris	Emmanuel Macron	66.81 M	58.51 M	14th July 1790
United Kingdom	London	Theresa May	65.14 M	57.25 M	-
Japan	Tokyo	Shinzō Abe	127 M	123.5 M	11th February 660 BCE
United States	Washington, D.C.	Donald Trump	321.4 M	249.6 M	4th July 1776

....

Figure 6.1: An illustration of a web table with time-dependent data and various timestamps.

This chapter is organized as follows. The next section presents an exploration of the task of fusing time-dependent web table data for slot filling. Section 6.2 outlines our experimental setup. Sections 6.3 and 6.4 respectively describe our methodology and discuss our results. We compare our approach to related work in Section 6.5 and discuss it in Section 6.6. The final section concludes this chapter.

The work presented in this chapter has previously been published in [Oulabi et al., 2016].

6.1 Motivating Example

Figure 6.1 illustrates a web table with three columns that describe time-dependent data: one leader and two population columns. The leader column corresponds to the year 2017, the first population column to 2015, and the second to 1990.

The table first of all shows that timestamps on the web page are not explicitly associated with data in the table. While a human might be able to understand which timestamp corresponds to which data, this task is non-trivial for a machine. Additionally, we find that not all relevant temporal scopes are present as timestamps. For example, the first column with population numbers contains data from 2015. This is a date not found anywhere on the web page, i.e. lacking a timestamp. Finally, timestamps unrelated to the data also exist, e.g. the year found in the footer of the website, or the national days found in the last column of the table.

We aim to use this data to complete time-dependent properties in a knowledge base. As outlined in Section 2.1.3, knowledge bases can represent time-dependent data in different ways. In this chapter, we test our methods on slot filling time-dependent data in a snapshot-based knowledge base, where the knowledge base tries to reflect the most current facts at the time of its release.[2]

[2]In the next chapter, we introduce time-aware fusion methods and test them on extending a temporal knowledge base, where time-dependent data is stored as a series of temporal facts.

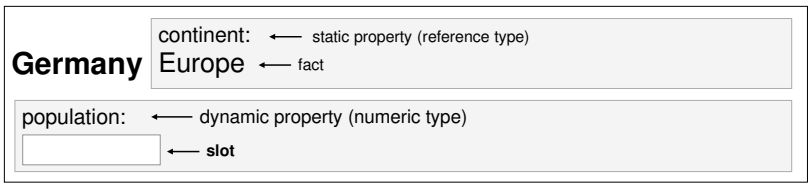

Figure 6.2: An illustration of an entity with an empty slot for a time-dependent property within a snapshot-based knowledge base.

Figure 6.2 illustrates an entity in a snapshot-based knowledge base. Within the entity, there is an empty slot for the time-dependent property *population*, i.e. a missing fact in the knowledge base. While this slot is not further annotated with a temporal scope, we know for example that the knowledge base was created at a certain point in time, e.g. the year 2015. As such, we can assume that the slot should reflect the population number for Germany in the year 2015.

We define this as *targeted slot filling*, where, in addition to knowing the entity and the property of a slot, we know that the slot corresponds to a certain target temporal scope. A similar task is performed by the InfoGather+ table augmentation system [Zhang and Chakrabarti, 2013]. In their approach, the authors require as part of the input query, in addition to the specific property for which values which should be extended using web tables, the target temporal scope of those values.

To use the data from the table in Figure 6.1 for targeted slot filling, we must first match the data to the knowledge base. Rows in the table are matched to entities in the knowledge base, while columns are matched to properties. Cells are therefore matched to triples. As a result, for a given triple in the knowledge base, population numbers from the both columns of the table are taken as candidate values for fusion. The task of time-aware fusion methods is then to find the candidate value that is both correct and valid given the target temporal scope of the slot. This can be done for example by assigning candidate values a temporal scope extracted from timestamps found in the table or the page around it.

6.2 Experimental Setup

This section outlines the experimental setup of this chapter. In summary, we built a ground truth for the two topical domains countries and NFL athletes, describing for each facts for two time-dependent properties. All facts within the ground truth are annotated using temporal scopes. We use the ground truth to evaluate fusion strategies for the task of targeted slot filling using web tables from the 2015 WDC web table corpus. To create correspondences between the web tables and the ground truth, we match both to the schema of and the entities within DBpedia using the T2K matching framework. We evaluate targeted slot filling on the ground truth using the *Local Closed-World Assumption* (see Subsection 4.3.4). As we also use the ground truth for training regression models, we perform our experiments using

five-fold cross validation. Throughout this chapter, we assume that all temporal scopes are points in time and with a granularity of years. This is similarly done in the experimental setup of related work [Zhang and Chakrabarti, 2013].

6.2.1 Ground Truth

While DBpedia is a snapshot-based knowledge base and it attempts to always reflect the most recent facts for time-dependent data, this is not ensured. As such, we refrain from using DBpedia for our experiments and create our own ground truth for a selection of two classes and four time-dependent properties from the schema of DBpedia. The classes included are Country and GridironFootballPlayer (GF-Player). The ground truth was built using data from the Worldbank[3] for Country, and from The Football Database[4] for GF-Player.

We overall extracted facts for 212 different countries for the years 2008 and 2014 and for the two properties, *Population* and *Population Density*. For GF-Player, we extracted facts for 737 athletes for the years 2010, 2012, and 2014 and the two properties *Team* and *Number*. We included in the ground truth only athletes that switched their team at least once within the three years.

In a temporal knowledge base, multiple temporal facts can exist per entity and property. In the context of a snapshot-based knowledge base, only one fact can exist. As such, for each triple we randomly choose the year for which we include the fact. For each included fact, we also include its temporal scope annotation to enable the evaluation of targeted slot filling.

Mixing the years in our ground truth is important. A non-time-aware fusion strategy that measures source reliability by using its overlap with a ground truth, e.g. Knowledge-Based Trust, could possibly implicitly capture the temporal scope of data in their source reliability score. In fact, this is the premise of Timed-KBT, an approach we introduce in the next chapter. By mixing values from different temporal scopes, we ensure that a time-aware fusion strategy that explicitly estimates temporal scope annotations is required for achieving a good performance.

We match both, the web tables and the ground truth to DBpedia. This is done automatically for the web tables using T2K (see Section 4.3). For the ground truth, the properties were matched manually, and only the entities were matched automatically using T2K. To evaluate the correctness of the entity matching, we randomly sampled and manually evaluated for each class 50 entity correspondences. In both classes, we achieve an accuracy of 100%. However, for countries, 49 out of 242 countries were not matched, while 286 out of 737 football athletes were not matched. We assume that all countries should have a match in the knowledge base. In the case of GF-Player, we looked at 15 unmatched entities, and found that all have a corresponding instance in DBpedia, so that we also assume that all entities

[3] http://data.worldbank.org/indicator/SP.POP.TOTL and http://data.worldbank.org/indicator/EN.POP.DNST

[4] https://www.footballdb.com/

Table 6.1: Number of facts in the ground truth and corresponding number of candidate values matched from web tables.

Class	Property	Type	Facts	Matches
Country	Population	Numeric	212	7,437
Country	Pop. Density	Numeric	212	80,598
GF-Player	Team	Reference	737	2,112
GF-Player	Number	Numeric	737	473

should have been matchable. This gives us estimates of recalls of 0.80 for countries, and 0.61 for football athletes. Using the accuracy to estimate precision, this gives us an estimated F1 of 0.89 for countries, and 0.76 for football athletes.

6.2.2 Web Table Corpus

For our experiments, we use the 2015 WDC web table corpus (see Section 3.3), which consists of 90 million web tables. This corpus also includes contextual information, possibly containing useful timestamp information. We use *HeidelTime* [Strötgen and Gertz, 2010] for both, detecting timestamps and extracting temporal scopes from those timestamps. For this, HeidelTime uses hand-crafted language-specific rules in the form of regular expression patterns, combined with additional post-processing and normalization. It achieved an F1 score of 86% on the TempEval-2 English-language Task A challenge, the best score achieved by any participating system [Verhagen et al., 2010].

As outlined above, correspondences between the web tables and the ground truth were generated by matching both to DBpedia. This was done for web tables using T2K (see Section 4.3). Table 6.1 shows the resulting number of matches between the web tables and the facts contained in the ground truth. Except for the property *Number* of the class GF-Player, we could on average find more than one candidate matched value in the web tables per triple in the ground truth.

Table 6.2 provides an overview of matched values by individual property. The first row shows the overall number of web table values matched to triples of that property, while the remaining rows show the number of matched values with a temporal scope extracted from a certain timestamp location. The locations *before table* and *after table* include timestamps found in text within 200 characters before and after the table respectively. *On page* includes on the other hand timestamps found anywhere on the page. We also extracted timestamps from the titles of the page and the table, the column header of the matched value as well as other cells of the same row. The last row of the table concerns temporal scope propagation, which we describe in Subsection 6.3.2.

Overall, we observe that most timestamps are found in text around the extracted table, while the second most frequent source is the page title. Few timestamps are found in the table title and for most properties few are also found in the column

Table 6.2: Timestamps present within the web table corpus.

Class Property	Country Pop.	Country Pop. Density	GF-Player Team	GF-Player Number
Matched values	27,297	104,502	35,035	12,472
Before table	6,254	82,512	27,413	8,268
After table	12,911	78,189	16,240	6,702
On page	4,893	4,576	4,224	4,787
In page title	4,615	44,128	17,425	6,904
In table title	0	297	8	0
In column header	284	71,399	42	0
In cells of same row	5	14	11,289	1,290
Inc. by Propagation	12.62%	29.30%	22.19%	2.35%

header. For GF-Player, we find a lot of timestamps in the cells of the same row of a value. Generally, the distribution of timestamp locations differs per property, which suggests that the usefulness of certain locations also differs per property.

6.2.3 Evaluation

We report performance using precision, recall and F1. We estimate all three using the Local Closed-World Assumption (LCWA), where we assume that facts present in the ground truth are correct and can be used to determine whether fused facts are correct. We outline this further in Subsection 4.3.4.

In order to measure recall, it is necessary to calculate the number of triples for which a perfect fusion strategy should be able to find a correct and valid fact. We include all triples where at least one of its matched web table values is equal to its fact in the ground truth. This is a very strict approach, as this does not ensure that this value, which happens to be equal to the fact in the ground truth, actually comes from data that has the same temporal scope as the target temporal scope.

6.3 Methodology

The fusion strategies outlined in this chapter are based on the underlying fusion methodology described in Section 4.3.2. Within this methodology, we assume that values extracted from web tables are already matched to a specific triple in the knowledge base, and fusion strategies influence the fusion outcome solely by scoring these matched values. In the context of time-aware fusion, fusion strategies are additionally provided a target temporal scope as input. As such, time-aware

fusion strategies can score matched candidate values additionally by how likely they are valid, given that target temporal scope.

We first introduce an approach to score candidate values given an extracted timestamp and a target temporal scope. We then introduce a taxonomy of timestamp types based on the location from which timestamps were extracted, allowing us to derive scores per timestamp location. To reduce timestamp sparsity, we suggest the propagation of timestamps along the timestamp type taxonomy, while to aggregate timestamp scores, we suggest an approach using linear regression. Finally, we also use linear regression to aggregate our time-aware fusion methodology with static fusion strategies.

6.3.1 Timestamp Type Scoring

To score matched candidate values from web tables using timestamps, we compare the temporal scope extracted from a timestamp (t) with the target temporal scope of the slot (TS_{target}). Assuming that all temporal scopes are years, the score is computed as following:

$$T(t) = max(1 - \frac{|t - TS_{target}|}{d}, 0) \tag{6.1}$$

This means that the score is 1.0 if the years a equal, and discounted by $\frac{1}{d}$ for each year of difference, where the lowest score is 0.0. With this, we score matched candidate values with temporal scopes closer to the target scope higher. If no temporal scope is assigned to the matched value, a score of 0.0 is returned.

The described scoring method requires one single temporal scope for each matched value. As discussed above, temporal scopes can be extracted from any timestamp present in the web table or its webpage. To differentiate between all those timestamps, we categorize them into *timestamp types*. This is done based on the location in which they were found.

Using the timestamp types and Formula 6.1, we generate a score for each individual timestamp type for every matched value. This still leaves us with multiple scores per matched value, however, these scores are now categorized into a taxonomy. As such, they can be aggregated using machine learning and regression, e.g. through the approach we introduce in Subsection 6.3.3.

We limit ourselves to temporal scopes in the form of points in time and with the granularity of years. Based on this, the different timestamp types that we extract are the following:

- **`ColumnHeader:`** year is extracted from the column header. Most recent year is chosen, if multiple exist.
- **`RowCell:`** year is extracted from cells in the same row of the matched value. Most recent year is chosen, if multiple exist.
- **`PageTitle:`** year is extracted from the page title of website. Most recent year is chosen, if multiple exist.

- **`TableTitle:`** year is extracted from the title of the table. Most recent year is chosen, if multiple exist.
- **`TableContext:`** year is extracted from text that surrounds the table. Most frequent year is chosen, if multiple exist.

For all of these types, except for `TableContext`, we expect them to contain only a single year value. In case of multiple values (e.g. within the header of the table), we choose the most recent value. For the `TableContext`, where we expect multiple year values, we choose the most frequent one.

In addition, we generated three hierarchy timestamp types, which we derive from the other non-hierarchy timestamp types. This is done by returning the temporal scope of one of the non-hierarchy timestamp types in a predetermined order. If the timestamp type highest in the hierarchy does not have a temporal scope, the temporal scope of the next timestamp type in the hierarchy is returned. The hierarchy timestamp types are:

- **`Hierarchy:`** hierarchy type with the order `ColumnHeader`, `RowCell`, `TableTitle` and `PageTitle`.
- **`FullHierarchy:`** hierarchy type with the order `Hierarchy` and `TableContext`.
- **`TableHierarchy:`** hierarchy type with the order `TableTitle` and `TableContext`.

The intuition behind the hierarchy types is that a certain location is more likely to be relevant than another. For example, a timestamp found in a column header is more likely to be relevant than a timestamp found in the text somewhere after the table. The hierarchies attempt to reflect this intuition.

6.3.2 Temporal Scope Propagation

Given that timestamp information is often sparse, we implement a temporal scope propagation approach based on timestamp types. This approach is able to induce temporal information for web table values where timestamp information is absent.

Given for example a web table describing countries and their populations without any timestamps, this table might not be considered by a time-aware fusion strategy that relies solely on extracted timestamps. Given however a second table containing very similar population numbers for the same countries as well as explicit timestamps, we could use this second table to propagate timestamp information to the original table. This is because, these numbers likely refer to the same point in time as the numbers in the table containing the timestamps.

For deciding whether or not to propagate timestamp information, we use the equivalence functions described in Subsection 4.3.5. Given a certain triple for which we matched a set of candidate values from the web tables, we cluster all matches using the equivalence measures. For each timestamp type, we investigate the values in each cluster for which we could find a corresponding timestamp

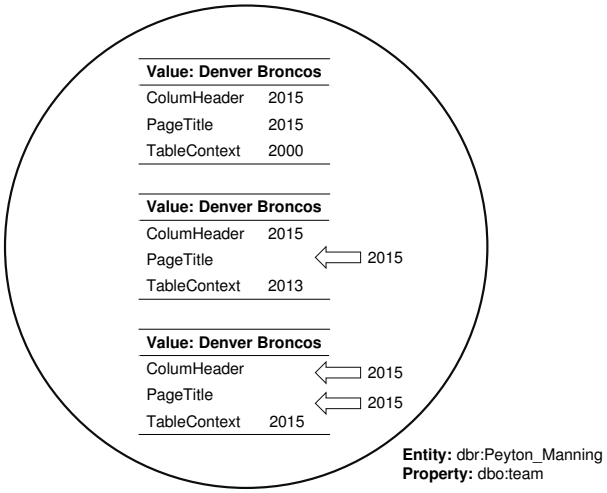

Figure 6.3: An illustration of temporal scope propagation based on timestamp types within a value cluster.

and identify the timestamp that appears most frequently. We then propagate this timestamp to all values in that cluster that do not have their own timestamp for that type.

Figure 6.3 provides an illustration of our propagation approach. We see a cluster of equal candidate values for the entity *Peyton Manning* and the property *Team*. This value cluster has a correct value that is valid for the year 2015, which is also the temporal scope of the target slot. The cluster however competes against other clusters with different candidate values. For the cluster to be returned as the cluster with the correct value, the individual candidate values within the cluster must achieve a high score when compared to the target temporal scope. However, two values lack temporal scopes for some timestamp types. We detect the most common temporal scope of those timestamp types within the cluster and propagate them to the values that lack the temporal scopes. In the end, the third matched candidate value, which had the least number of original temporal scopes, ends up with the highest number of correct temporal scopes.

In Table 6.2 we show how many new temporal scopes were created by property of our evaluation set. The increase in on average 16.62%.

6.3.3 Timestamp Type Weighting and Aggregation

As already mentioned in Subsection 6.3.1, there are possibly multiple scores assigned to a candidate matched value, each for a different timestamp type. As the extracted temporal scopes might differ per timestamp type, the various scores might conflict with each other. As such we require an approach that derives from these multiple scores one score per matched value. While with the hierarchies we attempt to reflect the importance of each type using a certain ordering, the chosen

order is based on our own judgment and should represent only a suggestion for which timestamp type is to be preferred.

We suggest therefore the use of trained weighted-multiple-linear regression models to aggregate the scores of multiple timestamp types using this formula:

$$y = c + \sum_{t=1}^{I} (x_t \times T(t)) \tag{6.2}$$

In the formula, t refers to the temporal scope extracted for a timestamp type for a value, and I is the overall number of timestamp types, including hierarchies, as described above. $T(t)$ refers to Formula 6.1.

The models are trained using existing facts in the knowledge base and individual candidate values matched to the triples of those facts. By comparing the values to the facts in the knowledge base using our equivalence functions, we can determine whether they are equal or not. We assign to candidate values a y value of 1.0 if they are equal, and of 0.0 if they are unequal. The regression algorithm then learns c and all the x variables.

A separate model is trained for each individual property of the knowledge base schema. Assuming that there is a certain relationship between a property and a timestamp type, we could capture this relationship by learning weights using regression. With these weights, we can then aggregate the various individual score into one score. This score is then assigned to the candidate value to be used within the fusion framework.

6.3.4 Aggregation with Static Fusion Strategies

A fusion strategy that aggregates all timestamp types using regression, while time-aware, does not consider the aspect of source quality. Incorporating source reliability estimation into a fusion strategy might be required, if we want a fusion strategy that ensures the correctness of a fused fact, in addition to its validity given a target temporal scope. We therefore suggest to combines both, scores computed by exploiting extracted timestamps and scores derived using static fusion strategies.

This is done by first simply including the score assigned to a candidate matched value by a static fusion strategy into our regression model. In addition, we also include another set of features, where we multiply the score returned by a static fusion strategy, with the score of every individual timestamp type. The intuition behind multiplying the timestamp type scores with static fusion scores is that a low score in estimating the correctness of a value would render a high score in the aspect of time-validity irrelevant, and vice versa.

Given for example the case, where we combine our timestamp types with Knowledge-Based Trust, the regression model would correspond to the formula below. KBT in the formula refers to the score assigned to a candidate value by a

fusion strategy based on Knowledge-Based Trust.

$$y = c + x_{KBT} \times KBT + \sum_{t=1}^{I} (x_t \times T(t) + x_{t_{KBT}} \times T(t) \times KBT) \qquad (6.3)$$

6.4 Experiments and Results

In this section we first evaluate a set of baseline static fusion strategies, to which we then compare a set of time-aware fusion strategies based on the methodology introduced above. The results for all experiments are presented in Table 6.3. We finally discuss in this section the importance of the weights assigned during regression. All regression models are trained by property of the knowledge base schema using five-fold cross-validation.

6.4.1 Static Fusion Strategies

Following, we will discuss the performance of three static fusion strategies. These include voting, PageRank and Knowledge-Based Trust. We describe these strategies in more details in Subsection 4.3.2.

Voting

Voting assigns all candidate values matched to a triple an equal score of 1.0. Within our fusion framework this strategy effectively resembles a simple count of the number of web tables that contain a specific value matched to a specific triple. This strategy is termed `Voting` in Table 6.3, and it achieves an average F1 of 0.27.

As all values have the same score, filtering values with a threshold is not applicable. In the case of only incorrect matched values, this necessarily leads to an incorrect fused value. Improving on this baseline method can therefore be achieved in two ways: (1) by scoring the values to allow a more effective selection of the correct group, and (2) thresholding possibly incorrect values with low scores.

Weighted PageRank

PageRank [Brin and Page, 1998] uses the link structure of the Web to rank websites. As such, PageRank can also be used to score candidate values for fusion.

To employ PageRank, we implement a scoring strategy based on the precomputed WDC Hyperlink Graph[5] [Meusel et al., 2015], which contains a full ranking of all hosts of the Common Crawl (2014). We score each candidate value with the precomputed score of its host.

We found that using the PageRank scores directly leads to a decrease of performance in comparison to `Voting`. While the precision increases marginally, the recall drops. We therefore learn a regression model using the PageRank scores for

[5] http://webdatacommons.org/hyperlinkgraph/

Table 6.3: Fusion performance for all fusion strategies.

	Population			Pop. Density			Team			Number			Average		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Voting	0.15	0.23	0.19	0.04	0.05	0.04	0.50	0.85	0.63	0.12	0.68	0.20	0.20	0.45	0.27
Ts	0.16	0.42	0.23	0.04	0.06	0.04	0.53	**0.99**	0.69	0.11	0.45	0.18	0.21	0.48	0.29
TsProp	0.16	0.44	0.24	0.13	0.13	0.10	0.53	0.98	0.69	0.26	0.32	0.28	0.27	0.47	0.33
wPr	0.22	0.39	0.28	0.06	0.08	0.07	0.53	**0.99**	0.69	0.07	**0.87**	0.13	0.22	0.58	0.29
wPrTs	0.21	0.39	0.27	0.18	0.12	0.09	0.53	0.97	0.68	0.17	0.24	0.18	0.27	0.43	0.31
wPrTsProp	0.20	0.41	0.26	0.11	0.12	0.09	0.53	0.98	0.69	0.24	0.24	0.22	0.27	0.44	0.32
Kbt	0.32	0.61	0.42	0.33	0.27	0.30	0.61	0.88	0.72	**0.56**	0.57	**0.56**	0.46	0.58	0.50
KbtTs	0.35	0.59	0.43	0.57	**0.34**	**0.42**	0.62	0.91	0.73	0.50	0.47	0.46	0.51	0.58	0.51
KbtTsProp	0.38	**0.64**	0.47	**0.60**	0.31	0.40	**0.64**	0.90	**0.75**	0.53	0.57	0.54	**0.54**	**0.61**	**0.54**
AllTs	0.34	0.59	0.43	0.57	0.27	0.35	0.62	0.84	0.71	0.44	0.40	0.40	0.49	0.53	0.47
AllTsProp	**0.39**	0.63	**0.48**	0.59	0.31	0.40	0.64	0.86	0.72	0.40	0.32	0.34	0.51	0.53	0.49
Δ(Voting, BestKBT)	**0.23**	**0.41**	**0.28**	**0.56**	**0.29**	**0.38**	**0.14**	**0.06**	**0.12**	**0.44**	**-.11**	**0.36**	**0.34**	**0.16**	**0.29**
Δ(KBT, BestKBT)	**0.06**	**0.03**	**0.05**	**0.27**	**0.07**	**0.12**	**0.03**	**0.03**	**0.03**	**0.00**	**0.00**	**0.00**	**0.09**	**0.03**	**0.05**

each class-property combination. The learned model, containing a weight for the PageRank score and a learned constant, reflects the importance of the PageRank score for that particular property.

The results for this fusion strategy are presented as `wPr` in Table 6.3. We find that we are able to outperform `Voting` in three out of four properties, however only marginally. On average we achieve an increase of 2 percentage points in F1.

When investigating the assigned weights, we find that the weight assigned to the PageRank score is the highest for the property *Population*, where the weight is six times larger than the fixed constant, followed by *Population Density*. The lowest relative weight was assigned to the property *Number*, where the learned weight of the PageRank score is about six times smaller than the constant. Given that the differences in the learned weights, this could potentially show that the importance of the PageRank score varies by property.

Knowledge-Based Trust

Knowledge-Based Trust (KBT) [Dong et al., 2014b, Dong et al., 2015] estimates the trustworthiness of a source by exploiting its overlap with a ground truth. As described in Subsection 4.3.2, we estimate using KBT the trustworthiness of a single web table column. For this, we first find the number of values in the column for which a fact exists in the knowledge base. This means, that the row of the value must be matched to an entity in the knowledge base, and that entity must have a fact present for the property to which the column is matched to. For those *overlapping* values, we then find the proportion of values that are correct. This proportion constitutes the KBT score of the column.

In Table 6.3, this strategy is termed `Kbt`. In comparison to voting and PageRank, using KBT for fusion has a large positive impact on performance for all properties. We achieve an average increase of 23 percentage points in F1 when compared to `Voting`.

6.4.2 Time-Aware Fusion Strategies

This subsection shows and discusses the performance of time-aware fusion strategies based on the methodology we introduced in Section 6.3. We differentiate between time-aware strategies by whether they were combined with a static fusion strategy, like KBT, or not. For all time-aware strategies tested in our experiments, we set the value of d in Formula 6.1 to 4.

Without Combination with a Static Fusion Strategy

We will first test two time-aware fusion strategies that do not integrate any static fusion strategy. First, `Ts` is a time-aware method, that uses linear regression to combine scores computed for all timestamp types, including the hierarchy types. `TsProp` is a strategy that additionally makes use of propagation.

When compared to Voting, Ts improves the performance for three of the four properties, on average by 2 percentage points in F1, equal to the increase achieved by wPr. On the other hand, TsProp is able to increases the performance for all properties when compared to Voting, achieving an average increase of 6 percentage points in F1. It also outperforms wPr by 4 percentage points in F1 on average. Both Ts and TsProp are still outperformed by Kbt.

Combined with PageRank

We then learned linear regression models that combine our time-aware fusion approaches with PageRank. This yields two strategies, one without propagation (wPrTs) and one with propagation (wPrTsProp).

The results, as shown in Table 6.3, are rather mixed. For wPrTs, only in two of the properties do we see an improvement, while in the other two we see a decrease in performance.

Combined with Knowledge-Based Trust

As we have seen, Kbt outperforms both Voting and wPr. We are therefore interested in the potential improvements achieved by extending this strategy with timestamp information. For this, we learn a regression model that combines the KBT score with all timestamp scores as outlined above in Subsection 6.3.4. We implement two strategies, one without propagation (KbtTs) and one with propagation (KbtTsProp). The results are shown in Table 6.3.

KbtTs is able to outperform Kbt in three out of four properties. The increase is even larger for KbtTsProp, where on average, we achieve an increase in 4 percentage points in F1 to Kbt. The performance for property *Number* is however anomalous. We find that F1 decreases by 10 percentage points when using KbtTs. While we are able to recover 8 of those points when using propagation, we are still unable to outperform Kbt when using timestamp information for the property *Number*.

At the bottom of the table we compare the best performance numbers achieved when using KBT with timestamp information with both Voting and Kbt. This is done by computing the absolute difference between the performance of either KbtTs or KbtTsProp, whichever is better, and both Voting and Kbt.

We find that, when compared to Voting, we are able to achieve on average an increase of 29 percentage points in F1. Compared to Kbt, the increase is 5 points on average. This could indicate that of the 29 points, 24 are due to KBT, while 5 are due to the inclusion of timestamp information. When excluding the *Number* property, which is comparatively anomalous, the increase compared to Voting is on average 26 points, while compared to Kbt it is 7 points.

The increase of 5 (or 7) percentage points in F1 from Kbt to either KbtTs or KbtTsProp is quite similar to the increase achieved by TsProp when compared to Voting. This could indicate that KBT and our time-aware fusion methodology

Table 6.4: Highest weighted timestamp types by fusion strategy and property for the class Country.

Strategy	Population	Pop. Density
Ts	PageTitle	TableHierarchy FullHierarchy
TsProp	PageTitle	TableTitle PageTitle
PrTs	PR_FullHierarchy	PR_Hierarchy
PrTsProp	PR_PageTitle	TableTitle PageTitle PR_TableTitle PR_PageTitle
KbtTs	KBT KBT_TableHierarchy KBT_FullHierarchy	KBT_TableContext KBT_Hierarchy KBT
KbtTsProp	KBT_PageTitle KBT	KBT_PageTitle KBT_TableHierarchy

are, as intended by design, dealing with separate aspects of conflict resolution, i.e. ensuring either correctness or validity.

Combined with PageRank and Knowledge-Based Trust

Finally, we implement a fusion strategy that combines timestamp information with PageRank and KBT. The performance of the resulting strategies, which are titled `AllTs` and `AllTsProp` can be seen in Table 6.3.

We find that for one property, our method is able to increase performance marginally, when compared to a time-aware method that makes use of only KBT. The performance for the remaining properties decreases. As such, we conclude that a time-aware method that already estimates source reliability using KBT, does not benefit from additionally integrating PageRank.

6.4.3 Utility of Weighting Timestamp Types

From the results presented in Table 6.3, we observe a positive impact when employing a method that weights features based on timestamp types and static fusion strategies using weighted-multiple-linear regression. However, the numbers do not provide any insight into which timestamp types are the most useful, whether the hierarchies are beneficial, and weather the learned weights differ per property.

Tables 6.4 and 6.5 show the features within the learned regression models

Table 6.5: Highest weighted timestamp types by fusion strategy and property for the class GF-Player.

Strategy	Team	Number
Ts	RowCell	FullHierarchy
TsProp	RowCell, FullHierarchy, PageTitle	FullHierarchy
PrTs	RowCell, PR_FullHierarchy, PR_RowCell	PR_FullHierarchy
PrTsProp	PageTitle, RowCell, FullHierarchy, somewhat lower for PR_RowCell, PR_FullHierarchy, PageRank	PR_FullHierarchy
KbtTs	Hierarchy, KBT, KBT_PageTitle	KBT, somewhat lower score for KBT_FullHierarchy
KbtTsProp	KBT, RowCell, somewhat lower score for FullHierarchy	KBT, somewhat lower score for KBT_FullHierarchy

where the weights were noticeably higher than other features. This is shown per fusion strategy and class-property combination. Features that multiply both, a timestamp type and a static fusion strategy are denoted with an underscore, e.g. KBT_FullHierarchy.

From the tables, we can observe that the timestamp types assigned higher weights clearly differ per property, but less so by strategy. We observe for example that the properties *Population* and *Population Density* often rank the type PageTitle very high. The importance of PageTitle for both properties is even more pronounced when using propagation. For *Team* the types RowCell and FullHierarchy have higher weights. While it seems that for the property *Team* and strategy KbtTs, the timestamp type RowCell is missing among the highest weighted timestamps, it is actually part of the Hierarchy timestamp type, where it is ranked second highest.

These observed patterns clearly indicate that there is a likely relationship be-

tween the importance of a timestamp type and a property of the knowledge base schema. This also suggest that with our learned weights, we are able to capture this relationship. This supports the idea of training a separate model for each property of each class.

Finally, we find that the hierarchies are an effective way of dealing with timestamps, as they are very often weighted highly. This applies especially to the hierarchy type `FullHierarchy`, indicating that all timestamp types included in the hierarchy are actually considered.

6.5 Related Work

This section provides a detailed overview of existing truth discovery and fusion methods for time-dependent web data. On the general topic of time-aware data integration methods for web data, there is a comprehensive survey by Dong et al. [Dong et al., 2016]. In their paper, the authors describe the requirements and challenges of integrating time-dependent web data. They also differentiate between the extraction of timestamps and their explicit mapping as temporal scopes, they however perceive both steps to be part of the extraction process, while we perceive the mapping as part of the fusion process. For the identification of timestamps, the authors suggest HeidelTime [Strötgen and Gertz, 2010], which is the method we also use in this work.

Table 6.6 provides an overview of works that introduce and evaluate time-aware fusion methods. TT-Weighting is the method introduced in this chapter, while Timed-KBT is another method that we introduce in the next chapter.

We describe and compare the methods in the table based on four aspects:

- **Time-Dependent Data.** There are generally two types of time-dependent data found on the Web that could require time-aware fusion: listing data and temporal knowledge. We describe the two types further in Subsection 2.1.3. For the case of extending a cross-domain knowledge base, we generally require methods that work on temporal knowledge.
- **Task Specification.** In this work, we focus on the task of targeted slot filling, where the fusion method is tasked with choosing from the conflicting values, the value that is valid given a target temporal scope. Another common task for time-aware fusion is to find the newest value.
- **Class Agnosticism.** As we are interested in extending knowledge bases that are cross-domain, time-aware fusion methods must be class-agnostic. They i.e. should not require class-specific implementations or supervision.
- **Temporal Scope Estimation.** Finally, web table data lacks explicit temporal scope annotations. A time-aware fusion method must in some way estimate temporal scopes. In the case of TT-Weighting, we exploit timestamp information to estimate missing temporal scopes.

Table 6.6: Overview of related work for time-aware fusion.

Method	Data	Task	CA	Temporal Scope Estimation
[Dong et al., 2009]	LD	NO	Yes	Crawl date of source
[Zhang and Chakrabarti, 2013]	TK	TF	Yes	Timestamp extraction / propag.
[Alexe et al., 2014]	TK	NO	No	Assumes existence
[Fan et al., 2014]	TK	NO	No	Not required
TT-Weighting	TK	TF	Yes	Timestamp extraction / propag.
Timed-KBT	TK	TF	Yes	Estimation using ground truth

LD listing data
TK temporal knowledge
CA class-agnostic
NO Newest Only
TF Targeted Fusion

Generally, the majority of time-aware fusion methods are not fully applicable for extending a cross-domain knowledge base from web table data. The primary issue is the estimation of missing temporal scope annotations. One work simply assumes the existence of annotations [Alexe et al., 2014]. Another work [Dong et al., 2009] employs the crawl date on which a source was crawled as the temporal scope of the crawled data. This is possible because the authors focus on listing data, however the crawl date of the web page from which a web table has been extracted has likely no relation to the temporal knowledge within the table. As such, estimating temporal scope annotations for web table data remains the primary problem of time-aware fusion, and is a task that remains unsolved by many methods. In addition, some works are either not class-agnostic [Alexe et al., 2014, Fan et al., 2014], consider primarily listing data [Dong et al., 2009], or attempt to find the newest value only [Dong et al., 2009, Alexe et al., 2014, Fan et al., 2014].

From the related work presented in this chapter, only InfoGather+ [Zhang and Chakrabarti, 2013] is an approach that considers a similar task to ours. The method performs targeted slot filling, estimates temporal scope from timestamps, is class-agnostic and is tested on temporal knowledge. We provide an in-depth look at this method, and other methods, in this section.

Truth Discovery and Copying Detection in a Dynamic World [Dong et al., 2009]

Dong et al. investigate a use-case where they regularly, i.e. every week, crawl and integrate a number of web sources. The authors suggest a method that estimates source reliability by considering transitions, which are changes in the data that occur between crawls. This is done by using a probabilistic model with three data quality measures: (1) coverage, which captures how up-to-date a source is, (2)

Table 6.7: Reported performances for time-aware fusion approaches by Dong et al. [Dong et al., 2009].

		Closed		
Method	**Ever-existing**	**P**	**R**	**F1**
All	n/a	0.60	1.00	0.75
All2	n/a	0.94	0.34	0.50
Naive	1192	0.70	0.93	0.80
CEF	5068	0.83	0.88	0.85

exactness, which captures how correct updates are when they occur, and (3) freshness, which captures how quickly sources change when an update is present.

The authors test their approach on real listing data for restaurants, considering only one property. They evaluate how well a time-aware fusion method is able to find whether at the end of the crawled period a restaurant is closed or not, when sources provide conflicting information. Table 6.7 shows the performance of their approach in comparison to three baselines. The first baseline (`ALL`) is a simplistic time-aware fusion method, where a restaurant is marked as closed when one source marks it as closed. The second baseline (`ALL2`) is similar, but there has to be at least two sources that change and mark a restaurant as closed, for a restaurant to be considered as closed. They also test their method on a non-time-aware method, which they term `Naive`. It resembles a voting strategy, as a restaurant is only considered open, when the majority of sources describe it as open. This achieves a good F1 score in regard to detecting what restaurants haves closed at the end of the crawled period, but it would fail in returning a complete list of open restaurants. This is because it considers a restaurant to actually ever exist, when it is mentioned as open by the majority of sources, which is not a realistic assumption.

From the table we can see that the author's approach (`CEF`), outperforms the baselines in time-aware fusion. This indicates that their model is able to effectively capture through the three quality measures the quality of a source when it comes to time-dependent data.

While the approach is evaluated on data, that is closer to listing data, i.e. restaurant listings, this itself does not reduce the applicability of the approach for extending a cross-domain knowledge base. For example, the approach can just as well be used to find the current team an athlete plays for, by regularly crawling football websites. The fact that the approach is only limited to finding the most recent value might reduce its applicability for extending some knowledge bases.

However, the approach can not be applied for fusing web table data, primarily because it relies on transitions between crawls and specific crawl dates. First, the crawl date of a web table can not be assumed to be the temporal scope of data in the web table. The authors can assume this for their sources, as they crawl sources that are by design supposed to show the most recent data. Additionally, while it

is possible to recrawl 12 web sources, web tables are extracted from the whole Common Crawl, and recrawling every web table repeatedly is not realistic.

Finally, the authors fail to compare their method to a state of the art non-time-aware method, i.e. one that measures source reliability. While a source reliability method primarily captures whether a source has a high quality or not, this quality might also cover how up-to-date a source is.

InfoGather+: Semantic Matching and Annotation of Numeric and Time-Varying Attributes in Web Tables [Zhang and Chakrabarti, 2013]

InfoGather+ is an approach that enables attribute expansion from web table data. For this, an input query defines a set of entities and one target property to be filled for those entities using data extracted from web tables. In case the property describes time-dependent data, a target temporal scope is also provided as part of the query. As such, the task resembles targeted slot filling. InfoGather+ also exploits timestamp information for the fusion of time-dependent data. Combined, these aspects make InfoGather+ comparable to our approach.

However, the authors have a highly different data integration and fusion approach. InfoGather+ employs probabilistic graphical models to build a semantic graph between web tables. This graph contains semantic matches between web table columns. A semantic match is defined as a match, where both the property and the temporal scope of the two columns are equal. As a result, the authors identify the assignment of temporal scopes as part of schema matching instead of fusion. Generally speaking, the authors perceive InfoGather+ as a matching approach, and not a fusion method. Regarding fusion and in case of conflicting candidates, they choose the ones with the highest aggregate score, computed based on the confidence of a semantic matches within the graph. In addition to temporal scope annotations, the authors also focus on other semantic labels like units or scales.

When it comes to slot filling for time-dependent data, InfoGather+ first creates the graph by finding columns matched to the same property that also contain values for the same temporal scope. This is done using an approach that resembles duplicate-based matching (see Section 4.2.2). This yields a graph between columns that likely describe the same property for the same temporal scope. To assign actual temporal scopes to a column, the authors extract scopes from timestamps within the column header and the context of a table. The temporal scopes are then propagated along the previously computed graph of semantic matches. As in our work, the authors focus solely on the year as the granularity of temporal scopes in their experiments.

Table 6.8 contains five experiments conducted within the paper for queries with a time-dependent property. For cities and companies, there are multiple experiments with different query sizes. We show the two experiments with the lowest and the highest query sizes. For company, the authors additionally conduct experiments for the property *Revenue*, which we do not include in the table, due to their similarity to the experiments for the property *Profit*.

Table 6.8: A selection of time-aware fusion experiments conducted by Zhang and Chakrabarti [Zhang and Chakrabarti, 2013].

#	Class	Property	Ground Truth	Query Size	Years
1	Country	Tax rate	OECD members	< 34	multiple
2	Cities	Population	600 largest cities	100	2011
3	Cities	Population	600 largest cities	400	2011
4	Company	Profit	Forbes Global 2000	100	2011
5	Company	Profit	Forbes Global 2000	500	2011

Table 6.9: Reported performances for experiments conducted by Zhang and Chakrabarti [Zhang and Chakrabarti, 2013].

	Baseline (S-Syn)			Collective Inference		
#	P	C	Correct Facts	P	C	Correct Facts
1	0.54	0.94	17	0.92	0.85	27
2	0.08	1.00	8	0.90	0.90	81
3	0.08	0.98	31	0.87	0.87	303
4	0.10	0.43	4	0.88	0.73	64
5	0.05	0.43	11	0.72	0.45	162

Table 6.9 shows the reported results for the given experiments. The authors report the precision (P) of returned facts. They also report coverage (C), which is the number of returned facts compared to the query size, no matter if the fact is correct or not. We believe that coverage is not a useful measure, as it does not indicate at all the correctness of the outcome. By multiplying the coverage with the query size and the precision, we can retrieve the number of correct facts returned.

The baseline approach treats target temporal scopes within the query as keywords, and considers tables only as matched to the query if the keywords are all contained in the column header. This baseline approach however still makes use of an underlying semantic matching graph when processing a query. The main approach suggested by InfoGather+ is titled Collective Inference. The authors do not compare their work to a method that exploits source reliability estimation.

InfoGather+ can achieve a large positive increase in precision, larger than the one we achieve. Compared to their baseline, they achieve an average increase in precision of 0.69, whereas compared to voting, we achieve an increase of 0.34.

However, the direct comparability of the results is limited. First, in experiments 2 to 5, the methods resolve both temporal scope and semantic labels like unit and scale. It is unclear what the individual impact of the use of timestamp information is. In experiments 4 and 5, the returned values must for example be in US dollars and in millions. This requirement for numeric conversions can explain the differ-

ence in performance between experiments 2 to 5 and experiment 1. In experiment 1, semantic labels for unit and scale do not matter. This could possibly explain why the increase from the baseline approach is smaller with just 0.38. However, it is still somewhat larger than the one we achieve with TT-Weighting.

The testing sets used by Zhang and Chakrabarti are however much smaller than the sets we use. They also consist entirely of head entities. In regard to countries, we use a ground truth with 212 entities, whereas they use a set with 34 entities. We include in our set almost all countries, whereas they only include countries that are part of the OECD, i.e. well-known and popular countries.

Table 6.9 also shows the number of correct facts per experiment conducted by Zhang and Chakrabarti. For comparison, in case of the better achieving properties *Population* and *Team* and the strategy `KbtTsProp`, we are able to retrieve 79 and 110 correct facts respectively. This means we are able to fill 37% and 15% of all facts in the ground truth from web table data.

TT-Weighting and InfoGather+ differ primarily in three ways when it comes to dealing with time-dependent data. First, Zhang and Chakrabarti consider InfoGather+ a matching method, whereas we focus entirely on the fusion of conflicting candidate values. This means that normalization of units and scale is part of InfoGather+, whereas in our case this is done by T2K during schema matching. Secondly, the authors consider temporal scopes only per column, and not per value. Any timestamps considered by InfoGather+ are always mapped to whole columns, not individual values. This for example, does not allow each value in a column to be assigned a different temporal scope, e.g. when using timestamps from other cells in the same row. We have shown that such timestamps are especially useful e.g. for the property *Team* of the class GF-Player. Thirdly, the propagation of timestamp information happens only on the basis of columns, and not on the basis of values or specific to the location from which a timestamp was extraction.

Finally, the authors make use of a non-public web table corpus, and do not provide code to replicate their methods. We provide all resources required to replicate our work, including code and the ground truth, while the web table corpus we use is publicly available.

Preference-aware Integration of Temporal Data [Alexe et al., 2014]

The authors present an operator that makes use of domain-specific preference rules that use the schema of integrated sources to determine which values are the most current. The work is not focusing on web data, nor on a large number of heterogeneous sources. As such, they also assume that temporal scope annotations are present. In some presented examples, the temporal scope annotations correspond to the time a dataset was released. In other examples the temporal scope annotations are provided through a time property that is part of the schema, and where schema mappings are known. Examples of preference rules described in the paper are: (1) ‘if the values conflict, choose the value where the date in the time property is higher’, or (2) ‘if the values conflict, choose the values from source A’. In their

Table 6.10: Reported performance for time-aware fusion approaches by Fan et al. [Fan et al., 2014].

	NBA	Career
Strategy	**F1**	**F1**
Random	0.38	0.84
Consistency CFD	0.78	0.91
CFD + currency constraints	0.80	0.97

approach the authors do not actually evaluate how correct the fused information is.

There are two main limitations of this work. First, temporal scope annotations are assumed to exist, limiting its applicability on web tables. More importantly, the approach is class and even source-specific. As the number of classes with a cross-domain knowledge base is large, a class-specific approach is not feasible.

Conflict Resolution with Data Currency and Consistency [Fan et al., 2014]

The introduced algorithm uses currency constraints to find the most current value given conflicting candidate values. The constraints are domain-specific, and have for example the following form: (1) 'given a conflict in the property *Points Scored* for the same NBA player, prefer the source where the number is higher', (2) 'given a conflict, where one source says that a person is retired, and the other the person is employed, prefer the source that for this case claims retired', or (3) 'given an identical author with two published paper where his affiliated institution differs, prefer the institution in one of the paper, if it cites the other paper'. The currency constraints are mined using a manually labeled domain-specific dataset, that contains temporal-scope annotations. The constraints are further manually refined.

The authors evaluate their approach on two datasets, one for NBA player statistics, and one extracted from Citeseer, for author career information. For the NBA dataset they extracted 760 players with 19,573 possibly conflicting facts, and for career information they extracted 65 authors with 2,080 facts. In addition to the currency constraints, the authors also make use of conditional functional dependencies (CFG) to ensure consistency of fused facts using other facts of an entity.

Table 6.10 shows the performance in F1 of the method as reported by the authors. The authors compared their time-aware fusion method (CFD + currency constraints) with one that solely ensures consistency (Consistency CFD) and one that fuses conflicting values by choosing a random value from the candidates. The absolute increase in performance from a non-time-aware fusion method to the time-aware is 2 percentage points in F1 for the dataset NBA, and 6 percentage points in F1 for the dataset Career. As such, the performance increase is comparable to ours.

The authors overcome the issue of missing temporal scopes, which is a problem that the authors explicitly mention in their work, by not requiring any during fusion.

Their method is able to fuse time-aware data solely by comparing the values among each other. However, the method requires manually labeled class-specific data with temporal scope annotations to be able to mine the currency constrains in the first place. The currency constraints additionally require manual refining. The issue of the labeled datasets can be resolved by using a temporal knowledge base as a ground truth. However, manual refinement limits the potential of applying the methods to extend a cross-domain knowledge base.

Additionally, the method requires a type of knowledge, where from the available data itself it becomes obvious which value is newer. This might be possibly for points scored by an NBA player, but might not be possible for population of a country, or the profit of a company. Also, the task solved by the authors is not targeted slot filling, but finding the most current value. This might limit its applicability for some knowledge bases. Finally, the authors test their data on conflicts within a small number of rather clean sources with known schema mappings, which yields an easier task than integrating data from a large-scale, noisy and heterogeneous web table corpus.

6.6 Discussion

Through the taxonomy of timestamp types, we are able to both, propagate temporal scopes at a more fine-grained level, and, learn relationships between certain timestamp locations and properties of the knowledge base. Based on our analysis of the weights assigned during regression, there is clear indication that these relationships exist, supporting the idea of exploiting them for time-aware fusion.

We are also able to successfully combine time-aware fusion strategies that exploit timestamps with strategies based e.g. on source reliability estimation. This yields a fusion method that considers both, the correctness of a candidate value, and its validity given a target temporal scope. We find in Section 6.4 that aggregating time-aware fusion with either, a static strategy based on voting or on Knowledge-Based Trust yields a similar average increase in F1 in both cases. This indicates that a strategy based on source reliability estimation and the time-aware fusion approaches introduced in this chapter are indeed dealing with different aspects of conflict resolution, i.e. correctness and validity.

Our approach could be improved in two ways. First, we can additionally consider learning a relationship between a certain web table column and the timestamp types. This would allow us to be capture more specific relationships, when compared with just learning per property of the knowledge base schema. Secondly, we could alternatively train regression models that are more expressive than the ones learned using weighted-multiple-linear regression.

The primary weaknesses of this chapter lie in the experimental setup. First, we evaluate on too few properties. More importantly, the way we use the LCWA to determine the maximum recall is possibly too strict. We assume that our fusion strategy should be able to find a correct and valid value, if a value, that happens

to be correct, exists among the matched candidates. However, this value might actually be valid for a different temporal scope, than the one in the target slot. An improvement could be, that values from one table column, if they are to be counted towards recall, all just count for slots with one equal target temporal scope. A more comprehensive approach would be to manually identify the maximum recall using an annotated gold standard, instead of using the LCWA.

6.7 Summary

In this chapter, we presented a time-aware fusion method for fusing time-dependent data from a large corpus of web tables. To the best of our knowledge, we introduce with this the first work to perform time-aware fusion explicitly for the task of slot filling time-dependent data for knowledge base augmentation.

Our approach works by exploiting timestamps found in the table and the page around it. We first introduce a taxonomy of timestamp types, where we differentiate between the timestamps based on which locations they were extracted from. This taxonomy allows us to be much more particular when using timestamp information. We also introduce timestamp hierarchies, where we exploit timestamps in a predetermined order of locations.

We first introduce an approach to propagating timestamps on the basis of timestamp types. This allows us to reduce timestamp sparsity by estimating timestamps for some web table data using other web tables within the corpus. Using this approach, we are able to increase the number of timestamps by 16.62% on average.

Using weighted-multiple-linear regression, we then learn models that capture relationships between timestamp types and a given property from the knowledge schema. These models allow us, given conflicting and noisy timestamp information, to identify which timestamps are relevant when fusing a web table value and to score values accordingly. When investigating learned weights, we find that some timestamp types are assigned higher weights for some properties, which indicates that there is indeed a relationship between the locations from which timestamps are extracted and a given knowledge base property.

Finally, we introduce an approach that again uses regression to combine the approaches described above with a static fusion strategy like Knowledge-Based Trust (KBT). This yields a time-aware fusion strategy that considers both aspects of conflict resolution, i.e. correctness and validity given a target temporal scope.

We find that while KBT outperforms other baseline static fusion strategies, we can further improve KBT by around 5 percentage points in F1 by aggregating it with our time-aware fusion approaches that exploit timestamp information.

While the time-aware fusion methods introduced in this chapter are useful in reducing the noisiness and sparsity of timestamps, they still rely on timestamps extracted from the table and its context. In the next chapter, we introduce an approach that is able to overcome the reliance on timestamps by estimating temporal scope annotations using a temporal knowledge base.

Chapter 7

Estimating Temporal Scopes Using Knowledge-Based Trust

To fuse time-dependent web table data, time-aware fusion strategies require temporal scope annotations. Existing approaches like InfoGather+ [Zhang and Chakrabarti, 2013] and TT-Weighting, the method introduced in the previous chapter, estimate the temporal scopes of web table data by extracting timestamps from tables and their contexts.

Fusion strategies that solely make use of timestamp information suffer from two problems. First, the relationship between timestamps and the data in the table is often unclear. More than one timestamp can usually be extracted per table and many of the extracted timestamps likely have no relevance to the data in the table at all. Secondly, web tables suffer from timestamp sparsity, so that for many tables we are unable to extract any timestamps. Given these limitations, it would be highly useful to be able to estimate missing temporal scope annotations using other sources and without depending solely on timestamps.

This chapter introduces Timed-KBT, an approach that estimates missing temporal scopes using Knowledge-Based Trust (KBT) [Dong et al., 2015]. KBT estimates the trustworthiness of data sources using their overlap with a ground truth. It is based on the idea that non-overlapping data shares similar quality with neighboring overlapping data. This shared quality possibly incorporates multiple dimensions, e.g. data, extraction, and matching quality.

Timed-KBT is based on the assumption that the temporal dimension is one of the qualities shared by neighboring data. The idea is to use a temporal knowledge base as ground truth to detect temporal scopes for overlapping values and propagate these scopes to neighboring non-overlapping values. For example, let's consider a web table column with time-dependent data, where there are values within this column that are already covered by our knowledge base, i.e. they overlap, and values that we could potentially use for slot filling. If we are able to determine that a majority, if not all, overlapping values correspond to one certain temporal scope, we could, based on our assumption, assign this temporal scope to the non-

overlapping values. This temporal scope could then be used to enable time-aware fusion.

The contributions of this chapter are:

- **Timed-KBT:** a method that can estimate missing temporal scope annotations for web table data using a temporal knowledge base as ground truth. This approach overcomes the dependence on timestamp information and enables the time-aware fusion of time-dependent web table data that lacks any timestamps. Previous works [Zhang and Chakrabarti, 2013] for fusing time-dependent web table data always relied on the existence of timestamps.
- **The Time-Dependent Ground Truth (TDGT):** a publicly available dataset consisting of 7 classes with overall 19 time-dependent properties. The dataset resembles in structure a temporal knowledge base, in our case based on the schema of and matched to entities within Wikidata. TDGT can be used as a ground truth for any tasks that consider the time aspect of data. We employ it as a ground truth to be exploited by Timed-KBT.
- We evaluate Timed-KBT on the use case of data fusion using a large corpus of web tables and TDGT as a target knowledge base. We find that Timed-KBT is able to estimate missing temporal scope annotations effectively and improve fusion results.
- We further introduce and evaluate a version of Timed-KBT, where the set of scopes that can be assigned to web table data is restricted to timestamps present in the table or its context. By using timestamps as a restriction for Timed-KBT, we are furthermore able to derive a precision-orientated time-aware fusion method.

.

We publish both our code and the datasets used for this research, while using a publicly available web table corpus within our experimental setup. As such, all resources required for the replication of this work are publicly available.[1]

This chapter is structured as follows. The next section provides a motivating example and describes the overall use case at hand. Section 7.2 describes our experimental setup, while Section 7.3 describes the Timed-KBT approach. The results are discussed in Section 7.4, while Section 7.5 frames this chapter within related work and provides a discussion. Section 7.6 is our conclusion.

The work presented in this chapter has previously been published in [Oulabi and Bizer, 2017].

7.1 Motivating Example

Figure 7.1 shows a modified version of the illustration of a web table presented in Figure 6.1 in the previous chapter. Like the original, it has two columns with

[1] http://data.dws.informatik.uni-mannheim.de/expansion/time-aware-fusion/

Page Title: Country Data 2017

...

The following table provides information about those five countries, including capital, the national day, the current leader and current population. In comparison we provide population numbers from the year 1990

Country	Capital [15]	Current leader [98]	Population	Past Population	National day
Germany	Berlin	Angela Merkel	81.69 M	79.43 M	3rd October 1990
France	Paris	Emmanuel Macron	66.81 M	58.51 M	14th July 1790
United Kingdom	London	Theresa May	65.14 M	57.25 M	-
Japan	Tokyo	Shinzō Abe	127 M	123.5 M	11th February 660 BCE
United States	Washington, D.C.	Donald Trump	321.4 M	249.6 M	4th July 1776

....

Figure 7.1: An illustration of a web table with time-dependent data and various timestamps.

time-dependent data for the same property *Population*. One column corresponds to the year 1990, and one to the year 2015. As in the original example, the year 2015 is a date not found anywhere on the web page, i.e. lacking a timestamp. In this version of the example however, the column with the 1990 population additionally lacks the timestamp in the column header.

The previous chapter introduced TT-Weighting, a time-aware fusion strategy that exploits timestamps by learning weights for timestamp types. While TT-Weighting has a positive impact on performance, the weights were learned per property of the knowledge base, and therefore capture a relationship between the timestamp types and a property. In the example in Figure 7.1, we see that two columns, that both describe data for the property *Population*, have however different relationships with the timestamps on the web page. For one of the columns, one temporal scope is additionally not even present at all amongst the timestamps found on the website. TT-Weighting is first of all not able to capture the difference between the columns, and secondly not even able to find the correct temporal scope for one of the columns due to timestamp sparsity.

In TT-Weighting, we attempt to reduce this problem using propagation at the level of values. Propagation is used to fill in the sparsity of missing temporal scope annotations by looking at what timestamps were assigned to equal matched values in other tables of the corpus. In the example in Figure 7.1, individual values within the population columns, might through propagation be assigned a correct temporal scope for the timestamp type of the column header. However, this means that propagation still essentially requires that correct timestamp information to be present in the first place somewhere in the web table corpus. As such, the primary approach to improving time-aware fusion is to find additional sources for estimating temporal scopes and reduce the dependency on timestamps.

To capture the difference between the two population columns in the example, we need a method for estimating temporal scope annotation that is both, specific to a web table column, instead of a knowledge base property, and does not require the

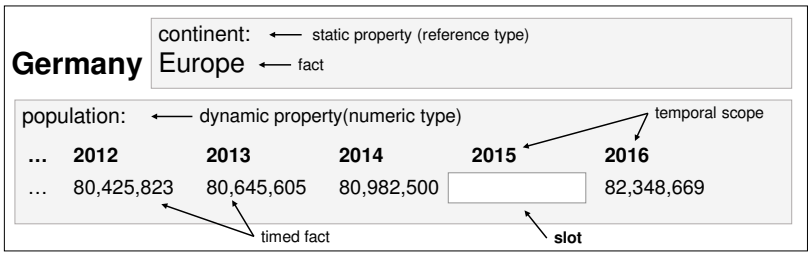

Figure 7.2: An illustration of an entity with an empty slot for a time-dependent property within a temporal knowledge base.

actual presence of timestamps. Timed-KBT is able to deal with those challenges by learning a temporal scope per web table column using a temporal knowledge base as a ground truth.

We evaluate Timed-KBT for the task of targeted slot filling. Unlike in the previous chapter, where we investigated this task for a snapshot-based knowledge base, in this chapter we make use of a temporal knowledge base. Temporal knowledge bases store time-dependent data as a series of temporal facts, where each fact is annotated with a temporal scope. For each triple, a temporal knowledge base tries to reflect all current and historic facts.

Figure 7.2 illustrates knowledge for the entity *Germany* in a temporal knowledge base. The example contains two properties, one static (*Continent*), and one time-dependent (*Population*). While the knowledge base contains data for both properties, for *Population* one fact is missing for the year 2015. The task of targeted slot filling is to find the missing fact for this specific target temporal scope.

Targeted slot filling for a temporal knowledge base is therefore in essence equal to that for a snapshot-based knowledge base. A temporal knowledge however contains a large number of temporal facts, which contain explicit temporal scope annotations. When it is used as a ground truth, it can possibly enable types of learning not possible with a snapshot-based knowledge base. This is the case for Timed-KBT, which exploits the temporal scopes in a temporal knowledge base to estimate missing temporal scopes for web table data.

7.2 Experimental Setup

This section describes the experimental setup of this chapter. First, we built from a large number of sources and using the schema of and entities within Wikidata [Vrandečić and Krötzsch, 2014] a dataset that contains time-dependent data. We employ this dataset in lieu of a temporal knowledge base to be extended and as a ground truth that can be used for learning. As in the previous chapter, we use the 2015 WDC web table corpus, and create correspondences to the ground truth using the T2K matching framework. Finally, we set up experiments using the Local Closed-World Assumption, to evaluate how well we are able to perform time-aware fusion

Table 7.1: Overview of classes, entities, time-dependent properties and facts in the Time-Dependent Ground Truth.

Class	Entities	Prop.	Facts	Sources
Basketball Athl.	3,781	1	10,625	basketball-reference.com
City	11,372	2	40,666	wikidata.org
Country	197	7	44,013	worldbank.org, wikidata.org
NFL Athlete	12,756	2	96,711	footballdb.com
Office holder	22,062	1	36,816	wikidata.org
Soccer Athlete	134,617	1	778,602	wikidata.org
Traded Company	1,646	5	73,806	stockrow.com

using Timed-KBT. Throughout this chapter, we assume that all temporal scopes are points in time and of the granularity year. This is similarly done in the experimental setup of related work [Zhang and Chakrabarti, 2013].

7.2.1 The Time-Dependent Ground Truth

We build the ground truth on a subset of the schema of the temporal knowledge base Wikidata [Vrandečić and Krötzsch, 2014]. The properties in the subset were chosen based on a profiling of the web table corpus to ensure overlap with web table data. However, for some of the chosen time-dependent properties, Wikidata did not contain sufficient data for a proper evaluation. To create the ground truth, we therefore extended the subset with various datasets that cover time-dependent data. The resulting Time-Dependent Ground Truth (TDGT), has been published as part of the WDC project.[2]

Table 7.1 provides an overview of the classes, entities, properties and facts in TDGT. The table also shows by class from which sources datasets were used to complement Wikidata. We acquired data from these sources either by manually written crawlers and extractors, or through data dumps. Acquired datasets were matched to entities in Wikidata using class-specific matchers, while the schema was mapped manually.

Not all the classes and properties within TDGT were used in the experimental setup of this chapter. Table 7.2 shows the properties actually included in the experiments and provides further statistics for each property.

7.2.2 Web Table Corpus

For our experiments, we use the 2015 WDC web table corpus[3], which was extracted from the July 2015 Common Crawl. The corpus consists of 90 million relational HTML tables, containing additionally timestamp and contextual data [Lehm-

[2] http://webdatacommons.org/TDGT/

[3] http://webdatacommons.org/webtables/#toc2

Table 7.2: Overview of properties included in our experimental setup, the number of triples (series of temporal facts), and the overlap with the web table corpus.

Class	Property	Triples	Average # of matches	Overlap
Basketball Athl.	Team (R)	2,296	2.64	1,064
City	Population (N)	5,884	4.25	3,120
Country	Nominal GDP (N)	191	15.14	2,020
Country	Nom. GDP p. capita (N)	191	68.27	7,734
Country	Population (N)	193	43.93	8,032
Country	Pop. Density (N)	193	100.37	8,805
NFL Athlete	Number (N)	11,295	4.35	9,344
NFL Athlete	Team (R)	8,962	3.56	1,075
Soccer Athlete	Team (R)	14,970	2.12	4,688

Table 7.3: Proportion (in %) of matched values containing timestamps extracted from certain locations.

Class	Property	before	after	on page	page title	table caption	column header	in row	at least one
BA	**Team**	52.10	38.75	21.86	26.27	1.14	1.45	30.70	72.16
CI	**Population**	58.04	55.88	21.74	20.48	2.68	16.74	10.19	83.00
CO	**Nom. GDP**	45.38	46.07	24.49	12.38	0.80	3.01	26.25	68.21
CO	**Nom. GDP-PC**	53.71	55.90	23.71	21.29	3.70	12.99	19.65	76.01
CO	**Population**	52.47	56.03	27.72	27.07	0.96	14.58	21.70	78.65
CO	**Pop. Density**	56.28	55.42	26.62	27.40	1.98	9.17	19.00	78.62
NA	**Number**	62.20	39.46	30.37	45.31	1.23	0.46	28.87	83.22
NA	**Team**	69.24	44.14	23.78	51.18	1.17	2.50	30.79	84.56
SA	**Team**	69.38	47.41	38.64	63.72	2.91	0.04	8.31	89.11
	Average	**57.64**	**48.79**	**26.55**	**32.79**	**1.84**	**6.77**	**21.72**	**79.28**

berg et al., 2016]. We use the matching component of the T2K Framework [Ritze et al., 2015] to match the corpus to TDGT. Columns in the web tables are matched to properties, while the rows are matched to entities.

Table 7.2 describes the properties which we include in our experimental setup. Properties of datatype reference and numeric are denoted by (R) and (N) respectively. The column 'Triples' lists the number of triples of a property for which values from the web tables were matched. Each triple consists of a series of temporal facts, where each fact is annotated with the temporal scope for which it is valid. If a series is lacking one of its temporal facts, it is seen as a slot with a target temporal scope. Any web table value matched to a triple is seen as a candidate for any of its slots. The next column in Table 7.2 shows how many matched values exist per triple on average. The column 'Overlap' measures for how many temporal facts in the knowledge base there were candidate matched values which were equal to the fact, i.e. possibly correct matches. We have filtered from the corpus all

tables extracted from sources that were used to create the TDGT.

We additionally extracted timestamps using *HeidelTime* [Strötgen and Gertz, 2010, Strötgen and Gertz, 2015]. Table 7.3 shows the proportion of sources that have timestamps in certain locations. Columns '*before*' and '*after*' refer to timestamps found in the context before and after the table respectively. Column '*on page*' refers to timestamps found anywhere on the page, while column '*page title*' refers to those found in the page title. The following three columns refer to timestamps extracted from table captions, column headers and cells of the same row of a value. The final column gives the proportion of sources for which a timestamp can be extracted from at least one location.

Most timestamps are found in the context of the table, which could mean that they have no explicit relation to the data in the table. Timestamps extracted from cells of the same row could similarly describe an unrelated date attribute, e.g. the 'National day' column in Figure 7.1. Timestamps in table captions and column headers, which are likely to be the most relevant, are also the least present. Presence also differs by class: For *Country* and *City* we find many timestamps in the column header, while for *NFL Athlete* we find more in cells of the same row.

7.2.3 Evaluation

To test our fusion methods, we make use of the Local Closed-World Assumption (LCWA), where we assume that facts present in the knowledge base are correct and can be used to determine whether fused facts are correct. We described how we use the LCWA in Subsection 4.3.4 and empirically examined it in Chapter 5.

We use the F_β score [Manning et al., 2008], as shown in Formula 7.1, to measure fusion performance. The F_1 score, which is the most commonly used F_β sore, has equal weights for both precision and recall. For the task of slot filling, we must ensure the correctness of filled facts, so that we care primarily about precision. We therefore compute results for F_β score at β of 1.0 and 0.25, where the latter weights precision four times as high as recall. The choice of β also affects the learned filtering thresholds, as we will describe further below. We measure performance per class-property combination.

$$F_\beta = (1 + \beta^2) \times \frac{Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{7.1}$$

One difficulty of using the LCWA is measuring recall, i.e. for how many of the evaluated slots we can actually find a correct value assuming a perfect fusion. We consider every slot, for which a candidate value from the web table corpus is matched to, while at the same time a value is equal to the existing fact of the slot in the ground truth, as a slot that counts towards recall. This is a very strict approach, as this does not ensure that the web table value, which happens to be equal to the fact in the ground truth, actually comes from data that has the same temporal scope as the slot.

As the ground truth is used for both, learning and testing, we split the data four times, each time placing approximately 25% of the data in the testing set, and the remainder in the learning set. To replicate the use-case of targeted slot filling, where some missing slots within a series are to be filled, we split by series of temporal facts, so that some temporal facts of a triple are in the testing set, while the remaining are used for learning. To ensure that the temporal scopes of removed facts are well distributed, we randomize how each series is split.

7.3 Methodology

This section introduces the underlying methodology behind Timed-KBT. It works by exploiting the temporal scopes in a temporal knowledge base, and the overlap between a web table column and the knowledge base, to assign an explicit temporal scope to that column. While Timed-KBT itself does not require timestamp information, we additionally introduce an approach that limits the temporal scopes that can be assigned to a web table column to the scopes extracted from timestamps present in the table and its context.

This section also introduces neighboring scope estimation. This is an approach we use with Timed-KBT to allow web table data that was assigned a specific temporal scope to be used for the fusion of a slot with a neighboring temporal scope.

This section however starts by first describing the fusion framework all strategies are based on, and second by introducing two baseline static fusion strategies, voting and Knowledge-Based Trust. We compare Timed-KBT to both.

7.3.1 Fusion Framework

All strategies introduced in this section make use of one common fusion framework introduced in Subsection 4.3.2. Essentially, we assume that all candidate web table values are matched to a slot of the knowledge base, where each slot is defined by an entity and a property. In the context of targeted slot filling, each fusion strategy is also given a target temporal scope for each slot to be filled.

Within our fusion framework, we learn per strategy and property of the knowledge base a threshold, that filters out matched values that were assigned a low score by the fusion strategy. This threshold is learned on the training set, and is optimized for F_β, so that different threshold will be learned, based on what β we use for evaluation.

7.3.2 Static Fusion Strategies

`Voting` is a common baseline strategy [Dong et al., 2014b], where all matched candidate values are assigned a score 1.0. Given a target slot, the value that hast the highest number of matched candidates is simply chosen, regardless of the correctness of the value or the validity given the target temporal scope. E.g., values from the table in Figure 7.1, which has only correct data, will share the same score

as values of a table with mostly incorrect data. As all scores are equal, no thresholding is possible. Additionally, `Voting` is not time-aware, so that the fused facts for all target slots within the same triple will be the same.

`KBT` is a static fusion strategy based on Knowledge-Based Trust [Dong et al., 2015]. It uses the correctness of data that overlaps with the knowledge base to estimate a trust score for the remaining data. It is based on the assumption that neighboring values share similar qualities. As data within a single web table column has equal extraction, normalization, matching, and potentially factual quality, we compute, using the following formula, `KBT` scores per web table column.

$$\text{KBT(column)} = \frac{\text{\# values in column with correct overlap}}{\text{\# values in column with overlap}} \tag{7.2}$$

As `KBT` is not time-aware, the fused facts of all target slots within one triple will be the same. With `KBT`, the values in the table in Figure 7.1 will have a higher score than a table with mostly incorrect data, while both population columns will still be used as fusion candidates for a target slot with any target temporal scope.

7.3.3 Timed-KBT

Timed-KBT assigns explicit temporal scopes to web table data by exploiting its overlap with a temporal knowledge base. It is based on the assumption that neighboring values, e.g. within one column, share a common temporal scope. The idea is to use the knowledge base to detect this scope for overlapping values and propagate the scope to neighboring non-overlapping values.

To generate missing temporal scopes, we find the temporal scope t that maximizes the KBT_t score of a column. The KBT_t score is computed by only using values from the knowledge base that are annotated with the given temporal scope t. We assign t to the to the web table column, while the KBT_t score itself is then used as the fusion score of the matched values. In the table in Figure 7.1, Timed-KBT will e.g. be able to assign different scopes to the population columns, as the temporal scope t that maximizes KBT_t will likely differ per column.

$$\text{KBT}_t\text{(column)} = \frac{\text{\# values in column with correct overlap given scope t}}{\text{\# values in column with overlap given scope t}} \tag{7.3}$$

$$t_{column} = \arg\max_{t \in T} \text{KBT}_t\text{(column)} \tag{7.4}$$

$$\text{Timed-KBT(column)} = \text{KBT}_{t_{column}}\text{(column)} \tag{7.5}$$

As mentioned above, a fusion strategy needs to assign a score to a candidate value given a target slot. Given e.g. that Timed-KBT assigned the column of a candidate value a temporal scope that equals the target temporal scope of a certain slot, the score given to that candidate value for that slot equals the Timed-KBT score of the column as computed in Formula 7.5. If the temporal scopes are not equal, but close, we use the neighboring scope estimation as described below.

In Formula 7.4, T is a set of temporal scopes. The first Timed-KBT-based strategy we introduced (`TKBT`), derives this set from all the temporal scopes present in the knowledge base. T can however also be restricted to a smaller set.

Restricting Timed-KBT Using Timestamp Information

We implement a second Timed-KBT-based fusion strategy. Unlike in `TKBT`, where T in Formula 7.4 is the set of all temporal scopes that exist in the knowledge base, in `TKBT-Rstrict`, we restrict T to temporal scopes extracted from timestamps in the table and its context.

This would mean that in the specific case of the table in Figure 7.1, that the first population column cannot be assigned a scope of 2015. `TKBT-Rstrict` is therefore likely to cause a drop in recall, which could however be offset if precision increases favorably.

`TKBT-Rstrict` also makes use of neighboring scope estimation. We introduce neighboring scope estimation below.

Neighboring Scope Estimation

As we assign only one temporal scope to a table column, its values are only used for the fusion of slots with that assigned scope. Assuming that temporal scopes are years and that facts of certain properties do not completely change annually, it would make sense to allow values that were assigned one scope, to be used to fuse facts for neighboring scopes. Given for example Figure 7.1 and that the first population column was assigned the scope 2015, its values can be used as candidates for slots with scope 2014, with an adapted score computed as

$$\text{estimatedScore} = \begin{cases} \text{Timed-KBT} - \text{diff} \times \alpha & \text{diff} <= \text{maxDiff} \\ 0 & \text{diff} > \text{maxDiff} \end{cases} \tag{7.6}$$

, where *Timed-KBT* equals the score of the column as computed in Formula 7.5, *diff* equals the absolute difference between the assigned year and the target year, while *maxDiff* equals the maximum difference allowed. This maximum difference is learned per class-property combination from 0 to 10. We define α to be $0.1 = 1/10$.

7.4 Results and Findings

This section presents and discusses the overall results of the implemented fusion strategies. We will also discuss the impact of timestamp information and neighborhood scope estimation.

7.4.1 Fusion Results

Table 7.4 shows the average performance by fusion strategy. We can first of all see that `KBT` outperforms `Voting` by a large margin for F_1 and $F_{0.25}$. Additionally,

Table 7.4: Average performance by fusion strategy.

	Optimized for $F_{1.0}$				Optimized for $F_{0.25}$			
Strategy	**P**	**R**	$\mathbf{F_{1.0}}$	$\boldsymbol{\sigma_{F_{1.0}}}$	**P**	**R**	$\mathbf{F_{0.25}}$	$\boldsymbol{\sigma_{F_{0.25}}}$
`Voting`	0.14	0.32	0.19	0.18	0.14	0.32	0.15	0.14
`KBT`	0.31	0.46	0.37	0.29	0.32	0.43	0.32	0.27
`TT-Weighting`	0.33	0.45	0.37	0.30	0.45	0.25	0.42	0.30
`TKBT`	0.49	0.47	0.47	0.23	0.54	0.34	0.51	0.22
`TKBT-Rstrict`	0.52	0.38	0.43	0.20	0.66	0.23	0.57	0.23

`KBT` has the highest recall for $F_{0.25}$ and among the highest for F_1, which means that any strategy that outperforms `KBT`, does so mainly by increasing precision.

`TT-Weighting` is the approach introduced in the previous chapter, i.e. the fusion strategy titled `KbtTsProp` in Table 6.3. When evaluating using a β of 1.0, the difference between `KBT` and `TT-Weighting` is minimal, however in the case of a β of 0.25, there is an increase in $F_{0.25}$ and a larger increase in precision when comparing `KBT` to `TT-Weighting`. This shows that the scores computed by `TT-Weighting` are relevant to the fusion precision, but also that they are only effective when a drop in recall is acceptable. The results could indicate that some timestamps are relevant to the data in the table and that timestamp locations have certain relationships with properties, which is the main assumption behind `TT-Weighting`. The fact that this possible relationship only has a positive effect when assigning recall a lower weight than precision, possibly indicates that the way we measure maximum recall is too strict.

Both Timed-KBT-based approaches show an increase in performance when compared to other methods for both F_1 and $F_{0.25}$. `TKBT` for F_1 even has the highest recall. Through this we can infer that a temporal knowledge base can successfully be used to estimate temporal scopes for web table data.

`TKBT-Rstrict` outperforms `TKBT` for $F_{0.25}$. While its increase in precision comes at the cost of recall, the decline happens at a favorable rate. `TKBT` is unable to yield a higher precision for $F_{0.25}$, e.g. by increasing the threshold, without a performance drop, whereas the precision increase for `TKBT-Rstrict` is large enough to compensate for the drop in recall. This shows that timestamps from the tables and their context can be relevant to the data and that `TKBT-Rstrict` is able to use them effectively.

7.4.2 Impact of Using Timestamp Information

There are two fusion strategies that make use of timestamp information. These are `TKBT-Rstrict` and `TT-Weighting`. Both strategies, generally speaking, come at a cost of recall. When optimizing and testing for $F_{1.0}$, this can not be offset by `TT-Weighting` to achieve a higher performance than `KBT`. Similarly,

Table 7.5: Effect of neighboring scope estimation on fusion performance.

		No estimation			With estimation			Change		
Method	**β**	**P**	**R**	**F_β**	**P**	**R**	**F_β**	**ΔP**	**ΔR**	**ΔF_β**
TKBT	1.0	.290	.299	.233	.495	.467	.471	+.204	+.168	+.237
TKBT-Rstrict	1.0	.355	.299	.239	.517	.380	.430	+.162	+.081	+.190
TKBT	0.25	.427	.131	.367	.541	.340	.513	+.114	+.209	+.146
TKBT-Rstrict	0.25	.567	.121	.444	.658	.231	.571	+.092	+.110	+.127

while TKBT-Rstrict can outperform KBT, it can not outperform TKBT.

This however changes when we care more about precision than recall, e.g. when optimizing and testing for $F_{0.25}$. In that case the drop in recall is favorably offset by a rise in precision. TT-Weighting can outperform KBT, while TKBT-Rstrict outperforms TKBT.

These findings indicate that timestamp information is too sparse for a high recall fusion strategy, however useful enough to enable an effective precision-oriented fusion strategy.

7.4.3 Impact of Neighboring Scope Estimation

Table 7.5 shows that incorporating neighboring scope estimation into the Timed-KBT-based strategies had a large positive effect on fusion performance. The relative increase was more than 40% for F_1 and 20% for $F_{0.25}$ for both strategies. A rather unexpected result is that neighboring scope estimation increases precision in addition to recall. The reason for the increase in precision is likely that matched values of neighboring temporal scopes with a high score can outweigh low-scoring and probably incorrect values that were assigned the target scope.

7.5 Related Work and Discussion

The related work has been extensively discussed in Section 6.5 and summarized in Table 6.6 of the previous chapter. Table 6.6 shows how Timed-KBT is one of the only three approaches that allow targeted fusion using a class-agnostic approach with focus on temporal knowledge. Unlike TT-Weighting and InfoGather+ however, Timed-KBT is not dependent on the existence of timestamp information to estimate temporal scope annotations. This means that Timed-KBT can be applied to a much broader range of applications, even those that completely lack temporal scopes and timestamp information. For this, Timed-KBT requires a temporal knowledge base. In the context of slot filling for time-dependent data, the presence of such a temporal knowledge base is likely.

When compared to the overall average performance achieved by InfoGather+ [Zhang and Chakrabarti, 2013] (see Table 6.9), it seems that Timed-KBT lacks behind in precision. However, the results are not directly comparable, as they are

Table 7.6: Fusion performance of Timed-KBT for the property Population of the class City compared to reported performances for the InfoGather+ approach [Zhang and Chakrabarti, 2013].

	P	Correct Facts
`Voting`	0.19	1,172
`KBT`	0.37	1,664
`TKBT`	0.66	1,518
`TKBT` (Opt. for $F_{0.25}$)	0.71	979
`TKBT-Rstrict` (Opt. for $F_{0.25}$)	0.84	774
InfoGather+ Baseline	0.08	31
InfoGather+	0.87	303

performed on different datasets. In the case of InfoGather+, these datasets contain mostly head entities, and the experiments are limited by query size.

We can however compare performances for the property *Population* of the class *City*, as both, the evaluation in this chapter and in the paper by Zhang and Chakrabarti run experiments for this property. Table 7.6 compares reported performances. As Zhang and Chakrabarti do not measure recall, we will focus on comparing performances based on precision.

We find that our approach somewhat lacks behind in regard to precision. However, this can be explained by the fact, that our testing set is much larger. Overall, we include 11,372 cities in our ground truth, where 5,884 triples within the property *Population* have an actual candidate among the matched web table values. Zhang and Chakrabarti on the other hand use a dataset with just the 600 largest cities, i.e. containing mostly head entities, and have query sizes of just 400 entities.

As we tune most of our fusion strategies for F_1, where recall and precision are weighted equally, we might lack behind in precision, but are able to fuse a considerably larger number of facts. The strategy `TKBT` e.g. fuses 1518 correct facts, five times more than the 303 correct facts returned by InfoGather+. When we tune our performance for $F_{0.25}$, we achieve with `TKBT-Rstrict` a very similar precision, lacking behind in just 3 percentage points. However, we are able to fuse 2.5 times as many correct facts. As such, we believe that our approach is more suitable for knowledge base augmentation, where enriching the knowledge base with a large number of facts is an important aspect.

Timed-KBT could be extended and improved in two ways. First, temporal scopes could be estimated for other than just the column of a web table. There exist tables where every row corresponds to a different temporal scope, e.g. a table that lists the teams an athlete has played for over a year. In other cases, the whole table might correspond to a single temporal scope. An approach that estimates temporal scopes per row and table in addition to by column could further improve performance.

Second, the lessons learned from the previous chapter could be integrated with Timed-KBT. When restricting temporal scopes, we could take into consideration the timestamp types, and their relevance given a certain property of the knowledge base schema. Alternatively, we can consider the output of temporal scope propagation during restriction, which could have a positive effect on recall.

7.6 Summary

In this chapter, we introduced Timed-KBT, an approach that enables time-aware fusion of time-dependent web table data by generating missing temporal scopes using a temporal knowledge base. We test Timed-KBT using a large web table corpus on the task of targeted slot filling. As a target knowledge base, we make use of the Time-Dependent Ground Truth, a dataset with time-dependent data that we created and published.

We find that Timed-KBT is able to assign useful explicit temporal scopes to web table data. We also find that using scores estimated by Timed-KBT for fusing time-dependent web table data yields a performance increase when compared to other fusion methods.

We then utilized timestamps extracted from the web tables and their contexts as a restriction for candidate temporal scopes assignable by Timed-KBT. This approach yields a higher performance in regard to precision, and therefore a possibly more favorable performance for knowledge base augmentation. We conclude that timestamps in the table and its context are useful for a precision-oriented time-aware fusion strategy. Finally, we also show that explicitly assigned temporal scopes are highly useful for fusing data for neighboring temporal scopes.

Overall, we demonstrate that a temporal knowledge base can be used to estimate missing temporal scope annotations for web table data. We also show that with Timed-KBT, we are able to perform knowledge base augmentation from web table data for historic facts in addition to current facts. Our findings enable the time-aware fusion of time-dependent web data even if that data lacks timestamp information completely.

Part III

Long-Tail Entity Extraction

Chapter 8

The Long-Tail Entity Extraction Pipeline

Entity expansion is a knowledge base augmentation task that involves adding new and formerly unknown entities, including their descriptions, to the knowledge base. As such, entity expansion from web tables consists of two primary subtasks, first, identifying entities of a specific class that are not yet part of a knowledge base, and second, compiling descriptions of these new entities according to the schema of the knowledge base. For identifying new entities, it must be taken into consideration that classes in knowledge bases are very large, so that many different entities within a class could share surface forms, i.e. are homonyms. As such, we need to disambiguate entities by more than just their label.

There are two tasks related to or share similarities to the task of entity expansion: set expansion and emerging entity detection. Set expansion methods [Wang et al., 2015,Pantel et al., 2009,Wang and Cohen, 2007] complete a certain set of entities e.g. by using web data in the form of lists or tables. However, these methods by definition focus solely on determining new entities that are missing to complete a certain set and are not concerned with compiling structured descriptions of those entities according to a schema of a knowledge base. Additionally, set expansion methods disambiguate solely by names, as they attempt to complete a set of entity labels. This is because the sets evaluated in set expansion are small enough that homonyms are not an issue, unlike a class in a knowledge base.

Emerging entity detection [Hoffart et al., 2014, Färber et al., 2016, Derczynski et al., 2017] is an NLP task that determines if a certain entity mention in text specifically refers to a long-tail entity that is not yet part of a knowledge base. Existing approaches however do not attempt to link mentions of the same emerging entity together, and as such can not identify the exact number of long-tail entities. They are also unable to create descriptions for those entities. Therefore, they are unable to perform entity expansion.

As a result, no viable methods for entity expansion from web tables exist. In the work presented in this chapter, we close this gap by introducing and evaluating

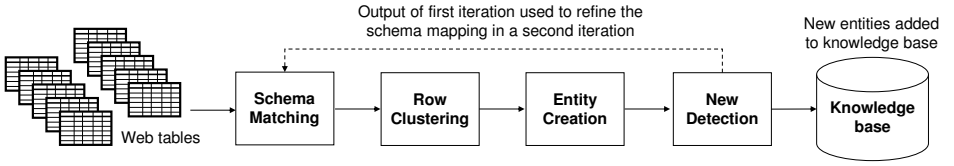

Figure 8.1: Overview of the Long-Tail Entity Extraction Pipeline.

the first system that is able to extract formerly unknown long-tail entities with their descriptions from a corpus of relational web tables.

The contributions of this chapter are:

- **The Long-Tail Entity Extraction Pipeline:** we introduce the first system that is able to identify new entities and generate their descriptions using a corpus of relational web tables for the purpose of entity expansion. Existing approaches with similar tasks are either unable to create descriptions (set expansion and emerging entity detection), disambiguate entities solely by their label (set expansion), or are unable to derive the exact number of unique long-tail entities (emerging entity detection).
- **The Web Tables for Long-Tail Entity Extraction (T4LTE) dataset**: we introduce the first gold standard for the task of entity expansion using web tables. This gold standard can be used for both training and evaluation. Entities marked as new in the gold standard are actually non-existing in DBpedia, which is the target knowledge base to be extended within the context of T4LTE. As such, T4LTE contains annotations for actual long-tail entities. When employed with our evaluation metrics and cross-validation folds, T4LTE acts as a benchmark for the task of entity expansion from web tables.
- We evaluate our pipeline using the T4LTE gold standard. We suggest, implement, and test alternative approaches for various components of the pipeline and extensively describe lessons learned.

Figure 8.1 gives an overview of the overall process performed by our pipeline. It consists of four components which are executed in two iterations. We first apply schema matching methods to match web tables and their attribute columns to classes and properties in the knowledge base schema. Second, a row clustering method identifies rows that describe the same entity. From these row clusters, the entity creation component creates entity descriptions according to the schema of the knowledge base. Finally, the new detection component determines whether an entity is new, by comparing it to all instances of existing entities in the knowledge base. We iterate over the pipeline a second time, using row clusters and entity-to-instance correspondences derived from the first run in order to refine the schema mapping using duplicate-based matching. After the second run of the pipeline, entities identified as new are added to the knowledge base.

We publish both our code and our gold standard, including the used cross-validation folds. As such, all resources used within this chapter are publicly available, allowing the replication of our work.[1]

This chapter is structured as follows. First, we describe our experimental setup, including the design and contents of the T4LTE dataset. Section 8.2 introduces our methodology by describing the individual components of the overall pipeline. For most components, we suggest and evaluate alternative implementations and discuss lessons learned. Section 8.3 presents the overall performance of the pipeline for the task of entity expansion. We evaluate both, how accurately we found new entities and how accurately we created their descriptions. Section 8.4 compares our system to the related work. We summarize this chapter in the final section.

The work presented in this chapter has previously been published in [Oulabi and Bizer, 2019a]. However, this chapter contains the following additions and changes: (1) an extended class-specific feature set for the two features `ATTRIBUTE` *and* `IMPLICIT_ATT`*, (2) additional similarity metrics for comparing labels, and (3) the use of nested cross-validation instead of out-of-bag error for hyperparameter tuning. The changes have a positive increase on performance, especially in comparison to related work.*

8.1 Experimental Setup

The experimental setup of this chapter, including that of Chapter 10, revolves around the T4LTE (Web Tables for Long-Tail Entity Extraction) dataset. It is a gold standard we created specifically for the task of entity expansion from web tables. It was created with DBpedia as a target knowledge base, and as a result, entities annotated as new in T4LTE are actually missing in DBpedia.

This section introduces the T4LTE dataset, including its design and statistics. This section also describes how we use T4LTE to evaluate our pipeline.

T4LTE has been made available publicly.[2] With the exact evaluation metrics used in this work, and the specific cross-validation folds, which we also provide, T4LTE acts a benchmark for entity expansion from web tables.

8.1.1 The T4LTE Gold Standard

We created T4LTE specifically for this research as a gold standard for evaluating the extraction of long-tail entities from web tables. It fulfills three tasks. First, it allows measuring the performance of long-tail entity extraction, including recall. By focusing on recall, we can evaluate how far methods retrieve all long-tail entities, instead of just evaluating whether retrieved entities are correct. Secondly, T4LTE enables the automatic evaluation of implemented methods. Finally, the dataset can also be used as supervision to train long-tail entity extraction methods.

[1] http://data.dws.informatik.uni-mannheim.de/expansion/LTEE/

[2] http://webdatacommons.org/T4LTE/

Following, we describe how we created the T4LTE gold standard, including what target knowledge base we employed, which classes we included, and how we extracted web tables from which web table corpus to be included in T4LTE. Finally, we will also provide statistics about T4LTE.

Target Knowledge Base

We employ DBpedia [Lehmann et al., 2015] as the target knowledge base to be extended. It is extracted from Wikipedia, and as a result the covered entities are limited to those identified as notable by the Wikipedia community. We use the 2014 version of DBpedia, as it has been used in related work [Ritze et al., 2015, Ritze et al., 2016, Oulabi et al., 2016, Oulabi and Bizer, 2017] and its release date is also closer to the extraction of the web table corpus from which we created T4LTE.

Class Selection

From DBpedia, we selected three classes for which we built the dataset. This selection was broadly based on two criteria:

- **Versatility:** the chosen classes must be from different first-level classes, which in DBpedia are Species, Work, Agent and Place. This allows us to ensure that the included classes cover domains that are sufficiently diverse.
- **Name conflict likelihood:** we utilize the labels of entities in the knowledge base to measure the potential for homonyms given a certain class. Classes with a higher relative occurrence of homonyms were preferred. This was done to ensure that we select the classes for which the task of entity expansion is likely more difficult than easy.

Based on this approach we chose the following three classes: (1) GridironFootballPlayer (GF-Player), (2) Song and (3) Settlement. The class Song also includes all entities of the class Single.

Source Web Table Corpus

We extract the tables included in T4LTE from the English-language relational tables subset of the 2012 WDC web table corpus[3]. The set consists of 91.8 million tables. Table 8.1 below gives an overview of the general characteristics of tables in the corpus. We can see that the majority of tables are rather short, with an average of 10.4 rows and a median of 2, whereas the average and median number of columns are 3.5 and 3. As a result, a table on average describes 10 entities with 30 values, which likely is a sufficient size to understand a table and be potentially useful for finding new entities and their descriptions. In Chapter 5, we have profiled the potential of the same corpus for the task of slot filling.

[3] http://webdatacommons.org/webtables/#toc3

Table 8.1: Characteristics of the employed 2012 WDC web table corpus.

	Average	Median	Min	Max
Rows	10.37	2	1	35,640
Columns	3.48	3	2	713

Web Tables Selection

For the gold standard we had to select a certain number of tables per class to annotate. We first matched tables to classes in the knowledge base using the T2K framework (see Section 4.3). Tables were then selected for each class individually.

We attempted to include tables with entities of varying degrees of popularity. For this, we first divided the entities in the knowledge base into quartiles of popularity using the indegree count based on a dataset of Wikipedia page links.[4] We then select three entities per quartile, overall 12 per class.

Using the 12 selected entities, we searched for tables that contain the labels of those entities. We then looked for labels that co-occur in those tables, but which do not have any entity with a matching label in the knowledge base. This was done, to ensure that there are enough long-tail entities in the selected tables.

For each of those labels and the labels of the 12 selected entities, we then extracted up to 15 web tables. During this extraction, we additionally ensured that few tables are chosen from the same PLD, and that tables have a variety in their attribute columns.

Labeling Process

We did not label all tables and especially not all rows, but rather aimed to label new entities and entities with conflicting names (homonyms).

We first annotated rows that describe the same entity to create row clusters. We then annotated if the entity of a cluster already exists in DBpedia, or whether it describes a new entity. For existing entities, we additionally annotated clusters with the URI of the corresponding entity instance in DBpedia.

We annotated all web table columns with corresponding properties from the knowledge base schema, if a matching property exists. These attribute-to-property correspondences allow us to identify how many candidate values exist for each triple, i.e. a certain combination of entity and property. For all triples which have candidate values, we finally annotated the correct fact, and additionally annotated whether that correct fact exists among the candidate values. This allows us to precisely measure the precision and recall of descriptions created for new entities.

[4] https://wiki.dbpedia.org/Downloads2014#owikipedia-pagelinks, provided along with DBpedia.

Table 8.2: Statistics for the T4LTE gold standard.

Class	GF-Player	Song	Settlement	Sum
Tables	192	152	188	**532**
Attribute columns	572	248	162	**982**
Rows	358	195	413	**966**
Existing clusters	80	34	51	**165**
New clusters	17	63	23	**103**
Matched values	1,177	428	487	**2,092**
Value groups (facts)	460	231	152	**843**
Correct value present	436	212	124	**772**

Statistics

Table 8.2 provides an overview of the number of annotations in T4LTE. The first three rows show the number of table, attribute and row annotations. On average, we have 1.85 attribute annotations per table, not counting the entity label column. The two following rows show the number of annotated clusters. We overall annotated 268 clusters, of which 103 are new, and where each cluster has on average 3.60 rows and 7.81 matched values.[5] The number of matched values is given in the next row and equals the number of cells, which are both part of a row within an annotated cluster and a column with an attribute correspondence. Value groups is the number of unique triples the matched values correspond to, for each of which we have also annotated the correct fact. The last row shows how many of those groups actually contain the correct value. Per cluster we can derive on average 3.15 facts, for 92% of which the correct value is present.

8.1.2 Cross-Validation Splitting and Evaluation Metrics

As we use the T4LTE gold standard for learning and testing, we apply three-fold cross-validation. For this, we split by cluster, so that the rows of one cluster are always fully included in one fold. We ensured that we evenly split both new clusters and homonym groups. A homonym group is a group of clusters with highly similar labels. All clusters of a homonym group were always placed in one fold.

Using T4LTE, we evaluate the performance of five aspects of our pipeline: (1) schema matching, (2) row clustering, (3) new detection, (4) end-to-end entity expansion, and (5) created descriptions of new entities. The evaluation metrics are described in the sections where we also present the results. For schema matching, row clustering, and new detection, we evaluate the performance within the methodology section, as we test alternative implementations for each component.

[5] For the class Song, we additionally annotated 17 existing clusters. These are included for training purposes only, and are not fully labeled, as they are missing fact annotations.

8.2 Methodology

In this section, we present our long-tail entity extraction system by describing the individual components of the pipeline. As shown in Figure 8.1, the pipeline begins with the web tables and ends with new entities being added to the knowledge base. In between, the pipeline consists of four components: schema matching, row clustering, entity creation and new detection. For all components, except entity creation, we will suggest and evaluate alternative approaches throughout this section.

We iterate over the pipeline twice. During the second run, we utilize the output of the row clustering and the new detection to generate a refined schema mapping using duplicate-based matching. The attribute-to-property correspondences derived by the schema matching are important, because they allow us to extract for each row a set of values which correspond to the schema of the knowledge base. These values are utilized as similarity features by the row clustering and new detection components with a positive impact on performance. More importantly, these values are required to create descriptions for new entities.

During the schema matching phase, we also match each table to a class in the knowledge base. Afterwards we run the remainder of the pipeline for each class separately.

8.2.1 Schema Matching

The first step in the pipeline is to create a mapping between the schemata of the individual web tables and the schema of the knowledge base. As the web tables have heterogeneous schemata, this task is non-trivial. Overall, there are four steps necessary: (1) data type detection, (2) entity label column detection, (3) table-to-class matching and (4) attribute-to-property matching. Regarding the first three steps, we have provided a more detailed description in Section 4.3 of Chapter 4.

Data Type Detection

Throughout our pipeline we utilize a number of data types to type individual values, facts, attribute columns and knowledge base properties. Each type has a corresponding similarity function and an equivalence threshold, which are used to determine if two compared values of that type are equal. We employ overall six data types:

- **String:** string, where two strings do not have to be exactly equal to be similar, e.g. label of an entity.
- **Entity Reference:** reference to an entity, e.g. team of an athlete or musical artist of a song.
- **Date:** date with two possible granularities: year or specific day, e.g. release year of song, or birth date of a person.

- **Numeric:** numeric quantity, where numeric closeness has a semantic relevance, e.g. population of a settlement.
- **Nominal String:** string, where two strings are either completely equal or otherwise unequal, e.g. ISO code of a country.
- **Nominal Integer:** integer, where numbers close to each other are not semantically related. This include e.g. numbers or draft rounds of athletes. Nominal integers are similar to nominal strings. However, typing nominal integers in addition to nominal strings allows some components, especially the attribute-to-property matcher, to use methods tailored for this type.

We run the data type detection algorithm of the T2K matching framework [Ritze et al., 2015], which detects data types using manually defined regular expressions. It assigns to each table column one of the following basic types: string, date and numeric. Detecting types with a higher granularity requires an understanding of the actual semantics of the data in the column. This is done by the attribute-to-property matcher we suggest below. The types are assigned to a column after the column attribute has successfully been matched to a knowledge base property.

Entity Label Column Detection

For each table, we assign one column as the entity label column, which contains natural language labels for the entities described in the table. This is done by finding the column with the data type string and the highest number of unique values. In case there is a tie between multiple columns, we choose the column that is furthest to the left. This is the approach employed by T2K.

Table-to-Class Matching

For table-to-class matching, we again utilize the approach that is part of the T2K framework. It performs both row-to-instance and attribute-to-property matching iteratively to find the class of a table. We describe their approach in more details in Subsection 4.3.1 of Chapter 4. The approach was evaluated by Ritze et al., where authors find that it can achieve an F1 score of 0.97 on a gold standard of web tables [Ritze et al., 2015].

Attribute-to-Property Matching

Our attribute-to-property matching approach consists of three steps. We first select candidate properties from the knowledge base schema based on data types. For string columns, we choose all properties with types entity reference, nominal string and string, for numeric columns, we choose properties with types numeric and nominal integer and finally for date columns, we choose properties with types date, numeric and nominal integer as candidates. After successful matching, the data

Table 8.3: Attribute-to-property matching performance by pipeline iteration.

Iteration	P	R	F1
First	0.929	0.608	0.735
Second	0.924	0.916	0.920
Third	0.929	0.916	0.922

type of the column is changed to the data type of the matched property and the values of the column are accordingly normalized.

Secondly, we use various matchers, described further below, to compute matching scores. Given a candidate knowledge base property, a matcher finds a score from 0.0 to 1.0 that measures the likelihood that the attribute described by a column matches the property. Scores of multiple matchers are aggregated using a weighted average, where weights are learned for each class individually.

Finally, we utilize thresholds on the aggregated scores to determine if a certain candidate property matches an attribute. The thresholds are also learned per property of the knowledge base schema. An attribute is matched to a property if it is both, a property that achieves a score above the property-specific threshold, and the property with the highest aggregated score.

Overall, we implement five matchers, three of which exploit the knowledge base. **`KB-Overlap`** computes the proportion of values in the attribute that generally fit the candidate property in the knowledge base. **`KB-Label`** compares the label in the attribute header row to the labels of the candidate property in the knowledge base. **`KB-Duplicate`** assumes the existence of row-to-instance properties generated by a previous iteration of the pipeline. It computes the proportion of values in a column that are equal to the fact present for a candidate property in the knowledge base for the entity instance matched a value's row.

We further implement two matchers that exploit the web table corpus. For this, we first match attributes using the above described matchers that exploit the knowledge base for a preliminary mapping. We then rerun the matching using two additional matchers that exploit this preliminary mapping and the web table corpus. **`WT-Label`** utilizes the column headers of columns matched in the preliminary run, to derive label-to-property scores, where the score represents the likelihood that an attribute with a certain header row label corresponds to a certain candidate property of the knowledge base. **`WT-Duplicate`** assumes the presence of row clusters generated by the row clustering component of a previous run of the pipeline. Using the clusters, we know which values in web tables match the same entity. Using the preliminary mappings, we additionally know which values match which property. `WT-Duplicate` measures and returns the proportion of values in a column, for which an equal value exists in the corpus, that is matched to same entity and property.

Table 8.3 shows by iteration the performance of an attribute-to-property match-

ing method that aggregates all the matchers described above. The duplicate-based methods are not included in the first iteration, as they require output from the other pipeline components. We evaluated the methods on the attribute annotations in the gold standard using three-fold cross validation.

From the table, we can see that a second iteration and the utilization of the output of the row clustering and the new detection components for duplicate-based matching have a large positive effect on schema matching performance. The table also shows that a third iteration has only a marginal positive effect, so that two iterations suffice.

To determine the usefulness of each individual matcher, we evaluate the relative weight assigned to it in the aggregated method of the second iteration. As we learn weights per class, the following weights are averages for all three classes. The duplicate-based matchers have a combined weight of 0.43, where `KB-Duplicate` with a weight of 0.25 is more important. The label-based matchers achieve a higher combined weight of 0.46 where the `WT-Label` with a weight of 0.25 is very effective. Finally, the `KB-Overlap` method is the least important method with a weight of 0.10. The distribution of weights for the individual classes were similar to the here mentioned averages.

From the weights we can first of all see, that the label is quite an effective method for schema matching. The labels of the web table corpus are possibly more important, because the corpus includes a larger diversity of attribute labels than the knowledge base. On the other hand, for duplicate-based matching, it makes sense the method that uses facts in the knowledge base for matching is more important, as facts in the knowledge base are likely more accurate.

Overall, we also find that the most effective approach is one that combines various matchers and thereby exploits the highest number of individual signals for schema matching.

8.2.2 Row Clustering

After matching tables to classes and table columns to properties of the knowledge base, we cluster rows that describe the same entity together. This step is especially important, as it reveals the overall number of unique entities described in the tables.

Our row clustering methods consist of two parts: (1) a method that computes row similarity, which measures the likelihood that two rows describe the same entity, and (2) a clustering algorithm, that utilizes the row similarities to create the actual row clusters.

Clustering Algorithm

In the context of this work, a required feature of the clustering algorithm is its ability to determine the correct number of clusters itself. In case of a perfect clustering, this number would correspond to the exact number of unique entities described by the rows of all tables.

Correlation clustering approaches [Bansal et al., 2004, Demaine et al., 2006, Ailon et al., 2008, Bonchi et al., 2014] fulfill this requirement. Clustering here is viewed as an optimization problem that aims to find the optimal partitioning of a set of vertices. The optimal partition maximizes a fitness function that aggregates similarities within clusters and dissimilarities between clusters.

Due to the large number of tables and rows in a web table corpus,[6] we need clustering methods that scale. As correlation clustering approaches try to find a globally optimum solution, they do not scale for the task at hand. We therefore look at greedy or local search clustering algorithms, that are able to approximately determine the optimum number of clusters, while still remaining scalable.

ElsnerVOTE [Elsner and Charniak, 2008, Elsner and Schudy, 2009] is a greedy clustering algorithm. It attempts to sequentially assign each vertex (i.e. a row) to the optimum cluster, by summing the weights of the edges between the vertex and all the vertices within an already created cluster. The vertex is assigned to the cluster with the highest aggregated score, if the score is positive (this assumes a similarity score normalized from –1.0 to +1.0). If all aggregated scores are negative, the vertex can not be assigned to any cluster and a new cluster is created with that vertex in it. As every vertex assignment or cluster creation would only maximize the fitness function at the time of the operation, this possibly does not result in an optimal solution.

Kernighan-Lin with Joins (KLj) [Keuper et al., 2015] is an extended version of the Kernigen-Lin [Kernighan and Lin, 1970] heuristic algorithm. KLj improves an existing preliminary clustering, by comparing two clusters and attempting to move individual vertices between those clusters or merging the clusters fully, in case this has a positive increase on the fitness function locally. Similarly, each cluster is compared with an empty set to find whether splitting a cluster increases the fitness function locally. The operations are repeated until no further operation can increase the fitness function. As such, unlike ElsnerVOTE, KLj finds a local optimum.

We use in this work the greedy ElsnerVOTE as a preliminary clustering step. To achieve further scalability, we perform the row assignment in parallel instead of sequentially. While this parallelization is much faster, it can result in additional non-optimal assignments during clustering. We therefore run the the KLj algorithm on the output of parallelized ElsnerVOTE. The KLj algorithm would compare all created clusters and correct possible mistake to reach a local optimum.

Combining a greedy with a local search algorithm is also suggested by the related work on clustering [Keuper et al., 2015, Elsner and Schudy, 2009]. As a result, we are able to quickly build a preliminary clustering with complete parallelization, while with a second step, we ensure that clustering quality is still high and at least a local optimum has been reached. Scalability is further ensured by the blocking approach, which we describe further below.

[6] While in this chapter, we evaluate our pipeline only on the gold standard, in the next chapter we use it on the whole web table corpus. For the class Song, this means clustering more than two million rows.

An alternative greedy algorithm to ElsnerVOTE would be *Greedy Additive Edge Contraction (GAEC)*. GAEC works by joining per iteration two clusters, starting with each vertex as its own cluster. The two clusters to be joined are chosen such that joining them has the largest positive increase on the fitness function. We prefer ElsnerVOTE as it lends itself well to parallelization. With ElsnerVOTE, we parallelize by row, so that we attempt to assign more than one row at the same time, given the most recent state of clusters already created.

Row-To-Row Similarity Features

As mentioned above, we compute row-to-row similarity scores that measure the likelihood that two rows refer to the same entity. The scores are then used by the clustering algorithm to cluster rows.

To compute these scores, we first exploit a set of features and corresponding metrics. This allows us to generate large number of individual similarity scores. These scores are then aggregated to yield a single row-to-row similarity score.

Depending on the feature, other input, e.g. from the knowledge base or the web table corpus, might be utilized. Overall, our implemented features can be categorized into six different types:

- **`LABEL`.** We use the values within the entity label column of a table to derive a label for each row. We then compare these labels to derive similarity scores using three different approaches. First, we compare using the Generalized Jaccard similarity measure, using Levenshtein as the inner similarity [Doan et al., 2012b]. Secondly, we use the overlap similarity measure, where the intersection is also computed using Levenstein. Finally, we again use the Generalized Jaccard similarity, however this time additionally checking for surface forms when comparing labels. These surface forms are extracted from various label properties within DBpedia[7], Wikipedia redirects[8], and a large surface forms dataset, which itself is extracted from Wikipedia and the Common Crawl [Bryl et al., 2016].

- **`BOW`.** For each row we create a bag-of-words binary term vector that contains the terms that occur in all cells of a row. For this, cell values are cleaned, normalized and tokenized. To compare two rows, we compute the cosine similarity of their vectors.

- **`PHI`.** This approach allows us to compare two rows by comparing their tables. It derives a similarity between two tables using the phi correlation of row labels. The phi correlation of two labels is computed as following:

[7] These include foaf:name, foaf:nick, dbo:formername, dbo:longName, dbo:birthname, dbo:alias, and rdfs:label.

[8] https://wiki.dbpedia.org/Downloads2014#oredirects, provided along with DBpedia.

$$phi(x,y) = \frac{n \times n_{xy} - n_x \times n_y}{\sqrt{n_x \times n_y \times (n - n_x) \times (n - n_y)}}, \tag{8.1}$$

where n = total number of unique labels,

n_{ab} = occurrence of labels a and b in same table,

n_a = occurrence of label a in a table.

For each label, we compute a vector that consists of all labels in the corpus and contains the phi correlation with those labels. We then create such a vector for each table, by averaging the vectors of the table's row labels. We attempt to reflect with this vector semantic information about all rows described in a given table. When comparing two rows, we return the cosine similarity of the vectors of their tables.

- **ATTRIBUTE.** Using the attribute-to-property correspondences, we can derive a set of values for each row, where each value is matched to a property of the knowledge base schema. This allows us to apply data-type-specific similarity functions when comparing the values of two rows. This is possible because we only compare two values matched to and normalized for the same knowledge base property, whereas for example in BOW, we compare tokens without considering if they semantically match.

  We return for a row pair two scores for each property from the knowledge base schema with a range from 0.0 to 1.0. One score measures the confidence of the pair having equal values given that property, the other of the pair having unequal values. If no overlap between the attribute columns exist, both scores are returned as –1.0.

  The number of similarity scores generated therefore depends on the number of properties that exist in the schema of the knowledge base given a certain class.[9] This means that the exact feature set is therefore class-specific.

- **IMPLICIT_ATT.** Many tables have rows that describe entities that are similar, e.g. cities in Germany or athletes drafted in 2010. This information is not stated explicitly in any of the row cells. Using the following approach, we attempt to derive for a table implicit property-value combinations that apply to all entities described by the table. We can then use these implicit property-value combinations to compare rows with each other.

  We first use the row labels to find candidate instances in the knowledge base for all rows of a table. Using these candidate instance, we then derive all property-value combinations that exist for at least one candidate in the knowledge base. For each property-value combination, we then derive a score for the whole table, which equals the proportion of rows for which

[9] In our experiments, we only consider properties that have a density of at least 30% in DBpedia.

we could derive this combination. We keep only combinations with a score above a certain threshold, which are then assigned to the table as implicit attributes with their scores included as confidences.

Given two rows, we compare the implicit attributes of one row with overlapping implicit attributes and column attributes of the other row and vice versa. Again, when comparing two rows we compute a similarity score for each property in the knowledge base schema, given a certain class. This means that the exact feature set differs per class and is therefore class-specific.

- **SAME_TABLE.** This feature builds on the observation that rows in a single table usually describe different entities. We assign two rows of the same table a similarity of 0.0, otherwise 1.0.

Class-Specific and Class-Agnostic Feature Sets

The features of type `ATTRIBUTE` and `IMPLICIT_ATT` are class-specific. In the published work on the LTEE Pipeline [Oulabi and Bizer, 2019a], we used a version of those features that were class-agnostic, i.e. the set of features is the same for any class. We use these class-agnostic versions in Chapter 10 for building a class-agnostic unsupervised matching model. To allow comparability, we will additionally describe how we create these class-agnostic versions of those features and compare their effect on performance in this chapter.

The class-specific versions of `ATTRIBUTE` and `IMPLICIT_ATT` yield two scores with a range from 0.0 to 1.0 for every single property of a class. One score describes the likelihood that, given that property, the values of a compared pair are equal, while the other describes the likelihood that they are unequal. Scores of –1.0 mean that there exists no overlap between the values of the compared pair for that property. We compute one single class-agnostic score for each feature by averaging all the scores that describe the likelihood that the values are equal, and for which overlap exists, ignoring the scores that measure if two values are unequal.

These class-agnostic scores therefore discard two types of information. First, they do not consider scores per individual knowledge base property, but only an aggregate. Second, they are unable to reflect whether there actually exists any overlap, as a score of 0.0 could indicate either no overlap or unequal values.

To reduce some of these limitations, we introduce a class-agnostic confidence score, that returns the actual number of value pairs compared per aggregated score. When no values overlap, the confidence score would be equal to zero, indicating that there was nothing to compare. This potentially allows the differentiation between the cases where no values overlap, or all compared value pairs were unequal.

Similarity Score Aggregation

We implement two methods to aggregate the row similarity scores. We first utilize a weighted average, where the weights assigned to each similarity score are

learned. We then learn a threshold, where aggregated scores above the threshold indicate that the rows describe the same entity. To learn the weights, we model the data in the learning set as row-pairs that either match or not, i.e. with scores of either 1.0 or 0.0. When learning weights, we utilize a genetic algorithm that attempts to maximize the F1 classification performance on the learning set. The confidence of each classification is the relative distance from the aggregated score to the threshold.

As a primary aggregation approach, we use random forest classifiers [Breiman, 2001], where the similarity scores are included as features. We again model the data as row-pairs, where rows within one cluster in the learning set are labeled as matching, otherwise as non-matching. To train the random forest we utilize the WEKA library. We learn the hyperparameters of the algorithm using three-fold nested cross-validation.[10]

Blocking

To ensure that the row clustering scales to large web table corpora, we implement a blocking algorithm. We block comparisons at two points. First, we block to reduce the number of clusters a row is compared to during the parallel greedy clustering, and, second, we block to limit the cluster pairs that are compared with each other during the KLj clustering.

For blocking, we make use of the row labels. We first normalize the labels of all rows and use them to build a Lucene index. Each label in the index forms a block that includes all rows with that exact label. Given a row, we use the row's label to retrieve from the Lucene index similar labels. The blocks of rows behind each retrieved label are candidates to which the original row is compared to.

During the parallelized greedy clustering, we compare a row only to clusters with which the row shares a block. The blocks of a cluster are the union of the blocks of all rows in that cluster. Similarly, during the KLj clustering, two clusters are only compared when they share at least one block.

The blocking yields no decrease in F1 on the gold standard, which shows that it is an effective approach with minimal loss in recall.

Evaluation

To evaluate the performance of the row clustering, we employ the evaluation approach proposed by Hassanzadeh et al. [Hassanzadeh et al., 2009]. We use the set of clusters annotated in the gold standard, denoted as G, and the set of clusters returned by our method, denoted as C, to first compute a one-to-one mapping between the clusters in G and the clusters in C. We map a cluster in C to a cluster in

[10]Tables 8.4 and 8.5 show performances for methods that aggregate features in a given order. We use nested cross-validation to learn the hyperparameters only for the method in the last row, i.e. the method that aggregates all features. We however also use these hyperparameters for all the methods in the rows above the last, i.e. we do not tune the parameters for every row. They are however tuned per aggregation method, also by agnostic or class-specific feature sets, and class separately.

Table 8.4: Average clustering performance and feature importance (FI) scores for alternative row clustering methods.

	Agnostic Features		Class-Specific Features			
	WA	RF	RF			
Method	**F1**	**F1**	**PCP**	**R**	**F1**	**FI**
`LABEL`	0.76	0.74	0.69	0.80	0.74	0.25
`+ BOW`	0.82	0.82	0.78	0.86	0.82	0.07
`+ PHI`	0.82	0.81	0.78	0.86	0.82	0.03
`+ ATTRIBUTE`	0.82	0.81	0.80	0.87	0.83	0.26
`+ IMPLICIT_ATT`	0.81	0.84	0.85	0.89	0.87	0.36
`+ SAME_TABLE`	0.81	0.84	0.86	0.90	0.88	0.02

G, if it contains the highest fraction of rows that are from that cluster in G. In case two clusters in C have the same proportion, we take the cluster with the highest absolute number of overlapping rows. We denote the mapping as M.

Using this mapping we compute average recall (AR), penalized clustering precision (PCP) and their F1 score. Average recall is the average of the individual recalls of the clusters in G. The recall of a cluster in G is equal to the fraction of rows in G that are inside a mapped cluster from C. If no cluster from C was mapped to a cluster in G, the recall of that cluster is zero.

To compute the clustering precision, we compute the precision of all row pairs that are part of the same cluster in C. A pair is determined to be correct if both rows are part of the same cluster in G. Unlike the average recall, this does not measure how well we find cluster, but how well we place rows in the same cluster.

As finding the correct number of unique entities described in the web tables is important, finding the correct number of clusters is important. We therefore penalize the clustering precision, as is also suggested by [Hassanzadeh et al., 2009], if the number of returned clusters deviates from the correct number of clusters. We penalize by multiplying the clustering precision by a penalizing factor. This factor is computed by finding the sizes of C, G or M and dividing lowest by the highest.

Results and Lessons Learned

Table 8.4 shows the performance of various row clustering methods. All numbers in the table are first averaged per fold and per class. We then average the results of the three classes together. The first two columns show row clustering methods that only make use of class-agnostic features. The first uses the weighted average approach to aggregation, while for the second we train a random forest classifier. The following three columns show the performance when using a class-specific feature set and training a random forest classifier. The first row contains the results when using only the label-based similarity scores. For every following row we aggregate

one additional feature or feature set. The last column of the table shows the feature importance, which is the average of the relative importance of the feature inside the learned random forest, for the method with class-specific features. The importance scores shown are derived for the method that aggregates all feature, i.e. the one that corresponds to the last row of the table.

As mentioned above, we use the class-agnostic feature set and the weighted average aggregation to allow comparability. From the table we can see that the class-specific feature set yields a much better performance. We also find that using a random forest outperforms using a weighted average aggregation, at least when it comes to the method shown in the final row. This shows that the expressiveness of the random forest alone, already has a positive impact on performance.

In the remainder of this discussion, we will focus on the class-specific feature set with the random forest aggregation, i.e. the last four columns of the table. The numbers show that the similarity of row labels is a very good indicator if two rows describe the same entity, as it has a high average feature importance of 0.25 and with it we are able to achieve a moderate F1 of 0.74. At the same time, it alone is not enough, and all other similarity features positively impact the row clustering performance when aggregated. This applies especially to the features `BOW`, `ATTRIBUTE` and `IMPLICIT_ATT`, which either have a high feature importance or a large increase in F1.

Both `PHI` and `SAME_TABLE` have smaller feature importance scores and corresponding small effects on F1. It is however important to note, that the increases in F1 is highly influenced by the order in which scores are aggregated. For example, if we remove `PHI` but keep the remaining features, we would end up with an F1 of 0.86 instead of 0.88. As such, the feature importance is likely a better indicator of how useful a feature is. `PHI` however also has a low importance, very close to `SAME_TABLE`, which is a rather primitive method. `PHI` likely achieves a low impact because it does not measure the similarity of two rows directly, but rather compares their tables. On the other hand, the same applies to `IMPLICIT_ATT` and it has a considerably higher impact. This in turn shows the benefit of utilizing the knowledge base as background knowledge.

From the overall results, we can conclude that the best approach is to aggregate multiple features, thereby combining the different signals exploited by the individual features. The `LABEL` feature utilizes the output of the entity label column detection, while the `ATTRIBUTE` feature utilizes the knowledge base as background knowledge to semantically understand the attributes of the table. The `IMPLICIT_ATT` feature also exploits a knowledge base, but to assign semantics to the table as a whole. Finally, the `BOW` method exploits all information of a row, whether it could be mapped to a schema or not.

8.2.3 Entity Creation

The entity creation component receives clusters of rows and transforms each cluster into an entity. An entity first consists of one or more labels, which we extract from

the entity label column of all rows within the cluster. More importantly, an entity contains a set of values mapped to the properties of the knowledge base. Given that at the row-level, each row can have multiple values matched to certain knowledge base properties, and that we have multiple rows in a cluster, there are likely to be conflicting candidate values for one property when creating an entity. We therefore apply the following four-step method to fuse candidate values:

1. **Scoring:** we score candidate values using the Knowledge-Based Trust score of the column from which they were extracted.
2. **Grouping:** we group equal values together. This is done using the data-type-specific equivalence functions.
3. **Selection:** we then select the group with the highest sum of individual candidate value scores.
4. **Fusion:** we fuse a group into one value by using data-type-specific fusers. For string and entity reference types, we utilize the majority value in a group, whereas for numeric and date types we use a weighted median approach. For nominal string and nominal integer, no fusion is necessary, as all values in a group are equal due to the fact that their data-type-specific equivalence functions do not allow for deviation.

For scoring of candidate values, we make use of Knowledge-Based Trust [Dong et al., 2015], where we measure for a certain table column the correctness of its overlapping values, i.e. those matched to an existing fact in the knowledge base, to estimate the trustworthiness of the whole column. We describe Knowledge-Based Trust, the fusion framework and the data-type-specific equivalence functions in more details in Section 4.3.2.

8.2.4 New Detection

After creating entities from row clusters, we now determine whether a created entity describes a new entity, not yet present in the knowledge base. This is done by attempting to match the created entities to the instances of existing entities in the knowledge base. For this, we suggest a set of entity-to-instance similarity features. If there are no matching instances found or the distance between a created entity and an instance in the knowledge base is large enough, the entity is determined to be new. For created entities classified as not new, we attempt to match them to an existing entity in the knowledge base. This allows us to derive row-to-instance correspondences, which are fed back into a second iteration of the pipeline to refine the schema mapping.

Our new detection approach consists of three steps:

1. **Candidate Selection:** we find a list of candidate instances from the knowledge base using a Lucene index built on their labels. To search candidates

for a created entity, we utilize the labels attached to the entity in the entity creation component. Additionally, candidates found must be of the class of the created entity or share one parent class.

2. **Similarity Score Computation:** we compute a score to measure the similarity between the created entity and a candidate instance in the knowledge base. We present multiple different approaches to computing entity-to-instance similarity scores, including how we aggregate them.

3. **Classification:** if the highest similarity for any candidate instance is lower than a learned threshold, we classify the created entity as new. Otherwise we find the candidate instance with the highest similarity score, and in case its score is higher than another threshold, the created entity is classified as existing and a correspondence from the created entity to that instance is generated. The first threshold is optimized for the F1 score of new classifications, while the second threshold is optimized for the F1 score of existing entities correspondences (see Evaluation below).

Entity-To-Instance Similarity Features

We use the following features and metrics to compute a variety of similarity scores between a created entity and one candidate instance from the knowledge base. There are overall six feature types:

- **`LABEL`.** We compute the similarity between the labels of the created entity and the labels of the candidate instance. This is done using two ways. First, we compare the labels of the created entity solely to the label derived from the URI of the knowledge base instance in DBpedia. Second, we compare the labels of the created entities with all surface form labels. The source of these surface forms equals the one used for row clustering (see Subsection 8.2.2). In both cases, we use the Generalized Jaccard similarity [Doan et al., 2012b] with Levenshtein as its inner similarity to compare the labels.

  Using a label extracted from the URI on its own is important for entities with highly ambiguous labels. This is because the label chosen for the URI is usually the one with the least ambiguity. For example, Delhi and New Delhi have equal surface forms in the knowledge base. Only the label extracted from the URI allows us to disambiguate both entities.

- **`TYPE`.** In DBpedia every class is part of a hierarchy with a certain number of parent classes. We compute the overlap of the classes of the candidate instance with the class of the created entity, including in both cases the parent classes. If the candidate instance only shares a parent class with the created entity, it is still deemed somewhat similar.

- **`BOW`.** We create a bag-of-words binary term vector for each created entity by combining the vectors of all its rows, which themselves are created as

described for row clustering above. We then create a vector for the candidate instance in the knowledge base, using its labels, abstract and facts of individual proprieties. We then compute the cosine similarity of both vectors.

- **ATTRIBUTE.** For each property, where a fact exists in both the created entity and the candidate instance in the knowledge base (i.e. an overlap), we determine if the fused fact is equal to the fact in the knowledge base using data-type-specific similarity functions. As there could be multiple overlapping properties, we return a class-specific set of scores, two for each property of a class. This resembles the approach used for row clustering.
- **IMPLICIT_ATT.** We utilize the implicit property-value combinations derived for tables, as described for row clustering above, to derive property-value combinations for a created entity. For this, we sum up the confidence scores of all property-value combinations for the tables of all rows in the entity and divide by the total number of rows to compute an entity-level confidence score. We then compare these property-value combinations at the entity level with overlapping facts of a candidate instance in the knowledge base. We again derive a set of class-specific, two for each property.
- **POPULARITY.** We use a dataset of Wikipedia page links[11] to rank all candidate instances of a created entity by their number of incoming page links. A similarity score is assigned to each candidate instance based on its rank.

Class-Specific and Class-Agnostic Feature Sets

As described in the subsection regarding row clustering, we create class-specific and a class-agnostic versions of the `ATTRIBUTE` and `IMPLICIT_ATT` feature sets for the purpose of comparability. This is done for new detection using the same approach used for row clustering.

Similarity Score Aggregation

We aggregate various similarity scores using the same aggregation approaches utilized for row clustering.

Evaluation

We evaluate the new detection component using the clusters annotated in the gold standard. Before we run new detection on those clusters, we create entities from them as outlined in Section 8.2.3 above. From the gold standard, we know whether a certain created entity describes a new entity or not. In case it describes an existing entity, we additionally know from the gold standard the exact knowledge base instance it describes.

[11] https://wiki.dbpedia.org/Downloads2014#owikipedia-pagelinks, provided along with DBpedia.

Table 8.5: Average performance and feature importance (FI) scores for alternative new detection methods.

	Agnostic Features		Class-Specific Features			
	WA	RF	RF			
Method	**ACC**	**ACC**	**ACC**	**$F1_E$**	**$F1_N$**	**FI**
`LABEL`	0.59	0.57	0.56	0.55	0.72	0.13
`+ TYPE`	0.72	0.70	0.70	0.69	0.83	0.05
`+ BOW`	0.82	0.80	0.78	0.79	0.79	0.07
`+ ATTRIBUTE`	0.87	0.89	0.88	0.86	0.90	0.43
`+ IMPLICIT_ATT`	0.86	0.90	0.89	0.89	0.90	0.26
`+ POPULARITY`	0.86	0.89	0.91	0.90	0.91	0.06

We run our new detection component on those entities to receive a set of entities classified as new, and another set classified as existing with additional correspondences to existing instances in the knowledge base. We use the gold standard to first determine the accuracy of those classifications. The accuracy equals the fraction of correctly classified entities. Existing entities must additionally be matched to the correct instance in the knowledge base to be counted as correctly classified.

We additionally evaluate classification for new and existing entities separately using F1. The precision of the new entities equals the fraction of entities returned with a new classification that are actually new, whereas recall is equal to the fraction of total new entities in the gold standard that were correctly classified as new. The same applies to the existing entities, with a second condition that the entity must be matched to the correct entity instance in the knowledge base as well.

Results and Lessons Learned

Table 8.5 shows the performance of various new detection methods. All numbers are first averaged per fold and per class. We then average the performances of all three classes. The first row shows a method that only utilizes the label-based features. For each following row we aggregate an additional feature into the method.

We show the performances of three methods of aggregation. First, we use the weighted average and random forest aggregation with a class-agnostic feature set. The last four columns all concern the use of class-specific feature set and random forest aggregation. The feature importance shown in the last column is derived for a method that aggregates all features, i.e. the one presented in the last row. Both aggregating using a random forest and using a class-specific feature set have a positive impact on performance.

In the remainder of this discussion we will focus on the class-specific feature set with the random forest aggregation, i.e. the last four columns of the table. From the table we can see that with an accuracy of 0.91, the final aggregated method per-

forms quite well. It achieves a performance considerably better than the LABEL method, which only has an accuracy of 0.56. Additionally, we can see that all similarity features are important and contribute positively to the overall performance. This shows that the combined approach is able to leverage the different signals exploited by the individual features.

The later a feature set is aggregated, the more difficult it is to yield a large absolute increase in performance. As such, the features TYPE and BOW increase accuracy by 14 and 8 percentage points respectively, which is much larger than for the features we include in the rows that follow. At the same time, we see that IMPLICIT_ATT, which has a higher importance score than BOW, only increases accuracy by 1 percentage point. This is likely, because at this stage, improving performance further is more difficult. However, it does increase F1 for existing entities by 3 percentage points.

We find that BOW, while highly useful for existing entities, causes a drop in performance for new entities. This anomaly could be caused by the fact that we tune the hyperparameters only for the method in the last row, while also using them for the remaining methods. If we would actually remove the BOW from the method in the last row, we would not gain any increase in performance in $F1_N$. As such, the anomaly does not appear in the method presented in the last row.

The feature with the largest impact is ATTRIBUTE. It has the highest importance score and high impacts on both accuracy and F1 of new classification. While BOW is aggregated before ATTRIBUTE, its positive impact on accuracy is lower. This indicates the benefit of using the knowledge base and its schema to semantically understand the table.

POPULARITY has a high positive impact on $F1_E$ for the class Settlement, increasing it from of 0.85 to 0.91. This indicates that, given just the name of a settlement, it is safe to assume that the most well-known settlement is meant. This makes sense, as this assumption is typically made when speaking about cities. However, the same feature had a negative impact on the $F1_E$ for the class Song, decreasing it from 0.83 to 0.81. The $F1_E$ remained the same at 0.98 for the class GF-Player.

In this section, we have discussed performances on average, without investigating them at the class-level. In the next section, we will show end-to-end performances per class, additionally discussing also the impact of row clustering and new detection on the overall entity expansion performance per individual class.

8.3 Results

Using the row clustering and new detection methods that aggregate all similarity features and exploit the class-specific feature set, we run the whole pipeline on the testing folds and evaluate its output. We will first evaluate how many of the new entities were correctly found. In the second part, we will evaluate how accurately facts were compiled for new entities. Throughout this evaluation, we utilize three-fold cross-validation, so that the numbers represent averages of all three folds.

Table 8.6: Performance of the LTEE Pipeline in finding new entities.

Class	Clustering	New Detection	P	R	F1
GF-Player	**GS**	**ND**	0.90	0.95	0.92
GF-Player	**RC**	**ND**	0.80	0.95	0.87
Song	**GS**	**ND**	0.89	0.94	0.91
Song	**RC**	**ND**	0.74	0.84	0.78
Settlement	**GS**	**ND**	0.88	0.90	0.89
Settlement	**RC**	**ND**	0.80	0.90	0.84
Average	**GS**	**ND**	0.89	0.93	0.91
Average	**RC**	**ND**	0.78	0.90	0.83

8.3.1 New Entities Found Evaluation

To evaluate how well new entities were found, we utilize precision and recall. To compute precision, we determine the proportion of entities returned as new that are correct. An entity is only correctly new, if its cluster includes the majority of the rows of a new cluster in the gold standard, and these rows at the same time form the majority within the entity's cluster. Recall is the fraction of new entities in the gold standard for which a correct new entity was returned.

Table 8.6 shows the performance of our system for the three classes separately. To evaluate the individual impact of row clustering and new detection, we once evaluate using the row clusters annotated in the gold standard (GS), and once with the row clusters returned by our suggested row clustering method (RC). For new detection, we run in both cases our suggested new detection method (ND).

We generally achieve good performances for all classes, with the best F1 being 0.87 for GF-Player and the lowest 0.78 for Song. This is likely because for songs, the homonym problem is much larger, as there exist many songs of the same name by different artists. Sometimes, these homonyms even represent cover versions, so that they are highly similar in their descriptions, e.g. in runtime or writer.

We find that both new detection and row clustering have a similar negative impact on performance. For GF-Player and Settlement, the new detection component causes a slightly larger decrease, while for Song, the clustering has a higher negative impact. We find that errors of both components compound, e.g. entities that are noise due to bad clustering are returned as new by the new detection, instead of being filtered out. We find that bad clusters often consist of rows that describe existing entities, which is also why new entities found recall is higher than precision.

8.3.2 Facts Found Evaluation

This subsection evaluates how well our pipeline can generate facts for new entities. Again, we need a mapping between entities returned and new entities in the gold

Table 8.7: Performance of the LTEE Pipeline in finding facts for new entities.

			New Entities			Penalized	
Class	**Clustering**	**New Detection**	**P**	**R**	**F1**	**P**	**F1**
GF-Player	**GS**	**GS**	0.84	0.80	0.82	0.84	0.82
GF-Player	**GS**	**ND**	0.85	0.78	0.81	0.79	0.79
GF-Player	**RC**	**ND**	0.85	0.78	0.81	0.74	0.76
Song	**GS**	**GS**	0.79	0.84	0.81	0.79	0.81
Song	**GS**	**ND**	0.80	0.79	0.80	0.73	0.75
Song	**RC**	**ND**	0.80	0.69	0.74	0.60	0.65
Settlement	**GS**	**GS**	0.97	1.00	0.98	0.97	0.98
Settlement	**GS**	**ND**	0.96	0.89	0.93	0.86	0.88
Settlement	**RC**	**ND**	0.96	0.89	0.93	0.81	0.85
Average	**GS**	**GS**	0.87	0.88	0.87	0.87	0.87
Average	**GS**	**ND**	0.87	0.82	0.85	0.79	0.81
Average	**RC**	**ND**	0.87	0.79	0.83	0.72	0.75

standard, for which we utilize the same approach used for the new entities found evaluation. To determine if returned facts for matched entities are correct, we compare them to the facts in the gold standard using the data-type-specific equivalence functions (see Subsection 4.3.5). Recall is measured relative to the number of facts that have been annotated as present for all new entities in the gold standard.

Table 8.7 shows performance per class. We compute two precisions, the first considers only the facts of entities returned as new that were successfully matched to a new entity in the gold standard. The second additionally penalizes precision by counting as wrong all facts of entities returned as new but that could not be matched to a new entity in the gold standard. The first measures how well we actually create descriptions for new entities, while the second additionally highlights the error compounding due to bad row clustering and new detection.

In order to measure the individual impacts of new detection and row clustering on the overall performance, we again perform multiple runs. For the first run, we utilize for both components the correct annotations from the gold standard and evaluate solely fusion performance. In the following run, we use our new detection methods, while for the third run, we additionally use our row clustering methods.

From the table we can see that for football players and songs, we respectively lose 18 and 19 percentage points in average F1 even when row clustering and new detection are perfect. We looked at a sample of errors and were able to identify two main causes. The largest amount of errors is due to the attribute-to-property matching component, where the proportion of errors caused by wrong or missing column matches makes up 43% of all errors. This is followed by 35% of errors that occurred as a result of wrong or outdated data in the tables.

Row clustering and new detection both cause less correct new entities to be created, i.e. they reduce recall. This is especially the case for the class Song. On the other hand, we find that the correctly created new entities consistently have high quality descriptions, as both components do not cause a decrease in precision when not penalizing. For GF-Player, the recall also barely drops, so that the loss in precision when penalizing is due to existing entities being incorrectly determined to be new. Computing separate precisions allows us to differentiate between loss in precision due to low quality descriptions for new entities, which we want to evaluate in this subsection, from loss in precision because entities were incorrectly classified as new. The latter, we have already considered in the previous subsection.

We are able to successfully create descriptions for new entities with an average F1 of 0.83 (0.75 when penalizing). We however find that errors compound throughout the pipeline, which shows the difficulty of the task at hand. Every component has to perform very well for the pipeline to achieve a good performance.

8.4 Related Work

Entity expansion for knowledge base completion requires two subtasks. First, the method must identify the unique entities that are not yet part of a knowledge base, and second, creating descriptions for those entities according to the schema of the knowledge base. To the best of our knowledge, the LTEE Pipeline is the first system to handle the problem of entity expansion.

While there exist approaches for completing a set of incomplete entities (set expansion) or detecting which entities are not part of the knowledge base (emerging entity detection), these fall short when it comes to the two subtasks of entity expansion. Set expansion approaches disambiguate solely by name, and as such are not concerned with the homonym problem, while emerging entity detection do disambiguate between existing and new entities by more than just their label, but are not concerned with actually identifying the exact number of unique new entities to be added to the knowledge base. In regard to the second subtask, both do not create any descriptions for new entities.

However, as both set expansion and emerging entity detection solve similar and related tasks, we are still able to compare our work to both, which we do in this section. Additionally, we compare the performance of individual components of our pipeline to the performances of related work for similar tasks. These are slot filling, schema matching and identity resolution. Finally, we will briefly compare our research to recent and yet unpublished work on long-tail entity extraction.

Set Expansion

Set expansion is a task, where new entities are retrieved to complete a set [Wang et al., 2015, Pantel et al., 2009, Wang and Cohen, 2007]. Set expansion methods however only focus on finding the labels of new entities. The methods rely on

Table 8.8: Comparing the performance of the LTEE Pipeline with reported performances of works on set expansion.

Work	MAP@256	P@5	P@20	P@50	P@100
LTEE Pipeline	0.94	0.90	0.81	0.79	0.79
[Bing et al., 2013]		0.94	0.91	0.77	0.52
[Zhang and Balog, 2017]	0.63				
[Wang and Cohen, 2007]	0.95				
[Wang et al., 2015]	0.78				

a small number of seeds to find more entities from that set. Both the complete sets and the number of seeds are however often small. In contrast, we focus on a scenario in which large sets of entities, already contained in the knowledge base, are extended with potentially large sets of new entities. Finally, most set expansion methods make use of ranked evaluation, where the precision of the top k found entities is measured. In contrast, our evaluation also considers recall.

To compare our work with works of set expansion, we need to utilize ranked evaluation and therefore implement a ranking algorithm for new entities. We rank based on the similarity scores returned by the new detection. These scores measure the distance between one or more existing instances in the knowledge base, and an entity created from web tables. We rank new entities higher, the higher the distance to the closest knowledge base instance is.

Table 8.8 shows the performance of our method when using ranked evaluation compared to the reported performance of set expansion methods. In regard to MAP, we are able to achieve a performance of 0.92, so that we outperform two related works, and are very close to a third. For precision at 5 and 20 we achieve 0.90 and 0.81 respectively, while a related work achieves 0.94 and 0.91. We therefore find that our performance is generally comparable, even though the task we are solving is more difficult. This is because we require new entities to be correctly disambiguated, whereas set expansion methods only care about their labels without disambiguating further.

One work in the area of set expansion also finds descriptions in addition to entities when completing a set [Zhang and Balog, 2017]. For this, the authors exploit a corpus of tables extracted from Wikipedia and DBpedia to augment an incomplete relational table with more entities and their descriptions. To find more entities, the method uses 1 to 5 seeds and first searches for candidates in the web tables and the knowledge base. For this, it exploits the labels of the seeds and the caption of the seed table, and also the ontology of DBpedia. Candidates are then ranked based on how often they co-occur with the seeds and how similar their tables' captions are. Similarly, the method searches for candidate columns in the corpus and ranks them based on how well they fit the seed table. The method then returns a fixed number of entities, where the authors use 256 as their cut-

off. While this approach does generate descriptions, it does not resolve the other limitations of set expansion. Their approach still ranks candidate entities based on their similarity to the seeds, and, more importantly, always returns a fixed number of entities. More importantly, the entities in the testing set are not actual long-tail entities, i.e. no new entities are added to the knowledge base. Finally, the homonym problem is not discussed, and the disambiguation of entities is still primarily based on surface forms. The authors achieve a MAP of 0.63, which is much lower than our of 0.94.

Emerging Entity Detection

Emerging entity detection (EED) is an NLP task about recognizing when a certain mention of a named entity in unstructured text refers to an entity that is out-of-knowledge-base [Hoffart et al., 2014] or that is trending and has only recently become notable [Färber et al., 2016]. EED is considered a sub task of named entity disambiguation or entity linking, where named entity mentions that were recognized in text, are linked to an existing entity in a knowledge base [Hoffart et al., 2014, Färber et al., 2016, Derczynski et al., 2017].

The purpose of EED is to specifically discover that a certain entity mention refers to an actual long-tail entity, not yet present in a knowledge base. This approach differs conceptually from other named entity disambiguation tasks, that could only declare mentions as unlinkable, e.g. due to low confidence, without explicitly identifying them as new [Hoffart et al., 2014].

There exist various approaches to EED [Hoffart et al., 2014, Hoffart et al., 2016, Singh et al., 2016, Wu et al., 2016, Yeo et al., 2016, Färber et al., 2016, Derczynski et al., 2017]. There also exist one widely used benchmark titled AIDA-EE, which was published by Hoffart et al. [Hoffart et al., 2014], and a second benchmark titled WNUT2017, published by Derczynski et al. [Derczynski et al., 2017].

However, EED approaches are not comparable with our pipeline. While EED methods detect if a named entity mention recognized in text is specifically about an out-of-knowledge-base entity, they do not perform any linking between those mentions to determine which mentions are about the same unique emerging entity. As a result, the approaches do not determine the exact number of emerging entities that can be added to the knowledge base, and as such do not attempt to extend knowledge bases with emerging entities.

EED approaches actually resemble our new detection component, as they classify whether an entity is new without a broader context of determining the unique number of entities to be added to the knowledge base, nor creating their descriptions. We can therefore compare the performance of our new detection method with the results reported by emerging entity detection works, which we do in Table 8.9. We are able to compare two measures: the accuracy of correctly classifying entities as either new or existing, and the F1 score of classifying entities as new.

From the table we can see that we outperform all EED methods in both accuracy and F1. It is important to note that the tasks are different, and the results are

Table 8.9: Comparing the new detection performance of the LTEE Pipeline with reported performances of works on emerging entity detection.

Work	Dataset	ACC	$F1_N$
LTEE New Detection	T4LTE	0.91	0.91
Best in [Derczynski et al., 2017]	WNUT2017	-	0.42
[Hoffart et al., 2014]	AIDA-EE	0.76	0.69
[Wu et al., 2016]	AIDA-EE	0.78	0.72
[Zhang et al., 2019]	AIDA-EE	0.79	0.70

therefore not directly comparable. Most importantly, EED methods are in the area of NLP, whereas our methods are data integration methods.

Slot Filling

Many works that exploit web table data for knowledge base augmentation [Dong et al., 2014a, Sekhavat et al., 2014, Ritze et al., 2015, Ritze et al., 2016, Ritze and Bizer, 2017] focus on the task of slot filling, i.e. adding missing facts for existing entities. Dong et al. [Dong et al., 2014a] introduce a probabilistic approach that exploits background knowledge, in their case Freebase, to construct a large knowledge base using web data, including web tables. The extracted facts however only describe entities that already exist in Freebase. For more than 200 million facts, they report an expected accuracy higher 0.90. In Chapter 5, we were able to achieve for slot filling an F1 of 0.71, and a precision of 0.64. While we focus on finding facts for new entities, which is likely a more difficult task, we achieve with an F1 of 0.83 and a precision of 0.87 (see Table 8.3.2) a performance comparable to that achieved by Dong et al., and higher than the one achieved in Chapter 5.

Schema Matching and Identity Resolution

Throughout the components of the pipeline, we apply approaches for which a large corpus of related work exists. This includes schema matching methods, which are surveyed in [Bernstein et al., 2011]. For row clustering and new detection, we employ identity resolution methods, which are extensively surveyed in [Christophides et al., 2015, Dong and Srivastava, 2015c].

For the specific use case of matching web table attributes to DBpedia properties, authors from our research group were able to achieve an F1 score of 0.81 [Ritze and Bizer, 2017]. While our performance of 0.92 (see Table 8.3) is higher, we consider a smaller number of classes and properties.

We can compare our identity resolution methods to work that perform row-to-instance matching, for which a large corpus of research exists. We can compute row-to-instance correspondences by looking at all entities returned as existing by

Table 8.10: Comparing the row-to-instance matching performance of the LTEE Pipeline with reported performances of related work.

Work	F1	ACC
LTEE Pipeline	0.87	0.89
[Ritze and Bizer, 2017]	0.80	
[Zhang, 2017]	0.87	
[Limaye et al., 2010]		0.83
[Bhagavatula et al., 2015]		0.93

our method. For every row in the cluster of those entities, we can derive a row-to-instance correspondence. Using the testing folds of the gold standard, we can compute the accuracy and F1 score of those correspondences. Table 8.10 compares our performance to reported performances of related work.

We achieve an average F1 score of 0.88, compared to 0.80 [Ritze and Bizer, 2017] and 0.87 [Zhang, 2017] in the related work, and we achieve an accuracy of 0.90, compared to 0.83 [Limaye et al., 2010] and 0.93 [Bhagavatula et al., 2015] in the related work. Our performance is therefore comparable to existing methods for finding row-to-instance correspondences. This shows that our pipeline is generally robust, and could be used for a range of tasks, not just for entity expansion.

Recent Work on Long-Tail Entity Extraction

There exists one work by Zhang et al. [Zhang et al., 2020] in the area of long-tail entity extraction that has recently been presented at the Web Conference (WWW). We will briefly compare their approach to the LTEE Pipeline.

Similar to our pipeline, the approach performs identity resolution twice, albeit in reverse. We first disambiguate entities described in rows among each other in the row clustering component, and then determine in the new detection component whether an entity described by a cluster is new. Zhang et al. first perform novel entity discovery classification, where they determine whether entity mentions in rows describe entities that are out-of-knowledge-base. They then disambiguate between those entity mentions in a second step, which they term entity resolution.

The authors also use DBpedia as a target knowledge base. For their web table corpus, they use the 2015 WDC corpus. They evaluate their novel entity discovery classification on a sample of 20k individual rows drawn from the corpus, where they achieve an accuracy of 0.83. Our new detection component, which resembles novel entity discovery classification, achieves an accuracy of 0.91.

To evaluate their entity resolution approach, Zhang et al. randomly select 250 entity mentions, further selecting for each 5 candidate mentions with similar labels. They then evaluate how well they can disambiguate each of the 250 mentions with its candidates. We believe this to be an ineffective way of building a testing

set for entity resolution. The authors did not ensure that corner cases, especially homonyms, are covered. In fact, they achieve a performance of 0.97 in F1 simply by exploiting lexical string similarities on the labels of entity mentions. In contrast, considering homonyms is a primary aspect of our experimental setup.

Overall, the focus of Zhang et al. remains on the two identity resolution tasks. They do not perform end-to-end evaluation or profiling, nor do they create descriptions for new entities. One interesting aspect of their work is that they consider for novel entity discovery classification, i.e. new detection, features that go beyond comparing to existing entities in the knowledge base. E.g., they consider the origin table of a row and the correlation between rows to determine whether a row describes a novel entity. This could be a potential additional signal for new detection.

8.5 Summary

In this chapter, we introduced the LTEE Pipeline, which to the best of our knowledge is the first system that enables entity expansion from web tables. The pipeline consists of multiple components, including schema matching, row clustering, entity creation and new detection. With the pipeline, we are able to successfully compile from a web table corpus new and previously unknown entities along with their descriptions according to the schema of the knowledge base.

We evaluated our pipeline using T4LTE, a manually annotated gold standard of web tables. We built T4LTE for evaluating entity expansion for three versatile classes in DBpedia. In the gold standard, we annotate entities not yet present in DBpedia, so that we evaluate our methods on actual long-tail entities. We additionally annotate facts for those entities, for the purpose of evaluating how well we can compile their descriptions.

We achieve an average F1 of 0.83 for both finding new entities and compiling their descriptions. We find that the task is non-trivial, as errors compound along the individual components of the pipeline. As such, we suggest and evaluate alternative methods for all components of the pipeline.

Throughout all components, we generally find that aggregating multiple features that exploit different signals yields the best performance. We also find that features that make use of label similarity, while important, are not sufficient to yield good results. Additionally, we show that features that use the knowledge base as background knowledge, e.g. to semantically understand cell values or to derive semantic information about web tables, have a positive impact on performance. Finally, we are able to utilize the output of the pipeline in a second iteration to achieve a large improvement in schema matching performance.

While evaluating the pipeline on a gold standard gives us insight into how well our methods work in extracting long-tail entities from web tables, it does not provide insight into how many new entities can actually be extracted from web tables to be added to a knowledge base. In the next chapter, we run our pipeline on the full web table corpus to profile the potential of web tables for entity expansion.

Chapter 9

Profiling Web Tables for Entity Expansion

The previous chapter introduced the LTEE Pipeline, the first system for extracting long-tail entities from web tables. We evaluated the pipeline on web tables that are part of the T4LTE gold standard. While evaluation on a gold standard facilitates method development and enables comparability between different approaches, we are ultimately interested in running and evaluating our methods on an actual web table corpus. More importantly, we are interested in understanding how well web tables are suited for entity expansion, and what their potential in adding new entities to a cross-domain knowledge base is.

This chapter aims to answer those question by presenting the results of running our pipeline on the 2012 WDC web table corpus. As the pipeline is trained using T4LTE, we are able to run the pipeline to extend the three classes included in T4LTE: GridironFootballPlayer (GF-Player), Song and Settlement. We evaluate the output based on the number of new entities and the corresponding number of new facts we are able to add DBpedia. We also estimate the accuracy of the pipeline output by manually evaluating the accuracy of a sample of entities returned as new by the pipeline. Finally, we also compare the densities of properties for extracted long-tail entities with the densities of the same properties in DBpedia.

To the best of our knowledge, this research is the first to profile web tables for the task of entity expansion. The contributions and findings of this chapter are:

- **Profiling the Potential of Web Tables for Entity Expansion:** by running the LTEE Pipeline on a whole web table corpus, we are able to profile the potential of web tables for the task of entity expansion. We find that we are able to add approximately 14 thousand new entities with 44 thousand new facts for the class GF-Player, an increase of respectively 67% and 32% when compared to the existing entities in DBpedia. For Song, we are able to add about 187 thousand new entities with 394 thousand new facts, an increase of 356% and 125% respectively.

- **Pipeline Performance:** we find that the LTEE Pipeline is able to successfully perform entity expansion using web tables. The accuracy of new entities returned by the pipeline is 60% for GF-Player, and 70% for Song. The created descriptions of new entities have a high accuracy of 95% for GF-Player and 85% for Song. We describe extensively the sources of errors and discuss why the accuracy of new entities is lower than the one achieved by the pipeline on the T4LTE gold standard.
- **Profiling of Property Density Distribution for Long-Tail Entities:** we find that the property densities among the new entities is moderately high, but lower than the density in DBpedia. We also find that the distribution of property densities between web tables and DBpedia differs. We attribute this difference to the fact that data in web tables is relational, whereas in a knowledge base, the entity itself is in focus. This changes the likelihood of inclusion for certain properties.
- **Web Table Potential and Wikipedia Notability Criteria:** we discuss how the potential of using web tables to extend DBpedia with new entities likely depends on both the Wikipedia notability criteria, which determines which entities are ultimately included in DBpedia, and the likelihood of niche entities within a class. We find that for the class Settlement, the criteria for the inclusion within Wikipedia is very lenient, while at the same time, no settlement is truly niche. As such, we are unable to successfully perform entity expansion for the class Settlement from web tables.

We publish both our code and T4LTE, which is used to train the pipeline. The 2012 WDC web table corpus is already publicly available. As such, all resources used within this chapter are available, allowing the replication of our work.[1]

This chapter is organized as following. In the next section we will outline our experimental setup, which includes statistics of the existing knowledge present in the knowledge base and a look at the web table corpus for the three evaluated classes. Section 9.2 outlines how we trained the LTEE Pipeline and ran it on the web table corpus. Section 9.3 presents the results of the profiling, an evaluation of the accuracy of the pipeline output, and an extensive discussion on lessons learned. The final section summarizes this chapter.

The work presented in this chapter has previously been published in [Oulabi and Bizer, 2019a].

9.1 Experimental Setup

This section describes the datasets we use within this chapter. Primarily, we use DBpedia as the target knowledge base to be extended, while the 2012 WDC web table corpus is used as input by our pipeline. We limit ourselves to the evaluation

[1] http://data.dws.informatik.uni-mannheim.de/expansion/LTEE/

of three classes GF-Player, Song and Settlement, as we use the annotations within the T4LTE gold standard (see Subsection 8.1.1) to train our pipeline.

9.1.1 Target Knowledge Base

We employ DBpedia [Lehmann et al., 2015] as the target knowledge base to be extended. We have described DBpedia in more detail in Section 2.4. It is extracted

Table 9.1: Number of entities and facts for the three profiled classes in DBpedia.

Class	Entities	Facts
GF-Player	20,751	137,319
Song	52,533	315,414
Settlement	468,986	1,444,316

Table 9.2: Fact and density statistics for the profiled properties in DBpedia.

Class	Property	Facts	Density
GF-Player	**birthDate**	20,218	97.43%
GF-Player	**college**	19,281	92.92%
GF-Player	**birthPlace**	17,912	86.32%
GF-Player	**team**	13,349	64.33%
GF-Player	**number**	11,430	55.08%
GF-Player	**position**	11,240	54.17%
GF-Player	**height**	10,059	48.47%
GF-Player	**weight**	10,027	48.32%
GF-Player	**draftYear**	7,947	38.30%
GF-Player	**draftRound**	7,932	38.22%
GF-Player	**draftPick**	7,924	38.19%
Song	**genre**	47,040	89.54%
Song	**musicalArtist**	45,097	85.85%
Song	**recordLabel**	43,053	81.95%
Song	**runtime**	42,035	80.02%
Song	**album**	40,666	77.41%
Song	**writer**	33,942	64.61%
Song	**releaseDate**	31,696	60.34%
Settlement	**country**	433,838	92.51%
Settlement	**isPartOf**	416,454	88.80%
Settlement	**populationTotal**	292,831	62.44%
Settlement	**postalCode**	154,575	32.96%
Settlement	**elevation**	146,618	31.26%

from Wikipedia and, as a result, the covered entities in DBpedia are ultimately limited to those identified as notable by the Wikipedia community. We use the 2014 release of DBpedia, as this release has been used in related work [Ritze et al., 2015, Ritze et al., 2016], and its release date is also closer to the extraction of the web table corpus used in this chapter.

Tables 9.1 provide an overview of the number of entities and facts in DBpedia for the three classes profiled in this chapter. We only consider facts for properties that have an initial density of at least 30%. The table shows that DBpedia already covers tens of thousands of entities for the profiled classes. This could indicate that most of the well-known entities are already covered, so that we are especially interested in finding entities from the long tail.

Table 9.2 shows on the other hand the densities of the individual properties. The table reveals that the density differs considerably from property to property. Only the properties of class Song have consistently high densities larger than 60%. GF-Player has many properties, but half of them have a density below 50%. The class Settlement suffers from both, a small number of properties, and low densities for some of them.

Table 9.3: Characteristics of the employed 2012 WDC web table corpus.

	Average	Median	Min	Max
Rows	10.37	2	1	35,640
Columns	3.48	3	2	713

Table 9.4: Number of matched and unmatched values for the profiled classes when using the T2K matching framework.

Class	Tables	V_{Matched}	$V_{\text{Unmatched}}$
GF-Player	10,432	206,847	35,968
Song	58,594	1,315,381	443,194
Settlement	11,757	82,816	13,735

9.1.2 Web Table Corpus

As input for our pipeline we utilize the English-language relational tables set of the 2012 WDC web table corpus, which consists of 91.8 million tables.[2] We have described this corpus in more detail in Section 3.3.

Table 9.3 gives an overview of the general characteristics of tables in the corpus. We can see that the majority of tables are rather small, with an average of 10.4 rows and a median of 2, whereas the average and median number of columns are

[2] http://webdatacommons.org/webtables/#toc3

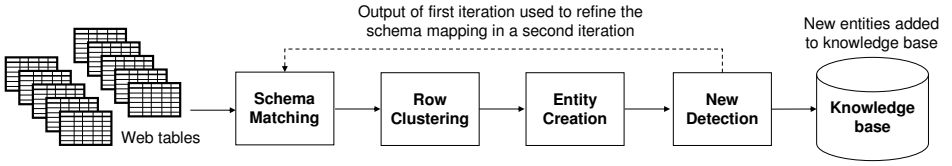

Figure 9.1: Overview of the Long-Tail Entity Extraction Pipeline.

Table 9.5: A comparison of the performances in row clustering (RC), new detection (ND) and end-to-end entity expansion (ETE) for the version of LTEE Pipeline introduced in [Oulabi and Bizer, 2019a] (CA-WA+RF) and the one introduced in the previous chapter (CS-RF).

	GF-Player			**Song**			**Settlement**		
	RC	**ND**	**ETE**	**RC**	**ND**	**ETE**	**RC**	**ND**	**ETE**
CA-WA+RF	0.90	0.91	0.87	0.76	0.90	0.72	0.82	0.87	0.80
CS-RF	0.94	0.92	0.87	0.81	0.92	0.78	0.89	0.89	0.84

3.5 and 3 respectively. As a result, a table on average describes 10 entities with 30 values, which likely is a sufficient size and potentially useful for finding new entities and their descriptions.

In Chapter 5, we have profiled the potential of the same corpus for the task of slot filling, i.e. finding missing values for existing DBpedia entities. To match the tables to DBpedia, we employed the T2K matching framework. Given the mapping generated by T2K, Table 9.4 shows matching statistics for the three evaluated classes. The first column shows the number of matched tables that have at least one matched attribute column. Rows of those tables were matched directly to existing entities in DBpedia. From the second and third columns we see how many values were matched to existing entities and how many values remained unmatched. While more values were matched, the number of unmatched values is still large, especially for the class Song. It is from these unmatched values that we can potentially find new entities and create their descriptions.

9.2 Methodology

We run a version of our pipeline introduced in [Oulabi and Bizer, 2019a]. Table 9.5 compares the performance of this pipeline with the one introduced in the previous chapter. The performance is measured using the T4LTE gold standard and includes numbers for row clustering, new detection and end-to-end entity expansion. While in essence, the structure of the pipelines is equal (see Figure 9.1), the pipeline in [Oulabi and Bizer, 2019a] uses only a class-agnostic feature set, and aggregates the similarity scores by combining both, a weighted average and a random forest

aggregation method. The pipeline introduced in the previous chapter uses a class-specific feature set and trains only a random forest. The performance of the pipeline from [Oulabi and Bizer, 2019a], i.e. the one used in this chapter, is somewhat lower, albeit comparable.

Before running our pipeline, we need to train various components within the pipeline using the T4LTE dataset. The components of the pipeline that contain trained models are the attribute-to-property matching, the row clustering and the new detection components. We train three separate instances of the pipeline, one per profiled class.

When running our pipeline on the web table corpus we first match all web tables jointly to all classes in DBpedia using T2K. The table-to-class matching approach of T2K is described in detail in Section 4.3.1.

After having a set of web tables matched to each of the three evaluated classes, we run the pipelines for each class separately. To make use of the duplicate-based matching in the schema matching component (see Subsection 8.2.1), we need row clusters and entity-to-instance correspondences. These are created using a preliminary run of the pipeline. The results shown in the next section therefore represent the output of the second iteration of the pipeline.

9.3 Results and Lessons Learned

Table 9.6 shows per class the results of the large-scale profiling. The first two rows list the number of tables and rows per class. The three rows below describe existing entities found by the pipeline, to how many unique instances in the knowledge base they were matched, and the ratio of the two numbers. The next two rows contain the number of new entities found by the pipeline, the number of corresponding facts, and for both the relative increases when compared to existing entities and facts in the knowledge base (i.e. the numbers in Table 9.1).

From the entities returned as new by the pipeline, we pull a stratified sample of 50 entities per class. For this, we first group the returned entities by the number of facts their descriptions contain. We then pull from each group a number of entities proportional to the size of the group in relation to the total number of new entities. The accuracy of new entities equals the fraction of entities that are actually new, given the 2014 version of DBpedia. The accuracy of facts equals the proportion of correct facts, within new entities that were determined to be correct. These accuracies are given in the final two rows of Table 9.6.

9.3.1 Existing Entity Matching Ratio

We find that the ratio of existing entities returned by the pipeline to matched instances in the knowledge base differs by class. While for players and settlements the number is good, i.e. it is close to 1, it is less so for songs. When evaluating the pipeline on the T4LTE gold standard, Song was the class with the worst

Table 9.6: Results and evaluation of running the LTEE Pipeline on the 2012 WDC web table corpus.

Class	GF-Player	Song	Settlement
Tables	10,432	58,594	11,757
Total Rows	648,741	2,173,536	1,472,865
Existing Entities	30,074	40,455	28,628
Matched KB Instances	24,889	29,140	27,365
Matching Ratio	1.21	1.39	1.05
New Entities	13,983 (+67%)	186,943 (+356%)	5,764 (+1%)
New Facts	43,800 (+32%)	393,711 (+125%)	7,043 (+0%)
N. Entities Accuracy	0.60	0.70	0.26
N. Facts Accuracy	0.95	0.85	0.94

performance at row clustering, i.e. identifying the exact number of unique entities described in the web tables. This shows that we need to implement more sophisticated clustering methods or, alternatively, perform deduplication after clustering.

As mentioned before, the pipeline used for the profiling in this chapter however differs from the one we actually suggested in this thesis. Table 9.5 shows that the one introduced in this thesis already improves row clustering methods, achieving a better performance for all classes, including Song.

9.3.2 New Entities Added and the Wikipedia Notability Criteria

Table 9.6 shows for both entities and facts the relative increase that was achieved by the LTEE Pipeline by extracting long-tail entities and adding them to the knowledge base. Figure 9.2 visualizes in addition to the table the increases for the classes Song and GF-Player.

We find that the number of new entities we can add differs considerably per class. For the class Song we have a very large number of new entities and facts, even if we would correct the number of new entities by the matching ratio. For Settlement, there are in comparison very few new entities. When considering that only 26% of them are actually new, we would achieve a relative increase in knowledge base entities of only 0.3%. While for the class GF-Player the absolute increase in the number of entities and facts is smaller than for the class Song, the relative increase is still quite large, as we achieve an increase of 67% for entities and 32% for facts.

The differences between the three classes can be explained by investigating the notability rules of Wikipedia and the nature of the classes themselves. There is a very large number of obscure songs. It is very common, even for well-known artists, to release only a few songs from an album as singles, which are then often the only songs that become popular enough to be actually covered by Wikipedia.

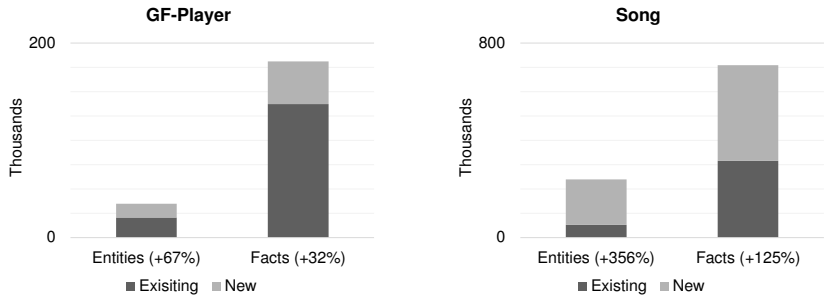

Figure 9.2: Visualization of the number of new entities and facts added to DBpedia for the classes GF-Player and Song relative to the number of existing entities and facts.

Here, the notability rules compound the issue, as songs only receive their own Wikipedia article if they are notable on their own. This applies even when they were independently released as a single.[3]

For Settlement, almost the opposite is true. While there are many small villages, they are never irrelevant, as there are always enough people living in them who might contribute to a Wikipedia article. More importantly, Wikipedia deems any place notable if it has legal recognition, which applies to any settlement, no matter how small it is.[4] As a result, Wikipedia already covers a lot of settlements, and it is difficult to find new ones.

Football players are in the middle of both classes. There are not as many obscure football players as songs. This is because the number of professional teams is somewhat limited. However, there are still many football players that are obscure enough not to be covered in a Wikipedia article.

9.3.3 Property Densities for New Entities

Table 9.7 shows the property densities for new entities. As one would expect, the properties are not as dense as in DBpedia (see Table 9.2). More importantly, the distribution of densities differs considerably between the new entities and the existing entities in the knowledge base. For football players, personal properties like *birthDate* and *birthPlace* have a very low density for new entities extracted by the pipeline, but a very high density for existing entities in the knowledge base. This might be, because the focus in Wikipedia is on describing one entity, i.e. the athlete, whereas in web tables, the games, teams, and drafts are more in focus. The date or place of birth might not be very relevant when looking at a table of athletes of a team, a draft, or a ranking. For those tables, a property like *position* might be

[3] https://en.wikipedia.org/wiki/Wikipedia:Notability_(music), accessed 2020-03-26

[4] https://en.wikipedia.org/wiki/Wikipedia:Notability_(geographic_features), accessed 2020-03-26

Table 9.7: Property densities for new entities extracted using the LTEE Pipeline.

Class	Property	Facts	Density
GF-Player	**position**	9,204	65.82%
GF-Player	**team**	7,637	54.62%
GF-Player	**college**	6,849	48.98%
GF-Player	**weight**	5,915	42.30%
GF-Player	**height**	4,253	30.42%
GF-Player	**number**	2,951	21.10%
GF-Player	**birthDate**	2,537	18.14%
GF-Player	**draftPick**	2,404	17.19%
GF-Player	**draftRound**	1,538	11.00%
GF-Player	**draftYear**	386	2.76%
GF-Player	**birthPlace**	126	0.90%
Song	**musicalArtist**	143,656	76.84%
Song	**runtime**	115,652	61.86%
Song	**album**	52,664	28.17%
Song	**releaseDate**	47,377	25.34%
Song	**genre**	23,814	12.74%
Song	**recordLabel**	10,278	5.50%
Song	**writer**	270	0.14%
Settlement	**isPartOf**	2,889	50.12%
Settlement	**postalCode**	1,605	27.85%
Settlement	**country**	1,232	21.37%
Settlement	**populationTotal**	1,214	21.06%
Settlement	**elevation**	103	1.79%

more relevant, as it allows comparability between players, which explains why its density is even higher than in the knowledge base.

For songs, the properties *writer*, *genre*, and *recordLabel* have very low densities when compared to DBpedia. It is likely that for *genre*, this is a column matching issue, as genres are not always objectively defined. For *writer* and *recordLabel* there could be two causes. First, they might be uninteresting properties, and second, there are often multiple correct facts. The record label might even differ by country. This makes these properties difficult to match, and, more importantly, unlikely to be included in a web table. We can confirm the latter, as these properties occurred very rarely in the tables we annotated in the T4LTE gold standard.

9.3.4 Accuracy and Sources of Errors

When looking at the accuracies of new entities, we also find differences per class. We achieve a moderate accuracy for songs, a sub-par accuracy for players and a

low accuracy for settlements.

The primary reason for the low accuracy for settlements are conflicting values in an existing entity extracted from web tables and its corresponding instance in the knowledge base. This includes outdated population numbers, but also isPartOf values, where the values in the entity and in the knowledge base are both correct, but different, preventing the entity from matching. This problem makes up 36% of all errors. On the other hand, 25% of errors are because the new entity does not describe a settlement, but a different place, like a region or a mountain. This error is caused by incorrect table-to-class matching. These problems are magnified because there are so few new entities to begin with, so that these corner cases make up a huge proportion of the new entities returned.

For GF-Player, the sources of errors include bad attribute-to-property matching, entities not being football players due to bad table-to-class matching, and incomplete information in DBpedia. The latter happened primarily when a football athlete was not assigned the correct class in DBpedia.

For songs, the sources of errors are versatile. The main contributors are bad new detection, incorrect table-to-class matching, and bad clustering. The latter meaning that an entity was incorrectly detected as new, as a result of being created from rows that describe different entities. Such entities would in fact constitute noise and should be filtered out.

9.3.5 Comparison with Gold Standard Performance

We generally notice that the end-to-end performance is not at the same level as the one achieved when evaluating the pipeline on T4LTE. This might indicate that T4LTE either does not completely reflect the nature of the task, or the it is not large enough. However, we believe it is because we did not annotate noise in the gold standard. This includes for example explicitly adding tables incorrectly matched to the evaluated class, rows that are inconsistent in the entities they describe, or tables that describe knowledge that is simply incorrect or fake. All this is present in the real web table corpus and requires consideration to achieve a high end-to-end performance during actual entity expansion.

On the other hand, we are able to achieve a higher performance when restricting evaluated entities to those with more than two facts. For example, for the class GF-Player, if we do not consider entities with one value, the accuracy of new entities rises to 0.72.[5] If we further do not consider entities with two values, we achieve an accuracy of 0.85. This would mean excluding 6,360 entities, but also that with an accuracy of 0.85 we can add 7,623 entities with 34,922 facts to the knowledge base, a relative increase of 37% for entities, and 25% for facts. This indicates that the more we know about an entity, the better we can determine if it is actually new or not, which is reasonable. However, this would also come at a loss of the number of new entities we can add to the knowledge base.

[5]The same effect can be observed for the other two classes, however the positive impact on performance is not as large.

We also achieve a consistently high accuracy for new facts, similar to the high performance on the gold standard, as seen in Section 8.3.2. This means that when it comes to finding descriptions, our performance is quite good, even if densities are lower when compared to DBpedia.

9.3.6 Comparison to Related Work on Slot Filling

While there is no related work that profiles the potential of web data for entity expansion, we can compare our results to works that focus on slot filling.

In Chapter 5, we use web tables to perform slot filling for DBpedia. We are able to fuse 378,892 facts, 64,237 of which are new facts for existing entities. The expected precision of those new facts is 0.64. Compared to that work, we are able to extract a higher number of new facts with a higher accuracy, especially considering that we are profiling only three classes.

When looking at numbers in Tables 5.5 and 5.11 of Chapter 5, we find that the classes Athlete and MusicalWork, i.e. the parent classes of GF-Player and Song respectively, have high numbers for matched tables, extracted values and fused triples. The opposite is true for Settlement, where the numbers of both, tables with attribute-to-property correspondences and fused triples, are comparatively very low. These numbers are in line with the finding of this chapter that the potential for entity expansion from web tables is low for settlements.

In their work on Knowledge Vault [Dong et al., 2014a], Dong et al. are able to extract from the Web 90 million new facts for existing entities in Freebase with an expected correctness higher than 0.9. While the amount of facts that we discover is much smaller, we are only dealing with three classes and looking at facts only for new entities. Additionally, we only use web tables to extract new facts, while Dong et al. also use free text, HTML DOM trees and schema.org annotations. In fact, only about 600 thousand values can be extracted from web tables with an expected accuracy higher than 0.9. These values correspond to both, triples that either exist or not in the knowledge base, and are not fused, i.e. multiple values could correspond to the same triple. Compared to this, the 445 thousand facts we are able to extract for only new entities and only three classes is quite a comparable number. Our average accuracy of 0.91 for these facts is also similar.

9.4 Summary

This chapter explored the potential of web tables for entity expansion, i.e. augmenting a knowledge base with new and formerly unknown long-tail entities. For this, we run the LTEE Pipeline on the 2012 WDC web table corpus, using DBpedia as the target knowledge base to be extended. To train the pipeline, we use the annotations in the T4LTE gold standard, which covers the classes GF-Player, Song and Settlement.

Depending on the class, we are able to successfully add to DBpedia tens and

even hundreds of thousand new entities and corresponding facts. We find that the numbers differ considerably per class and discuss in this chapter how this could be caused by the notability criteria of Wikipedia, from which DBpedia is extracted.

While the performance of our system is generally good, there is potential for improvements throughout all components of our pipeline. Many of the new entities are not actually new to a given class, but from a different class. This requires improvements in the table-to-class matching. By investigating the pipeline output for existing entities, we also find that our row clustering components is too strict in splitting clusters. An improvement in the clustering method or a deduplication component could reduce this problem. Finally, when looking at new entities extracted from web tables, we find that entities with more facts are more accurate. A filtering component that, based on this observation, computes a confidence for entities and filters out low-confidence entities could significantly improve the accuracy of the pipeline output.

Our pipeline currently requires for each class of the knowledge base class-specific manually annotated training data in the form of positive and negative entity matches. Considering the number of classes in a cross-domain knowledge base like DBpedia, this would ultimately limit the viability of our pipeline for automatic cross-domain entity expansion from web tables. In the next chapter, we introduce an approach that counteracts the dependence on manually labeled training examples by using weak supervision in the form of a small number of bold matching rules.

Chapter 10

Weak Supervision for Long-Tail Entity Extraction

In Chapter 8, we introduced the *Long-Tail Entity Extraction Pipeline*, the first system that performs entity expansion from web table data. In the previous chapter, we ran the pipeline on a complete web table corpus to evaluate the potential of web tables for the task of entity expansion. For this, models within the pipeline were trained using a large dataset of manually labeled class-specific data. Given that knowledge bases can have many classes, manual labeling limits the applicability of automatic entity expansion from web tables.

Weak supervision approaches aim at reducing labeling effort by using supervision that is more abstract or noisier compared to traditional manually labeled high-quality training examples, in this context also termed *strong supervision* [Ratner et al., 2017]. *Data programming* [Ratner et al., 2016] is a paradigm where experts are tasked with codifying any form of weak supervision into labeling functions. These functions are then employed within a broader system to generate training data by assigning labels and confidence scores to unlabeled data. Recently, various different systems based on the data programming paradigm have been suggested [Bach et al., 2019, Ratner et al., 2017, Varma and Ré, 2018].

For many types of entities, humans generally possess knowledge about when entities definitely match, and what are strong signals that entities do not match. Writing down this general knowledge in the form of simple bold matching rules requires far less effort than manually labeling a large set of entity pairs as matching and non-matching. Building on this observation and the data programming paradigm, this paper investigates for the task of long-tail entity extraction whether strong supervision in the form of positive and negative entity matches can be replaced by a set of simple bold matching rules. In order to make it easy to write down such rules, we restrict the rule format to conjuncts of equality tests. These tests are expressed using the schema of the knowledge base without requiring experts to assign weights or specify similarity metrics.

We ensemble these matching rules with an unsupervised matching model to

create a weakly supervised labeling function. We then introduce a bootstrapping method that exploits the weakly supervised labeling function to generate training data and train a supervised machine learning algorithm. Using these approaches, we are able to considerably reduce supervision effort compared to manually labeling positive and negative entity matches, while achieving a comparable performance.

The contributions of this chapter are:

- **Weak supervision using bold matching rules:** we introduce a weak supervision approach that substitutes manually labeled training pairs by a small set of user-provided bold matching rules. These rules are easy to create, as they use simple attribute tests built on the schema of the knowledge base. Previous approaches to weak supervision use labeling functions that are black-boxes and possibly much more complex in their nature. They can be dependent on external resources, contain actual hand-written programs, or require pre-trained and separately maintained models.
- **Ensembling of weak supervision with an unsupervised model:** we introduce an approach for ensembling weak supervision in the form of bold matching rules with an unsupervised model to create a weakly supervised labeling function. The unsupervised model ensures full coverage, while simultaneously counteracting the negative impact of possibly biased rules. This allows us first to require only a small number of rules per class and second to require rules to be only somewhat accurate. This further reduces the effort required to create these matching rules. Existing approaches make use of a larger number of individual labeling functions, while not actually explicitly considering their low coverage. To reduce the negative impact of inaccurate or biased labeling functions, generative algorithms are used.
- **Bootstrapping a supervised model using weak supervision and random forests:** we introduce a bootstrapping approach that uses the weakly supervised labeling function and a set of unlabeled web tables to generate, possibly noisy, training data for both row clustering and new detection. We discuss why random forests are especially useful for learning from noisy training data and what parameter choices reduce the impact of noise on the performance of the trained model. Using our approach, we can train a supervised machine learning model that outperforms the weakly supervised labeling function from which it was bootstrapped. While existing weak supervision approaches have suggested bootstrapping supervised models from labeling functions, they have not discussed the usefulness of random forests or optimal hyperparameter choices when lacking a validation set for tuning.

We publish both our code and our gold standard, including the used cross-validation folds. As such, all resources used within this chapter are publicly available, allowing the replication of our work.[1]

[1] http://data.dws.informatik.uni-mannheim.de/expansion/LTEE/

This chapter is structured as follows. First, we recap the experimental setup, which equals that of Chapter 8. Section 10.2 describes our weak supervision methodology, while Section 10.3 presents the results. In Section 10.4, we discuss our work. This includes discussions on the importance of ensembling with an unsupervised model, the impact of parameter choices for bootstrapping, and the ability of random forest to train on noisy training data. Section 10.5 compares our approach to the related work. The final section summarizes this chapter.

The work presented in this chapter has previously been published in [Oulabi and Bizer, 2019b], however it contains the following additions and changes: (1) we describe, investigate, and discuss the effect of parameters chosen for the bootstrapping approach, and (2) we evaluate the impacts of the unsupervised class-agnostic model and the user-provided matching rule sets extensively in our discussion.

Table 10.1: Overview of the number of labels in the T4LTE gold standard.

Label type	GF-Player	Song	Settlement	Sum
Row pair	1,298	231	2,768	**4,297**
Entity-instance-pair	80	34	51	**165**
New entity classification	17	63	23	**103**
Sum	**1,395**	**328**	**2,842**	**4,565**

10.1 Experimental Setup

In this chapter, we again utilize the *Web Tables for Long-Tail Entity Extraction* (T4LTE) gold standard, which we built and published specifically for evaluating entity expansion tasks.[2] It is built with the 2014 release of DBpedia [Lehmann et al., 2015] as the target knowledge base, and includes annotations for the three classes GridironFootballPlayer (GF-Player), Song[3], and Settlement. The web tables included in T4LTE were selected from the English-language relational tables subset of the 2012 WDC web table corpus. We have previously introduced and extensively described T4LTE in Section 8.1.1.

Table 10.1 provides an overview of the number of labels in T4LTE. Creating this dataset was rather laborious, as we labeled 4,297 matching row pairs, 165 entity-instance-pairs and 103 new entity classifications.

When evaluating the LTEE pipeline using the T4LTE gold standard, we were able to achieve an F1 score in the task of finding new entities of 0.83 (see Chapter 8). When running the pipeline on the whole web table corpus, we were able to add 14 thousand new gridiron football players and 187 thousand new songs to DBpedia, an increase of 67% and 356% respectively (see Chapter 9).

[2] http://webdatacommons.org/T4LTE/

[3] The class Song also includes all entities of the class Single.

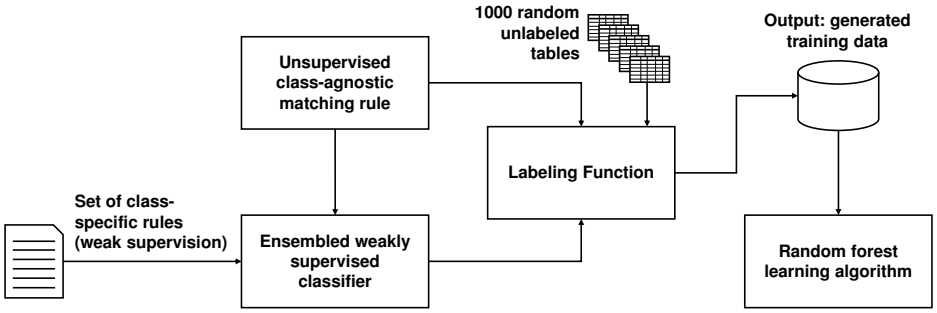

Figure 10.1: Overall methodology for introducing weak supervision using sets of class-specific bold matching rules. We ensemble the bold class-specific rules with a class-agnostic unsupervised matching rule to create a labeling function. Using this labeling function and a set of random web tables, we bootstrap the training of a supervised matching model using the random forest algorithm.

10.2 Methodology

This section describes our approaches for the task of reducing labeling effort using weak supervision. The overall methodology is illustrated in Figure 10.1.

We first introduce as a baseline two unsupervised class-agnostic matching rules for row clustering and new detection. These rules exploit class-agnostic similarity scores and aggregates them using a weighted average.

We then introduce an approach that exploits user-provided class-specific rule sets as weak supervision. The rules are bold by design, which is why we ensemble them with the unsupervised matching rule to derive weakly supervised labeling functions for both row clustering and new detection.

The unsupervised matching rules and the ensembled weakly supervised labeling functions can both be used within the row clustering and new detection components of the LTEE Pipeline directly. We additionally introduce an approach that exploits these methods as labeling functions to bootstrap a supervised learning algorithm. This is done by using a set of unlabeled web tables to label training pairs for both row clustering and new detection. The labeled data is then used to train random forest classifiers to be used in our pipeline.

10.2.1 Unsupervised Class-Agnostic Matching Rules

We suggest two unsupervised matching models in the form of matching rules that aggregate using a weighted average the individual scores generated by the features described in Sections 8.2.2 (row clustering) and 8.2.4 (new detection) of Chapter 8. To be used in a rule, all features must produce scores that are normalized and class-agnostic, as such we use the class-agnostic feature sets.

We determine the weights of the rules by assigning, based on our own judgement, importance factors from 4 to 1 to the individual features. The weight of a fea-

ture is equal to it assigned factor normalized by the sum of all factors. For the row clustering rule, we assign a factor of 4 to LABEL, 2 to BOW and ATTRIBUTE, and 1 to PHI, IMPLICIT_ATT and SAME_TABLE. For new detection we assign a factor of 4 to LABEL, 3 for BOW and ATTRIBUTE, 2 for TYPE and IMPLICIT_ATT, and 1 for POPULARITY.

The rules determine whether a pair matches or not using a fixed threshold, simply set at 0.5 for both rules. The absolute distance of a computed weighted average from the threshold constitutes the confidence of a matching decision.

10.2.2 Class-Specific User-Provided Matching Rule Sets

Humans often possess general knowledge about which conditions need to be fulfilled for entities of a certain domain to clearly match or clearly not match. Based on this observation, we suggest as weak supervision a set of class-specific user-provided bold rules that classify a given candidate pair as matching or non-matching. They can codify obvious knowledge, e.g. that a settlement can not be in two different countries, or non-obvious knowledge, e.g. that only one unique football athlete can be drafted in the same year with the same pick number.

The rules consist of conjuncts of attribute tests, expressed using the schema of the knowledge base. We only require rules to be somewhat accurate, regardless of their coverage. This makes it simple to identify suitable rules and is the reason why we term these rules as bold. For our experiments, we created per class four rules. For **GF-Player** we came up with two matching and two non-matching rules:

$$(\text{draftYear} = \text{Equal}) \wedge (\text{draftPick} = \text{Equal}) \rightarrow \text{Match} \tag{10.1}$$

$$(\text{LABEL} = \text{Equal}) \wedge (\text{birthDate} = \text{Equal}) \rightarrow \text{Match} \tag{10.2}$$

$$(\text{draftYear} = \text{Unequal}) \rightarrow \text{Non-Match} \tag{10.3}$$

$$(\text{draftPick} = \text{Unequal}) \rightarrow \text{Non-Match} \tag{10.4}$$

For **Song** we also came up with two matching and two non-matching rules:

$$(\text{LABEL} = \text{Equal}) \wedge (\text{artist} = \text{Equal}) \wedge (\text{releaseDate} = \text{Equal}) \rightarrow \text{Match} \tag{10.5}$$

$$(\text{LABEL} = \text{Equal}) \wedge (\text{artist} = \text{Equal}) \wedge (\text{album} = \text{Equal}) \rightarrow \text{Match} \tag{10.6}$$

$$(\text{artist} = \text{Unequal}) \rightarrow \text{Non-Match} \tag{10.7}$$

$$(\text{releaseYear} = \text{Unequal}) \rightarrow \text{Non-Match} \tag{10.8}$$

Finally, for **Settlement** we have three matching and one non-matching rule:

$$(\text{country} = \text{Equal}) \wedge (\text{postalCode} = \text{Equal}) \rightarrow \text{Match} \tag{10.9}$$

$$(\text{LABEL} = \text{Equal}) \wedge (\text{isPartOf} = \text{Equal}) \rightarrow \text{Match} \tag{10.10}$$

$$(\text{LABEL} = \text{Equal}) \wedge (\text{postalCode} = \text{Equal}) \rightarrow \text{Match} \tag{10.11}$$

$$(\text{country} = \text{Unequal}) \rightarrow \text{Non-Match} \tag{10.12}$$

The effort spent creating these rules is minuscule compared to manually labeling the correspondences in the gold standard. While for each class we created only 4 rules, they are intended to substitute 1,395, 328, and 2,842 labels for the classes GF-Player, Song, and Settlement respectively.

To apply a rule, we exploit the equal and unequal scores generated by the `ATTRIBUTE` and `IMPLICIT_ATT` features, as described in Subsections 8.2.2 and 8.2.4. A rule fires, when all equality and non-equality tests within the rule have corresponding equal and unequal scores higher than zero. From these scores we then derive for each rule firing a confidence score, which equals the product of all scores used within the rule. We also use the `LABEL` feature and the data-type-specific equivalence functions to allow attribute tests on labels.

10.2.3 Ensembling a Weakly Supervised Labeling Function

As the rules fire only when certain conditions are met, the set of rules is not exhaustive and only covers a subset of compared pairs. We therefore ensemble the rules with the unsupervised matching model described above through averaging. Given a compared pair, we first check how many rules fire. If no rule fires, we simply return the output of the unsupervised model. If multiple rules fire, which is possible as the rules are not mutually exclusive, we consider only the rule with the highest confidence, preferring negative rules in case of a tie. If the confidence of this rule is higher than the confidence of the output of the unsupervised model, the outputs of both are averaged and returned. Otherwise, if the confidence of the fired rules is lower than the unsupervised model, we simply return the output of the unsupervised model.

10.2.4 Bootstrapping Random Forests Using a Labeling Function

In our experiments, we, on the one hand, directly apply both the unsupervised rules and the weakly supervised labeling functions as models within our pipeline. On the other hand, we can use either method to create a pool of labeled, possibly noisy, pairs and use them to train a supervised machine learning model. We suggest in this section an approach to bootstrapping a supervised machine learning model using the random forest algorithm.

Labeling Pairs

We employ both, the unsupervised rule and the weakly supervised ensembled labeling function to label row pairs and entity-instance-pairs derived from 1000 randomly selected web tables as matches or non-matches. Additionally, the labeling functions assign confidences to the labeled pairs, using the confidence scores returned by the underlying method. Using these labeled pairs, we train a supervised machine learned model, which is then used within the pipeline.

To derive pairs to be labeled, we employ label-based blocking using Lucene for both row clustering and new detection. We additionally include random pairs to be labeled, for row clustering as many as there are positive pairs, and for new detection 8 random instances selected from the knowledge base from within the same class of an entity or its parent classes. As we will see in the following section, this leads to 1.99 million row pairs and 1.36 million entity-instance-pairs selected to be labeled.

For row clustering, we use the confidence scores to additionally perform correlation clustering. If after clustering, the two rows within a pair labeled as a match are placed in different clusters, the pair is discarded. A pair is also discarded from the training pool, if it was labeled as a non-match, but was then placed in the same cluster during clustering.

When bootstrapping for new detection, we need a set of row clusters from which we can create entities. Using these entities, we can then generate entity-instance-pairs and label them using our labeling functions. To create the clusters, we use the row clustering models trained by bootstrapping from a labeling function of equal supervision. E.g., when bootstrapping a supervised new detection model using the unsupervised rule as a labeling function, we use the clustering method also trained using bootstrapping from the unsupervised rule to create row clusters.

For new detection, multiple entity-instance-pairs for the same created entity could be labeled by our labeling functions as matching. A created entity should however only match at most one knowledge base instance. As such, we only include the entity-instance-pair with the highest confidence as a positive training example, discarding the rest.

Training a Machine Learning Model

Given the labeled pairs, we train models using the random forest algorithm. It uses bagging (bootstrap aggregation), where a number of trees are each grown on different samples of the training data.[4] To use the forest for prediction, the outcomes of the individual trees are averaged. Additionally, more randomness is introduced while growing trees, by randomly limiting the number of features available for each split. Trees are not pruned and fully grown by default.

We use random forest, as it is an expressive and popular general-purpose algorithm, which we have already used in the context of entity expansion. However, we also believe it has properties that are especially useful when training a model form noisy training data. We elaborate on this in the discussion in Subsection 10.4.3.

The random forest algorithm, as any machine learning algorithm, has a number of hyperparameters that can be tuned. However, we lack a validation set to perform hyperparameter optimization. Tuning the parameters on the pool of noisy labeled training examples, even when using cross-validation or a holdout set, is likely to yield hyperparameters that overfit and replicate the labeling function. As such,

[4]Samples usually have the same size as the original training pool. However as sampling is done with replacement, i.e. a sampled pair is returned to the pool, the samples differ in their composition.

any hyperparameter choices must be justified theoretically and conceptually. We overall adjust four parameters from the default choice:

Reformulation as a Regression Problem: we reformulate the problem from classification to regression, where the confidences of the labeled pairs are converted into numeric values. For a matching pair, the value of pair used for training equals its confidence score, whereas for a non-matching pair the confidence is multiplied by –1.0. The range of the regression model is therefore –1.0 to 1.0.

We reformulate the problem into a regression problem because we believe that the classification method will not fully exploit the information present among the confidence scores in the training data. A random forest grown as a classifier takes the confidences only into account as weights when sampling for bagging, and not when actually growing the tree. Given for example a leaf with two pairs, where both are labeled as matching, but have confidences of 1.0 and 0.1 respectively, the leaf would be considered as 'pure' and the tree would not be grown any further. However, in the regression case, the leaf would be split further, possibly discovering a useful split that allow us to differentiate high from low confidence pairs.

To use the learned regression model for classification, we make use of a threshold. For clustering, the threshold was set at 0.0, whereas for new detection the threshold was learned by optimizing the F1 for new entities on the learning set.

Weighting Pairs With the Square Root of Their Confidence When Bagging: during bagging, samples are drawn from the training pool to build a sample of a certain size. This sampling can be affected by weighting the individual training pairs. Giving all pairs the same weight, would sample high and low confidence pairs with the same probability. However, we believe that higher confidence pairs should be sampled more often than lower confidence pairs. We can assign each pair its confidence as its weight, however this would cause pairs with a moderate confidence to be sample with a very low probability. E.g. a pair with a confidence of 1.0 would be four times as likely to be sampled than a pair with a moderate confidence of 0.25. This could give us a sample that e.g. consists mostly of uninteresting high confidence pairs. We therefore assign each weight the square root of its confidence as a weight, where a pair with a confidence of 1.0 would be approximately two times as likely to be sample as a pair with a confidence of 0.25. This weighting scheme is likely to give us diverse and useful training samples.

Limiting Maximum Tree Depth: individual trees within a random forest are by default fully grown, so that they likely overfit their individual training samples. This is usually compensated by the properties of the random forest algorithm, i.e. bagging and the introduced randomness. Additionally, when using strong supervision, noisy or incorrect training examples are the exception. When training on data that is noisy by design however, overfitting could become a significant issue.

As such, we limit the tree depth to 20 levels. This would still allow more than one million leaves per individual tree. A max depth of 15 would only allow

about 32 thousand leaves, which we consider too low, while a max depth of 25 would allow more than 33 million leaves.[5] Through previous bootstrapping runs, we know that our noisy training pools have a size of about 500 thousand to 1.5 million pairs per class, and, considering, that many branches are likely to stop growing before they reach the maximum depth, a maximum number of leaves of about 1 million is likely an appropriate compromise. Limiting the tree depth is also beneficial as it reduces computation time and memory requirements.

Reducing Size of Bagging Samples: each tree is grown on a sample drawn from the pool of labeled pairs. This, along with the randomness introduced in each tree, reduces the correlation among the individual trees of the forest. By averaging the predictions of many weakly correlated and individually grown trees, a random forest reduces the errors due to variance, i.e. the errors caused by overfitting a potentially noisy learning set [Geurts, 2010]. Reducing this type of error is especially important in our case, as we are dealing with data that is potentially noisy. Therefore, we reduce the sample size from 100% to 25% of the size of the pool. Through a previous run of our bootstrapping approach, we are aware of the number of labeled pairs. A sample size of 25% still amounts to 125 thousand to 375 thousand in the learning set of each individual tree. Reducing the size of the sample is also beneficial as it reduces computation time and possibly also memory requirements.

In Subsection 10.4.2, we discuss the effect of these choices in more details. Finally, per forest, we train 2000 trees, which we consider to be sufficient to average out the errors of the individual trees. All hyperparameters are kept constant for all classes and for both row clustering and new detection.

10.3 Evaluation and Results

In this section, we evaluate on the T4LTE gold standard the approaches described above and compare them to a model trained using strong supervision. As for the latter, the gold standard is also used for training, we apply three-fold cross-validation throughout all experiments. As there are random components in our model, e.g. the randomly selected tables, all results are the average of 5 runs.

We first evaluate the performances of both row clustering and new detection individually. We then evaluate end-to-end entity expansion performance. In all cases, we use the evaluation methodologies described in Chapter 8.

10.3.1 Row Clustering Evaluation

We test a row clustering method by running it on the rows of the testing set. We evaluate the output by comparing it to the row cluster annotations in the gold standard using an evaluation methodology proposed by Hassanzadeh et al. [Hassan-

[5] We considered only multiples of fives as candidates for the maximum tree depth.

Table 10.2: Row clustering performance for runs with various types of supervision.

	Average				GF-Player	Song	Settl.
Method	**PCP**	**AR**	**F1**	$\sigma_{\mathbf{F1}}$	**F1**	**F1**	**F1**
Unsupervised	.73	.86	.781	.003	.90	.62	.83
+ Bootstrapping	.76	.88	.810	.004	.90	.67	.85
Weak supervision	.79	.87	.824	.004	.90	.74	.83
+ Bootstrapping	.82	.90	.859	.003	.92	.81	.84
Strong supervision	.86	.90	.880	.004	.94	.82	.89
+ Bootstrapping	.90	.93	.912	.000	.94	.88	.91

zadeh et al., 2009]. The methodology emphasizes replicating the exact number of clusters in the evaluation set and penalizes if this is not the case. It computes two metrics termed average recall and penalized clustering precision. We also compute the harmonic mean of both metrics using F1. We describe the evaluation methodology in detail in Section 8.2.2.

Table 10.2 shows row clustering performance for different types of supervision. The first two rows show performances when using the unsupervised matching rule alone, while the following two rows show the performances when using the weakly supervised ensembled classifier. The final two rows show the performances when using strong supervision. For each supervision type, we apply and evaluate the underlying method directly on the test set, and then use it as a labeling function to bootstrap a random forest, which we then also apply and evaluate on the test set. For strong supervision, the bootstrapped method resembles a semi-supervised learning approach. We additionally provide the standard deviation for the F1 of the five averaged runs.

From the table, we can see that the difference in average F1 between a model trained using strong supervision, which has an F1 of 0.880, and the unsupervised rule without bootstrapping is about 10 percentage points. We find that using bootstrapping with the unsupervised matching rule allows us to increase F1 by 2.9 percentage points on average, with an increase of 5 percentage points for the class Song. Using the weakly supervised labeling function, we achieve an average F1 score of 0.824, which is an increase of overall 4.3 percentage points from the unsupervised rule. Bootstrapping from the weakly supervised labeling function increases performance by further 3.5 percentage points, achieving an F1 only about two points below strong supervision. When bootstrapping from a model trained using strong supervision, we gain 3.2 percentage points on the strongly supervised model.

We investigated a run, to evaluate how many pairs were labeled during bootstrapping, and for how many pairs the rules fired. The labeling functions were given overall 1.99 million row pairs to label, which were selected either by the

Table 10.3: New detection performance for runs with various types of supervision.

	Average				GF-Player	Song	Settl.
Method	**P**	**R**	**F1**	σ_{F1}	**F1**	**F1**	**F1**
Unsupervised	.85	.80	.804	.000	.82	.73	.87
+ Bootstrapping	.88	.87	.857	.000	.90	.78	.91
Weak supervision	.85	.84	.831	.000	.82	.81	.87
+ Bootstrapping	.88	.89	.877	.006	.90	.84	.89
Strong supervision	.77	.87	.909	.000	.92	.92	.89
+ Bootstrapping	.79	.87	.900	.001	.92	.91	.87

label-based blocker or chosen randomly. Given as labeling function the weakly supervised classifier, 266 thousand pairs were labeled as matches, while 1.72 million pairs were labeled as non-matches. For this output, the user-provided matching rules fire in total 36 thousand times (14% relative to number of pairs labeled as matching), whereas the non-matching rules fire 169 thousand times (10% relative to number of pairs labeled as non-matching).

10.3.2 New Detection Evaluation

To evaluate new detection, we run it on the entities, existing and new, in the testing set, and measure how well we find new entities using precision and recall. Precision equals the proportion of entities returned as new by the method that are actually new, while recall equals the proportion of new entities in the testing set that were returned as new by the method.

Table 10.3 shows new detection performance for runs with various types of supervision, similar to Table 10.2. We find that a model trained using strong supervision outperforms the unsupervised matching rule in F1 by 10.5 percentage points on average, and by 19 points for the class Song. By employing the user-provided rule sets as weak supervision, we are able to increase average F1 by 2.7 percentage points. Bootstrapping is also effective, as it increases average F1 in the unsupervised case by 5.3, and in the weakly supervised case by 4.6 percentage points. Overall, we are able to recover 7.3 of the 10.5 percentage points difference between the unsupervised and the supervised model. Bootstrapping using the strongly supervised model does not work, as overall average performance falls by about one percentage point.

We investigated a run, to evaluate how many pairs were labeled during bootstrapping, and for how many pairs the rules fired. A sum of 1.36 million entity-instance-pairs are chosen to be labeled. When using the weakly supervised labeling function, we find that 26 thousand pairs were labeled as matches, and the remainder as non-matches. Within the ensembled classifier, the user-provided matching rules fire 14 thousand times (54% relative to number of pairs labeled as matching),

Table 10.4: End-to-end performance for runs with various types of supervision.

	Average				GF-Player	Song	Settl.
Method	**P**	**R**	**F1**	σ_{F1}	**F1**	**F1**	**F1**
Unsupervised	.70	.74	.693	.000	.76	.53	.79
+ Bootstrapping	.74	.82	.746	.005	.81	.61	.83
Weak supervision	.72	.79	.737	.005	.76	.65	.80
+ Bootstrapping	.75	.87	.791	.005	.82	.74	.82
Strong supervision	.79	.90	.835	.004	.88	.78	.84
+ Bootstrapping	.79	.93	.853	.000	.89	.84	.83

whereas the non-matching rules fire 174 thousand times (13% relative to number of pairs labeled as non-matching).

10.3.3 End-To-End Evaluation

We will now evaluate a full run of the pipeline using our suggested weak supervision approaches. As input, all methods receive the same set of rows and output a set of created entities determined to be new. We compare this output with the actual new entities annotated in the gold standard. As this runs row clustering and new detection sequentially, the errors of the methods tend to accumulate and reduce overall end-to-end performance, as we have discussed in Chapter 8.

To evaluate how well new entities were found, we utilize precision and recall. To compute precision, we determine the proportion of entities returned as new that are correct. An entity is only correctly new, if its cluster includes the majority of the rows of a new cluster in the gold standard, and these rows at the same time form the majority within the entity's cluster. Recall is the fraction of new entities in the gold standard for which a correct new entity was returned.

Table 10.4 shows end-to-end performance for different types of supervision similar to Table 10.2. The model trained using strong supervision achieves an average F1 of 0.835, while the performance of the unsupervised method without bootstrapping is the lowest, with an F1 of 0.693. The performance achieved by our weakly supervised method using bootstrapping is 0.791. Therefore, we find that we are able to achieve a performance much closer to that when using strong supervision, and much better than a simple unsupervised matching rule. Out of the 14.2 percentage points gap between the strongly supervised and the unsupervised run, we are able to recover 9.8 points. The effect is especially large for Song, where we are able to recover 21 percentage points in F1 from the 25 points difference between the supervised and the unsupervised model. As a result, we can successfully perform long-tail entity extraction with considerably reduced labeling effort.

The user-provided rule sets have a positive impact on performance, increasing F1 by 4.4 percentage points compared to the unsupervised model. Bootstrapping

also increases average F1 by about 5.3 and 5.4 percentage points for the unsupervised and weakly supervised runs respectively. Bootstrapping also has a positive impact when used with strong supervision, where it improves performance by 1.8 percentage points.

When investigating the performance per class, we notice that precision is continuously lower than recall. For the method bootstrapped from weak supervision, GF-Player and Settlement have e.g. precisions of 0.70 and 0.74, with recalls of 1.00 and 0.93 respectively. This is caused by bad clustering, primarily for existing entities, which are then classified as new by the new detection, thereby reducing only precision.

To investigate the issue further, we look at one of the five runs. We find that for GF-Player there are in total 17 genuine new clusters annotated in the gold standard, which were all correctly clustered and classified as new by the pipeline, hence a recall of 1.0. However, existing entities were not clustered as well, as 5 clusters were created that could not be assigned to any entity. These clusters should have been filtered out, but were all determined to be new, considerably reducing entity expansion precision. For Settlement, there are 23 genuine new clusters annotated in the gold standard, 21 one of which were correctly clustered and classified as new. The pipeline however generated 20 clusters too many, the majority of which contain rows of existing entities. Out of these bad clusters 8 were classified as new, reducing entity expansion precision considerably.

This shows that errors in the pipeline accumulate and that there is a need for an additional component in the pipeline that detects and filters out bad clusters, especially those that are created from rows that describe existing entities. While the observed pattern does not exist for class Song, it is because it suffers from bad clustering for new and existing clusters, leading to both lower recall and precision. As such, it would in fact also benefit from a filtering component for bad clusters.

10.4 Discussion

In this section, we will discuss some of the choices made in regard to our methodology and highlight what their effects are. We will first show the importance of ensembling with an unsupervised model and how this actually enables the use of bold class-specific matching rules in the first place. We will then discuss the bootstrapping methodology in two parts. First, we will outline the effect of our parameter choices, which we described in Subsection 10.2.4. Second, we elaborate on why we believe that random forests are especially effective for learning a supervised model from training data that is likely noisy.

10.4.1 Importance of Ensembling with the Unsupervised Model

While we focus in this chapter on weak supervision in the form of bold matching rules, this form of supervision is enabled by ensembling the rules with an unsu-

Table 10.5: End-to-end performance for runs that exploit various types of labeling functions.

	Average				GF-Player	Song	Settl.
Method	**P**	**R**	**F1**	$\sigma_{\mathbf{F1}}$	**F1**	**F1**	**F1**
Matching Rules	.43	.06	.095	.000	.00	.14	.14
+ Bootstrapping	.27	.80	.330	.001	.12	.76	.11
Label	.64	.30	.379	.000	.57	.20	.37
+ Bootstrapping	.83	.28	.372	.019	.67	.12	.33
Label + Match. Rules	.67	.31	.403	.000	.57	.27	.37
+ Bootstrapping	.77	.31	.407	.008	.69	.17	.37
Unsupervised	.70	.74	.693	.000	.76	.53	.79
+ Bootstrapping	.74	.82	.746	.005	.81	.61	.83
Weak Supervision	.72	.79	.737	.005	.76	.65	.80
+ Bootstrapping	.75	.87	.791	.005	.82	.74	.82

pervised model in the first place. This is demonstrated by Table 10.5, where we provide end-to-end performances for a set of alternative runs.

First, we use the matching rules on their own and without ensembling. Without bootstrapping, this yields a very low performance, which is not surprising as the rules have very little coverage. With bootstrapping, we see for songs a very large increase in performance, in fact one that outperforms our weakly supervised bootstrapped method. However, this is an anomaly, as for the other classes the performance is still very low.

We then attempt to ensemble the matching rules with an unsupervised model that uses just label features. This has a positive impact on performance, and, more importantly, we can see how ensembling the matching rules improves performance consistently for all classes in the case of bootstrapping. However, the performance is limited, and this seems to be caused by the ineffective label-based matching method with which we ensemble the rules. It is interesting to note, that the matching rules for the class Song are not as effective as they were without the label-based model. This shows how an ineffective unsupervised model can actually reduce the positive impact of the matching rules.

Unlike the label-based model, the unsupervised model is already quite effective on its own. By ensembling the class-specific matching rules with it, we can create effective labeling functions, that are able to bootstrap effective classifiers. It is also interesting that when combined with the label-based models, the rules achieve an increase of just 2.80 percentage points in F1, however when combined with the unsupervised model, they achieve an increase of 4.50 points. This shows that with a better unsupervised model, the rules have a higher positive effect.

Table 10.6: End-to-end performance for bootstrapping when varying the underlying machine learning method.

	Average				GF-Player	Song	Settl.
Method	**P**	**R**	**F1**	$\sigma_{\mathbf{F1}}$	**F1**	**F1**	**F1**
Classif. (default)	.73	.85	.768	.005	.78	.69	.83
Classification	.74	.84	.772	.004	.82	.67	.83
Regression	.75	.87	.791	.005	.82	.74	.82

Table 10.7: End-to-end performance for bootstrapping when varying sampling weighting during bagging.

	Average				GF-Player	Song	Settl.
Sampling Weighting	**P**	**R**	**F1**	$\sigma_{\mathbf{F1}}$	**F1**	**F1**	**F1**
Equal for all pairs	.72	.88	.778	.000	.79	.75	.80
Confidence score (CS)	.74	.85	.774	.006	.82	.73	.78
Square root of CS	.75	.87	.791	.005	.82	.74	.82

The unsupervised model not only provides full coverage to our labeling function, it also allows us to require that the rules be only somewhat accurate. This is possible, as we use the unsupervised model to average out the biases within the individual matching rules. This is especially important when it is difficult to find effective matching rules. This is the case for settlements, where we find that the number and density of attributes in the web tables are too limited for schema-based matching rules. However, using the unsupervised model, we are still able to achieve an acceptable performance for settlements, lacking behind strong supervision by only two percentage points in F1.

Improving either the unsupervised model or the ensembling method would likely increase the effectiveness of our method. In Subsection 10.5.3, we extensively discuss the related work in regard to unsupervised entity matching, highlighting how we could improve the unsupervised matching rule.

10.4.2 Impacts of Bootstrapping Parameter Choices

In Subsection 10.2.4, we outlined our choices regarding the design and the parameters of the machine learning algorithm used during bootstrapping. In this subsection, we will discuss the impact of these choices. This is done by testing alternative runs of the classifier bootstrapped from the weakly supervised labeling function. In each run, we alternate one of the parameter choices made, while keeping the others fixed. Tables 10.6 to 10.9 show the end-to-end performances of these runs.

We can first of all see, that no matter the choice of parameter, the performance

Table 10.8: End-to-end performance for bootstrapping when varying maximum tree depth within a trained random forest.

	Average				GF-Player	Song	Settl.
Maximum Depth	**P**	**R**	**F1**	$\sigma_{\mathbf{F1}}$	**F1**	**F1**	**F1**
10	.72	.84	.764	.004	.81	.69	.80
15	.72	.86	.771	.004	.81	.70	.80
20	.75	.87	.791	.005	.82	.74	.82
25	.74	.87	.789	.009	.81	.72	.83
30	.73	.86	.773	.007	.80	.69	.83
No maximum depth	.73	.85	.764	.006	.79	.68	.82

Table 10.9: End-to-end performance for bootstrapping when varying bagging sample size.

	Average				GF-Player	Song	Settl.
Sample Size (%)	**P**	**R**	**F1**	$\sigma_{\mathbf{F1}}$	**F1**	**F1**	**F1**
10	0.74	0.88	0.786	0.003	0.81	0.74	0.81
25	0.75	0.87	0.791	0.005	0.82	0.74	0.82
50	0.75	0.86	0.785	0.003	0.81	0.74	0.81
75	0.74	0.86	0.785	0.004	0.81	0.74	0.81
100	0.75	0.86	0.786	0.003	0.81	0.73	0.82

for many runs is at least 0.77 or 0.78. So while the parameter choices are important for achieving an additional one to two percentage points increase in end-to-end performance, our method works even when the parameter choices are made naively.

Table 10.6 shows end-to-end performance when using regression instead of classification. It also shows a run using classification with default hyperparameters, i.e. no depth limit and full-sized bagging sample. We denote this run as *Classif. (default)*.[6] As we have anticipated in Subsection 10.2.4, using regression instead of classification has a positive impact on performance. The classifier, by ignoring the confidence of training examples while splitting, ignores their noisy nature. To generalize effectively from noisy training data, the confidence scores must be taken into account in a form that is reflected in the predictions, and not just during sampling for bagging.

Table 10.7 shows performance when changing the weighting method during bagging. Both weighting all training pairs equally and directly with the confidence scores, are in our opinion naive approaches to the issue. As such, it is understandable that they achieve a lower performance. Using the square root of the confi-

[6]We were unable to perform a run using regression and default hyperparameters, as we ran out of memory, even when using a machine with 500 GB of RAM.

dence scores ensures that all training pairs have a good chance of being included in a sample. while still giving more prominence to high-confidence pairs, and lower significance to low-confidence pairs.

In regard to the maximum tree depth, we see a clear pattern in Table 10.8. Not limiting the tree depth enough leads to low performance, likely due to overfitting, while limiting the tree depth too much leads to low performance, likely by excessively reducing the expressiveness of the trees. In fact, this parameter has the highest impact on performance. As such it might make sense to choose a tree growing algorithm that limits depth by design, e.g. through pruning. However, we also believe that the chosen value for maximum depth, is the only reasonable choice.[7] A maximum depth of 15 would only allow a maximum of about 33 thousand leaf nodes, while a maximum depth of 25 would allow about 33 million leaf nodes. When considering the number of training pairs used for bootstrapping (see Sections 10.3.1 and 10.3.2), only a depth 20, which allows about 1 million leaf nodes, is a reasonable choice.

Table 10.9 shows the effect on performance when alternating sample size during bagging. While a sample size of 25% has the best performance, there is no clear pattern between performance and sample size. When considering however that reducing the sample size reduces correlation between trees, possibly leading to lower variance and higher performance, while at the same time reducing computation time and memory requirements, allowing us e.g. to train more trees, it is still a reasonable choice.

It is important to add, that both the reduced sample size and the limited depth are both helpful in ensuring that training the random forest is feasible in regard to memory. When setting both parameters to default, i.e. a sample size as large as the number labeled pairs, and trees with unlimited depth, we were unable to train a random forest when using regression, even given more than 500 GB of memory.

10.4.3 Ability of Random Forests to Learn from Noisy Data

In our methodology, we train a supervised machine learned model on labeled data that is possibly noisy. Ensuring that the trained model generalizes without overfitting the training examples is therefore important. We believe that we are successful at this task, as we are able to train a supervised machine learning model that outperforms the labeling function form which it was bootstrapped. In this section, we will discuss what possibly enables us to so.

First, the trained random forest has a different feature set than both, the unsupervised rule and the bold user-provided matching rule sets. The unsupervised rule, as it is class-agnostic, does not have any features specific to the schema of a class, i.e. its properties. On the other hand, the matching rules almost exclusively make use of class-specific properties. The bootstrapped random forest combines both and is therefore possibly able to generalize without overfitting either the un-

[7] Considering only maximum tree depths that are multiples of 5.

supervised class-agnostic rule or the user-provided class-specific rule sets.

However, we believe that the random forest algorithm itself is highly effective at learning from noisy training data. This is due to it using bagging and introducing randomness within each tree. The random forest does not overfit the noisy pool, because it never trains directly on the noisy pool. It only trains on samples drawn from that pool. While individual trees might overfit, this overfitting is likely averaged out when aggregating the output of the trees. The randomness within each tree additionally restrict the algorithm from always using the same features when splitting, forcing it to always generalize using a different set of features.

There might be additional aspects, also in the context of using random forests, that enable us to prevent overfitting the noisy training data and outperforming as a result the labeling functions. By limiting the tree depth for example, we do not allow the random forest algorithm to fully grow its trees, and as such prevent them from overfitting their own training sample. Also, the way we weight training examples during bagging reduces the likelihood of overfitting low-confidence training pairs, while ensuring that interesting training pairs are still included in the sample a tree is trained on.

10.5 Related Work

Weak supervision approaches exploit supervision at a higher abstraction or that is noisier in nature to efficiently generate a large number of training examples, even if those are of a lower quality [Ratner et al., 2017, Ratner et al., 2016]. This includes letting non-experts generate labels through crowdsourcing, using distant supervision, or employing rules and heuristics for labeling data.

There exist other approaches to reducing required manual effort spent by experts for the purpose of supervision [Han et al., 2011]:[8]

- *Semi-Supervised learning* methods use a small set of labeled and a larger set of unlabeled examples to train a model. This includes for example co-training and self-training [Mihalcea, 2004], which train models on data that they labeled themselves, using initially a small number of high-quality seed examples. The method presented in this chapter that bootstraps from a strongly supervised model resembles semi-supervised learning.
- *Active learning* approaches reduce labeling effort by querying a user to label only examples that are chosen to provide the most information when labeled, thereby using effort spent by experts more efficiently [Settles, 2012].
- *Transfer learning* approaches reduce supervision effort required for learning given a certain target domain or task by using knowledge gained for a different source domain or task [Pan and Yang, 2010].

[8]The authors of Snorkel provide an extensive overview of weak supervision, comparing it to other approaches to reducing supervision effort: https://www.snorkel.org/blog/weak-supervision.

These approaches to reducing effort differ conceptually from weak supervision. While they work towards reducing the amount of manually labeled data required, querying experts more efficiently, or exploiting knowledge from a different domain or task, they still often rely on some form of manually labeled high-quality training examples. Weak supervision approaches however differ, as they attempt to substitute supervision in the form of high-quality training examples with supervision that is noisier or more abstract in nature, thereby working towards reducing the reliance on manually labeled training examples completely.

However, the distinction is not always clear. Semi-supervised learning approaches could be categorized as weak supervision approaches. While they start with high-quality labeled training example as seeds, they essentially generate training data that is potentially noisy. There exist for example approaches that are described by the authors as weak supervision, which could however be understood as semi-supervised approaches. This applies to Snuba [Varma and Ré, 2018], that uses a small set of high-quality labeled data to derive heuristics which are then used to generate a larger set of noisy training data. This could be seen as semi-supervised learning, whereas the authors present it as a weak supervision approach.

In the remainder of this section, we will compare our work to three areas of related work. First, we will introduce works that use distant supervision and outline why distant supervision can not be used for long-tail entity extraction. We will then describe weak supervision approaches that use user-provided labeling functions or constraints as supervision. We will compare our approach to them and outline the contributions of our approach. Finally, we will look at related work in the area of unsupervised entity matching and highlight how our unsupervised model could be improved.

10.5.1 Distant Supervision

One method of weak supervision is distant supervision, where a knowledge base or any other external resource is used to train a supervised machine learned model. While originally applied in the context of relation extraction from text [Mintz et al., 2009], it has been used for the task of augmenting a knowledge base from semi-structured web data, including web tables [Dong et al., 2014a,Lockard et al., 2018]. Regarding identity resolution, Bizer et al. [Bizer et al., 2019] e.g. make use of semantic annotations extracted from 43 thousand e-shops to distantly supervise a deep neural network for product entity matching.

Distant supervision approaches rely on the assumption that the relationships that exist among the data to be labeled are also reflected in the external source. E.g., for relation extraction, when two entities that have a relation in the knowledge base appear in one sentence, it is assumed that this sentence described this relation [Mintz et al., 2009]. By definition, long-tail entities are not part of knowledge bases, therefore, no relationship or other knowledge about them exist in knowledge bases. Especially for the task of new detection, we need the external source to explicitly state that this entity has no corresponding instance in a knowledge base.

This is unlikely to be the case, which is why we do not consider distant supervision to be an option for long-tail entity extraction.

10.5.2 User-Provided Labeling Functions and Constraints

Ratner et al. [Ratner et al., 2016] introduce the *data programming* paradigm, where any weak supervision strategy, including domain heuristics and distant supervision, can be codified into user-provided individual labeling functions. The authors focus on denoising noisy and conflicting labels, by assigning accuracies to the labeling functions using a generative algorithm. They evaluate their approach on a number of machine learning tasks and find that using their approach gives an increase of 1 to 4 percentage points in F1 when compared to an approach that uses the labeling functions in a manually tuned order. The authors do not compare their performance to an unsupervised model.

Snorkel is a system that enables the use of weak supervision based on the data programming paradigm [Ratner et al., 2017]. Snorkel Drybell adapts Snorkel to exploit diverse organizational knowledge resources. Its effectiveness is evaluated in a large-scale case study at Google [Bach et al., 2019].

Throughout all three works [Ratner et al., 2016, Ratner et al., 2017, Bach et al., 2019], the number of labeling functions used per task is between 7 and 147. The average number of labeling functions per task is 35.5, averaged over 12 tasks.

Shen et al. [Shen et al., 2005] introduce *constraint-based entity matching*, where they suggest a probabilistic framework and a generative model within which domain-specific constraints can be exploited to perform entity matching. The introduced constraints are of a broad-variety, and not limited to a specific format. They can also make use of specific characteristics of the datasets, between which entity matching is performed. The authors view their work in the domain of unsupervised entity matching methods, as they use the constraints as relaxation labeling within a generative model that performs entity matching, and do not provide an approach to generate training data for a supervised machine learning method. They achieve for the two evaluated datasets increases of 13 and 3 percentage points in F1 using seven and three constraints respectively.

In essence, our approach has the same primary effect as both, data programming and constraint-based entity matching. It enables the use of supervision in a more abstract form than high-quality matching and non-matching pairs. However, our approach has the following contributions over the related work:

1. **Easy-to-create bold matching rules instead of complex black-box labeling functions:** in the data programming paradigm the labeling functions are by design black-boxes [Ratner et al., 2017, Bach et al., 2019]. They consist of actual programs, which could be complex in their nature or be dependent on external resources. For example, the labeling functions could use named entity recognition and topic models that require maintenance [Bach et al., 2019]. Shen et al. e.g. use constraints that are dataset-specific [Shen et al.,

2005]. We limit the format of the user-provided rules to conjuncts of simple attribute tests using the schema of the knowledge base. We believe this facilitates the creation of rules for supervision, and as such is an improvement over the state of the art.

2. **Requiring only a small set of rules, by ensembling with an unsupervised model**: Ratner et al. [Ratner et al., 2016] report coverage numbers for their weak supervision approach. They achieve an average coverage of 36%. We overcome low coverage by ensembling the user-provided matching rule sets with an unsupervised model. Additionally, the unsupervised model provides not only full coverage, but by averaging with it we are able to reduce the negative impact of possibly biased or even inaccurate rules.

Without these properties, weak supervision, e.g. using the data programming paradigm, is difficult to apply to entity expansion for a knowledge base with hundreds of classes. Providing a larger number of manually created black-box labeling functions for each class of the knowledge base is unlikely to be feasible. In our case however, a base performance is ensured by ensembling with an unsupervised model, reducing supervision effort spent on every single class. Additionally, the fixed bold format of the rules does not only allow easy creation, but in can even potentially allow automatic rule mining form the knowledge base, reducing supervision effort even further.

10.5.3 Unsupervised Entity Matching

While the primary contribution of this chapter is the possibility of introducing class-specific supervision with little effort, the unsupervised matching rule plays an important role in enabling our approach. Following, we will briefly introduce some related work regarding unsupervised entity matching, comparing our method to it, and suggesting possible improvements to our approach.

An early approach to unsupervised entity matching is introduced by Christen [Christen, 2008], where they use a two-step classification approach. First, they derive a set of seeds, then, using the seeds, they train a supervised classification model. To derive seeds, they use the distance between the feature score vectors of candidate pairs, to both, vectors of a perfect match (all scores 1.0) or a perfect non-match (all scores 0.0). They then select a certain top-k_M for matching and a different top-k_N non-matching seeds, where k_M and k_N are determined through a specific ratio. When deriving seeds, all features were weighted equally. Christen then suggest adapted versions of supervised classification algorithm: seeded k-NN and seeded iterative SVM. In both, they iteratively increase the size of the training data, starting with a model trained using the seeds. They achieve a performance generally 10 to 20 percentage points lower than the F1 of a supervised algorithm.

Su et al. [Su et al., 2010] introduce an approach, where they use a weighted average of feature scores to find training pairs. Unlike in our case, they learn the weights using a preliminary set of seeds. These seeds, are determined using an

iterative algorithm, starting with easy non-matching pairs. To use the weighted average function for classification, the authors use a threshold, which they set by hand. The training pairs returned by this weighted average function are then used to train an SVM model. They evaluate their method on five datasets, and generally achieve very good results, all above 0.92 in F1. However, they compare their performance to a supervised model only for one dataset, where they achieve 2.8 percentage points in F1 below the supervised model.

Kejriwal and Miranker [Kejriwal and Miranker, 2015] introduce a heuristics-based method that creates training data for a supervised learning algorithm. In their heuristics, they use bag-of-words vectors, similarities based on TF-IDF, and a top-k approach, where they set k to 500. They use the labeled data to train an iterative SVM. They achieve an F1 score on average 10 percentage points below a supervised algorithm, when evaluating their method on 10 datasets. Unlike in our approach, the features are not given, but actually created as part of the approach. Additionally, the method also includes unsupervised schema matching.

Jurek et al. [Jurek et al., 2017] introduce an approach based on ensembling multiple self-learned classifiers. Each individual classifier is trained using a different feature set, where in each set, the similarity metric used to compare a certain attribute is changed. Each classifier is trained using its own seeds, where the seeds are derived by iterating between seed selection and assigning weights to features, until the outcome converges. The first set of seed is selected using top-k approach and a high threshold. During iterative seed selection, features are weighted by how well they are able to distinguish matching and non-matching seeds of the previous iteration. Given the seeds, classifiers are then trained using an iterative SVM algorithm, giving us a set of self-trained classifiers, all using different feature sets. These self-learned classifiers are then ensembled. However, before ensembling, classifiers which conflict very often with the others are filtered out. The authors compare their methods to a fully supervised model on four datasets and achieve a performance below the supervised model by 2 to 9 percentage points in F1. They perform entity matching on datasets that have a small number of sources (mostly two), and where the schemata of all sources are equal. Like the approach by Kejriwal and Miranker, the features are not given and are created as part of the unsupervised method.

In summary, these approaches are similar to each other and to the approach we suggest: using a threshold or a top-k approach, seeds are derived, which are then used to train a supervised model. Two out of the four works, like us, use a weighted average to label data, however they learn the weights automatically. Finally, all works make some use of iteration in their approach.

Regarding performance, we achieve with our unsupervised model using bootstrapping 7.0 percentage points in row clustering and 5.2 percentage points in new detection below the average F1 of the supervised model. Our performance is better than that achieved by Christen, somewhat better than that achieved by Kejriwal and Miranker, and comparable to that that achieved by Jurek et al. Only Su et al. seem to achieve much better results with their approach, however they only compare

their method to a supervised model for one dataset.

Considering this summary on related work, our unsupervised approach could possibly be improved using the following approaches:

- **Ensembling:** while we already ensemble an unsupervised matching model with a class-specific user provided rule set, we could use ensembling to improve the unsupervised model itself. One approach could be, to ensemble multiple unsupervised rules, where in each rule, one feature is left out, or the weights of features are otherwise alternated.
- **Dynamic thresholding:** the majority of approaches use either arbitrarily set thresholds, or a fixed top-k to select seeds. We could improve our unsupervised rule, by finding a method to dynamically find a threshold, based on the actual distribution of the data, similar to Otsu's Method [Otsu, 1979].
- **Weighting:** one possible way of improving our method is to choose weights automatically, e.g. by how large their distinguishing impact is [Jurek et al., 2017] or using a set of initial seeds [Su et al., 2010].
- **Iteration:** iteration is used throughout all related work on unsupervised matching discussed above. Moreover, the method in this work that bootstraps from strong supervision essentially uses iteration and it has the best end-to-end performance. Introducing iteration in our approach could likely have a positive impact on performance of the unsupervised model.

10.6 Summary

In this chapter, we investigated the possibility of reducing the effort spent on manually labeling training data for the task of augmenting knowledge bases with long-tail entities from web tables. For this, we introduce and evaluate a weak supervision approach that exploits more efficient supervision at a higher level of abstraction.

Specifically, we suggest, as an alternative to manually labeling thousands of entity pairs as matching or non-matching, the use of a small set of bold user-provided class-specific matching rules. These rules are built upon properties from the schema of a knowledge base, making them universal and semantically easy to understand. More importantly, these rules require considerably less effort to create. To overcome their likely limited coverage and reduce the impact of possible biases within these rules, we suggest a method to ensemble these class-specific matching rules with a class-agnostic unsupervised matching model. This yields an effective weakly supervised labeling function for long-tail entity extraction.

We introduce an approach to bootstrap a supervised learning algorithm from the weakly supervised labeling function using a randomly selected set of unlabeled web tables. We find that with bootstrapping, we are able to achieve a performance close to that of supervision with manually labeled data and as such, are able to perform long-tail entity extraction with considerably reduced supervision effort.

We believe that our approach has the potential to be improved in various ways. As the class-specific bold matching rules are built using the schema of the knowledge base, they could potentially be mined from the knowledge base itself, reducing supervision effort even further. Additionally, the unsupervised model, which plays an important role in enabling supervision using bold matching rules, can be improved in a variety of ways, which we have, based on the related work, summarized within the chapter. Finally, we suggest and test only one approach for ensembling the user-provided rule set with an unsupervised model. Alternative ensembling methods could have a positive impact on the performance of the ensembled weakly supervised labeling function.

Our weak supervision approach can be highly useful for a variety of tasks. In case where recall is a secondary objective, our approach can be tuned towards precision and used to add highly accurate, albeit fewer, long-tail entities to a knowledge base. The approach can also be used to facilitate generating training data for manual labeling, where experts must only correct labels instead of creating them. This would considerably reduce effort required for manually labeling training data.

We believe that an interesting direction for future work would be combining our approach with active learning. Models bootstrapped using our methodology could be used to select more effective pairs given to the user for labeling at the beginning of the active learning process. This could possibly help alleviate the cold-start problem.

Part IV

Conclusion

Chapter 11

Conclusion

Cross-domain knowledge bases are becoming increasingly useful for a large variety of tasks. Due to the fact that their usefulness increases with their completeness, augmenting knowledge bases with new knowledge is an important task.

A source for this new knowledge could be in the form of web tables, which are relational HTML tables extracted from the Web. We can exploit these web tables to perform external knowledge base completion using data integration methods. This could potentially allow us to enrich knowledge bases with new relations for a versatile set of properties, with new literal values, and with new entities. More importantly, web tables are not limited to a specific topical domain and could therefore allow us to augment a knowledge base with cross-domain knowledge.

However, integrating web tables is challenging. Individual tables are relatively small, while the overall number of tables extractable from the Web is very large. Each table employs its own heterogeneous data model, and the quality of knowledge within web tables is inconsistent, potentially containing noise. To augment a cross-domain knowledge base from web tables, we therefore need data integration methods that are automated, domain-independent, noise-resistant and scalable.

Data integration consists of three tasks: schema matching, identity resolution and fusion. We find that for schema matching there are existing methods that enable a variety of knowledge base augmentation tasks. However, for identity resolution and fusion, we find that existing methods focus solely on the augmentation task of slot filling static non-time-dependent data. They do not enable slot filling time-dependent data, nor do they enable entity expansion.

Existing fusion methods make use of source reliability estimation, e.g. in the form of Knowledge-Based Trust (KBT). These methods allow us to estimate the correctness of extracted values for fusion, which generally is sufficient for fusing static data. However, there also exists time-dependent data, where a value must not only be correct, but also valid given a certain temporal scope. To perform slot filling for time-dependent data, we require time-aware fusion methods that are able to find from a set of conflicting web table values, the value that is first, correct, and second, valid given a certain temporal scope.

Entity expansion enriches the knowledge base with new and previously unknown entities. Existing identity resolution methods for web tables are insufficient for this task, as entities described within web table rows are matched solely to existing entities in the knowledge base. Unmatched rows, which potentially describe new entities not yet part of the knowledge base, remain undisambiguated. As such, we are neither able to determine the exact number of new entities described by web table data, nor which unmatched rows describe the same new entity. The latter is required to create from web table data entity descriptions according to the schema of the knowledge base. To exploit web tables for entity expansion, we need identity resolution methods that disambiguate entities within web tables among each other, in addition to disambiguating them with existing entities in the knowledge base.

In this thesis, we researched data integration methods for slot filling time-dependent data and for entity expansion. Regarding the former, we introduce two time-aware fusion approaches: *TT-Weighting*, which exploits timestamps extracted from the table and its context to estimate the temporal validity of a value extracted from web tables, and *Timed-KBT*, which uses a temporal knowledge base to propagate temporal scopes to web table data and uses these propagated scopes to perform time-aware fusion. Regarding entity expansion, we introduce the *Long-Tail Entity Extraction Pipeline*, the first system to enable the task of automatic long-tail entity extraction from web table data. To ensure that the pipeline is scalable while domain-independent, we additionally introduce a *weak supervision approach for long-tail entity extraction*, which exploits as supervision, in lieu of manually labeled pairs, bold matching rules built using the schema of the knowledge base.

The remainder of this chapter summarizes our work regarding time-aware fusion, long-tail entity extraction, and weak supervision for long-tail entity matching. For each of the three parts, we introduce the task at hand, describe our contributions, and discuss open issues and future work. The last section additionally describes the research impact of our work.

11.1 Time-Aware Fusion

Time-aware fusion is the task of finding, among a set of conflicting values, the value that is, in addition to being correct, valid for a given temporal scope. Time-aware fusion methods are required when performing slot filling for time-dependent data. Not considering the temporal aspect of time-dependent data might lead to inconsistent or outdated data being added to the knowledge base. It could also lead to the rejection of potentially new triples, because candidate values have been classified as incorrect, while they were in fact valid for a certain temporal scope.

Existing Methods

Existing time-aware fusion approaches make use of timestamps extracted from the table and its context. For this, timestamps are assigned to web table columns

and propagated along web table columns as part of a holistic schema matching method [Zhang and Chakrabarti, 2013]. However, using timestamps has its limitations. As web tables lack explicit meta-data and annotations, the relationship between the timestamps and web table values is unclear. While some timestamps might be relevant, many might actually be noise. Additionally, some web tables lack extractable timestamps completely. This sparsity among timestamps additionally increases the difficulty of exploiting them for time-aware fusion.

Summary and Contributions

As part of TT-Weighting, we first present a taxonomy of timestamp types. This taxonomy allows us to introduce methods that consider the locations from which individual timestamps were extracted. To reduce timestamp sparsity, we introduce an approach that propagates timestamp information along these timestamp type by individual web tables values. We then introduce a regression approach, that weights the importance of each timestamp type given a certain property of the knowledge base schema. This would potentially discover a relationship between a property and specific locations of timestamps in and around the web table. In our discussion, we have demonstrated evidence that such relationships exist. With these approaches, we are able to outperform KBT by 5 percentage points in average F1. Existing approaches [Zhang and Chakrabarti, 2013] do not consider the individual locations of timestamps, treating timestamps extracted from different locations equally. The authors also propagate timestamps only along whole web table columns, and not by individual web table values.

With Timed-KBT, we introduce an approach that reduces the dependence on timestamp information for time-aware fusion. Using a temporal knowledge base, we propagate temporal scopes to web table columns by exploiting the overlap of web table data with data in the knowledge base. We are able to outperform both KBT and TT-Weighting by 10 percentage points in average F1. We then combine Timed-KBT with timestamp information by restricting the temporal scopes that can be propagated to web table data to those also described in extractable timestamps. This yields a precision-oriented time-aware fusion method that outperforms KBT and TT-Weighting by respectively 25 and 15 percentage points in average $F_{0.25}$. Existing works [Zhang and Chakrabarti, 2013], including TT-Weighting, relied solely on the existence of timestamps for estimating temporal scopes.

Additionally, we have introduced the Time-Dependent Ground Truth (TDGT). It is built using the schema of and entities from Wikidata, a temporal knowledge base. Within the ground truth, we integrate data from 5 different sources, for 7 topical domains and overall 19 time-dependent properties, covering more than 180 thousand entities with more than one million temporal facts. TDGT could be used for a variety of tasks that make use of the temporal aspect of time-dependent data. We use TDGT to evaluate time-aware fusion performance using the Local Closed-World Assumption.

Open Issues and Future Research

There exist multiple ways in which our research on time-aware fusion methods could be improved or extended. These issues could be the focus of future research.

First, we limit the evaluation of our approaches to temporal scopes that are points in time, and only with the granularity of years. While this is also done in related research [Zhang and Chakrabarti, 2013], future research could extend the evaluation to temporal scopes that are ranges and have a finer granularity.

We also limit our evaluation to the case where the target temporal scope of a slot is known. Again, while this is also done in the related work [Zhang and Chakrabarti, 2013], future research on time-aware fusion could consider a task where the fusion methods derive the temporal scope from web table data to enrich the knowledge base with both new facts and corresponding temporal scopes.

While the Local Closed-World Assumption is useful in estimating fusion precision, in fact we confirm its validity in that regard in Chapter 5, it is potentially less exact when estimating recall. This could especially be the case for time-dependent slot filling, where we count a slot towards maximum recall when among any of its candidates an equal value exists, regardless of the temporal scope of that value. As such, an evaluation using a gold standard should be considered in future work.

In regard to our TT-Weighting approach, we believe that an extension where timestamp types are weighted by web table columns in addition to knowledge properties could be a useful extension. Just like there is likely a relationship between a certain location and a property, there is likely one between a certain location and an individual web table column.

In regard to Timed-KBT, we currently propagate temporal scopes from the knowledge base to web table columns only. While this makes sense, as each column in a web table generally describes similar data, the temporal scope could still differ by individual row. Additionally, a whole table could potentially describe data for one single temporal scope. As such, we could extend Timed-KBT by propagating temporal scopes to rows and tables in addition to columns.

Finally, the lessons learned from both approaches could be combined. When restricting temporal scopes for Timed-KBT, we could take into consideration the timestamp types and their relevance given a certain property of the knowledge base schema. Additionally, we could also consider temporal scope propagation during restriction, which could have a positive effect on recall.

11.2 Long-Tail Entity Extraction

Long-tail entity extraction from web data consists of two subtasks: (1) identifying entities that are not yet part of the knowledge base and (2) compiling their descriptions according to the schema of the knowledge base. To the best of our knowledge, our research is the first to investigate long-tail entity extraction from web data.

Existing Methods

There are two tasks that share similarities to long-tail entity extraction. In set expansion, a set of entities is completed using only a small number of seed entities as input. Set expansion methods are tasked especially with finding entities that have a high similarity to the seeds and are as such typically evaluated using ranked evaluation. In this regard, set expansion is not in line with the task of long-tail entity extraction, which aims to find all possible entities not yet part of a knowledge base, and not only those that share a high similarity to a specific set of seeds. Set expansion methods also by definition do not compile descriptions for new entities, only compiling their labels. While set expansion methods can identify a distinct number of entities to complete a set, i.e. they disambiguate among the new entities, this disambiguation is done solely by the label, and without considering other attributes. Methods therefore do not disambiguate between homonyms.

Emerging entity detection is an NLP task about determining if a certain entity mention in text refers to an entity not yet part of a knowledge base. Unlike set expansion, emerging entity detection methods explicitly consider disambiguation between homonyms. In this regard, emerging entity detection is more in line with long-tail entity extraction than set expansion. However, emerging entity detection is not concerned with compiling descriptions for emerging entities. More importantly, emerging entity detection is not concerned with disambiguating between entity mentions among each other. As such, while it is known which mentions describe long-tail entities, the number of distinct long-tail entities is not identified.

Regarding existing identity resolution methods for web tables, we find that they match rows of web tables directly and solely to existing entities in the knowledge base. As such, these methods, similar to emerging entity detection, can determine if a row describes a long-tail entity without a corresponding instance in the knowledge base, however they can not link rows that describe the same long-tail entity with each other. This prevents us from both, identifying the exact number of unique long-tail entities and compiling their descriptions. For rows that describe existing entities, the links between rows are known automatically, as all rows describing one unique existing entity are linked to its instance in the knowledge base.

Summary and Contributions

We introduce the Long-Tail Entity Extraction (LTEE) Pipeline, the first system that is able to both, identify long-tail entities and compile their descriptions from web table data. It consists of the four components schema matching, row clustering, entity creation and new detection.

Our pipeline enables long-tail entity extraction by performing identity resolution twice. In the row clustering component, we disambiguate between web table rows, where rows that describe the same real-world entity, existing or long-tail, are placed in the same cluster. As a result, the row clusters give us, in the case of perfect entity matching, the exact number of unique entities described in the web

tables. Using the entity creation component, we create for all clusters descriptions according to the schema of the knowledge base, giving us a set of entities extracted and created from a web table corpus.

Identity resolution is performed again in the new detection component, where we disambiguate between the entities created from the web table corpus and the entity instances in the knowledge base. The output of the new detection component is a set of entities, which are classified as new, i.e. they have no corresponding instance in the knowledge base. These entities already have descriptions according to the schema of the knowledge base, and, in case of perfect row clustering, each real-world long-tail entity is at most described by one entity returned by the pipeline. As such, we are able to identify long-tail entities and compile their descriptions, enabling the task of long-tail entity extraction.

We create and publish the Web Tables for Long-Tail Entity Extraction (T4LTE) gold standard. It consists of web tables with row-to-row and entity fact annotations. It also includes attribute-to-property and entity-to-instance correspondences to DBpedia. It allows us to evaluate long-tail entity extraction from web table data, including how well we identify new entities and how well we create their descriptions, but also how well we perform row clustering and new detection. T4LTE can also be used to train supervised models. It contains 532 tables, with annotations for 268 entities, of which 103 are new, and 843 facts. The dataset was created for the three DBpedia classes GridironFootballPlayer (GF-Player), Song and Settlement. The T4LTE gold standard is the first for the task of entity expansion from web tables. It acts as a benchmark when used with our evaluation metrics and our testing folds, which we also provide.

On the T4LTE gold standard, the LTEE Pipeline achieves for both tasks, identifying new entities from web tables and compiling their descriptions, an average F1 of 0.83. Additionally, we evaluate our pipeline by how well it performs related tasks. We compare the performance of the pipeline for reported performances in the related work for the tasks of set expansion, emerging entity detection, column attribute-to-property matching, and row-to-instance matching and find that the performances of the LTEE Pipeline are comparable or better.

Finally, we run the LTEE Pipeline on a large-scale web table corpus and profile the output. We also manually evaluate the accuracy of a sample of new entities returned by the pipeline. We find that we are able to add about 14 thousand new entities with 44 thousand new facts for the class GF-Player, an increase of respectively 67% and 32% when compared to the existing entities in DBpedia. For songs, we are able to add about 187 thousand new entities with 394 thousand new facts, an increase of 356% and 125% respectively. The accuracy of these newly added entities is on average 65%, while the accuracy of their descriptions is 90%.

As part of the profiling, we also extensively describe lessons learned. This includes a discussion on the relationship between the Wikipedia notability criteria and the potential of performing entity expansion from web table data. We also compare the distributions of properties for long-tail entities extracted from the web tables with that of existing entities in DBpedia. Additionally, we compare the

performance of the evaluated sample with the performance achieved on T4LTE and discuss how we could potentially improve the gold standard.

Open Issues and Future Research

The pipeline can first of all be improved by introducing noise detection and filtering. We find that errors compound throughout the components of the pipeline. Especially, we notice that a row clustering output with impure clusters leads to inconsistent entities, which are then incorrectly classified as new, as no corresponding instance is found in the knowledge base. This problem is demonstrated by the fact that recall for new entities found is consistently higher than precision, i.e. we are able to find many of the new entities, but we also classify noisy entities as new (see for example Subsections 8.3.1 and 10.3.3). A filtering component before or as part of new detection could significantly improve the performance and the effectiveness of the pipeline.

Similarly, noise detection is needed to reduce effects of errors during schema matching or entity creation. Considering that web tables are noisy in their nature, it is important to introduce noise detection and filtering into the pipeline.

The classes included in the T4LTE gold standard were chosen to reflect the diverse nature of DBpedia, while also being more difficult than other classes due to a higher occurrence of homonyms. The diversity of these classes is also demonstrated by the vastly different results we achieve in Chapter 9, and the different Wikipedia notability criteria that affect each class. As such, we believe that the classes were chosen well. However, future research should consider more classes to evaluate the LTEE Pipeline. This also applies to the profiling, where more classes would provide more insight into the potential of web tables for entity expansion.

One main limitation of the pipeline is the requirement for manually labeled class-specific training data. This limits the applicability of the LTEE Pipeline for cross-domain entity expansion at web-scale. However, we already researched methods to overcome these limitations using weak supervision, which we summarize in the next section.

11.3 Weak Supervision for Long-Tail Entity Extraction

Weak supervision refers to the task of reducing labeling effort by exploiting supervision that is noisier or more abstract in nature than traditional supervision. Given that the LTEE Pipeline requires class-specific supervision and that knowledge bases have many classes, reducing supervision effort is important to ensure that the pipeline is scalable when performing cross-domain entity expansion.

Existing Methods

There exist a variety of approaches to reducing supervision effort, e.g. in the form of active learning, transfer learning and semi-supervised learning. While these ap-

proaches also aim at reducing labeling effort, they differ conceptually from weak supervision. They work towards reducing the amount of manually labeled data required, e.g. by querying experts more efficiently, or exploiting existing labeled data from a different domain. However, they often still rely on some form of manually labeled high quality training data. Weak supervision approaches on the other hand, by exploiting noisier or more abstract supervision, aim towards reducing the reliance on high-quality manually labeled training examples completely.

There are two prominent approaches to weak supervision. Distant supervision [Mintz et al., 2009] uses the relations within an external source, e.g. a knowledge base, to derive likely noisy training data for a supervised machine learning algorithm. It relies on the fact that the relationships of the data to be labeled are covered within the external source. As in long-tail entity extraction we are concerned with explicitly identifying which entity is not yet part of a knowledge base, we would need this specific relationship to be covered in a source. It is unlikely to find such a source that covers this kind of knowledge, as by definition long-tail entities are not generally found in sources, especially not cross-domain sources.

Data programming [Ratner et al., 2016, Ratner et al., 2017, Bach et al., 2019] is a paradigm, where weak supervision is introduced in the form of user-provided labeling functions, which can then be used to label training data for a supervised machine learning algorithm. Existing works on weak supervision do not limit the format of these labeling functions and treat them as black-boxes or individual programs. These labeling functions likely produce noisy and conflicting labels, so that existing methods have primarily focused on denoising the output of such labeling functions using generative models.

Summary and Contributions

We introduce in this work a weak supervision approach, where class-specific supervision can be provided with a small set of class-specific bold matching rules expressed using the schema of the knowledge base. The rules are termed bold, as we require rules to be only somewhat accurate, while there is no requirement for these rules to have a high coverage. In our methodology, the rules are ensembled with a class-agnostic unsupervised matching model. This ensembling yields a class-specific weakly supervised labeling function with a full coverage provided by the unsupervised model. We use this labeling function in line with the data programming paradigm to bootstrap a supervised matching model. This is done by labeling training examples from a set of randomly chosen web tables and using them to train a random forest classifier.

Our approach has two contributions when compared to existing work in the area of weak supervision using heuristics or labeling functions. First, unlike black-box labeling functions, rules within our methodology are easy to create and semantically clear to understand. Individual rules are simple conjuncts of attribute tests based on the schema of the knowledge base. Within these rules, there are no weights, nor are there programs or external resources used. The rules are executed

using features and equivalence functions already present within the LTEE Pipeline. Additionally, the rules can be used as weak supervision for any matching tasks of a given class, i.e. the same rules are used for row clustering and new detection.

Secondly, we require only a small set of rules per class. By ensembling the rules with an unsupervised model, we are not only able to provide full coverage, but though averaging the output of the rules and the unsupervised model, we are able to reduce the negative impact of possibly biased or even inaccurate rules. We have showed in our discussion how using a small number of rules alone, without ensembling with the class-agnostic unsupervised model will not yield an effective labeling function.

Finally, we have provided an extensive discussion on the design and parameter choices of the actual machine learning algorithm we use to bootstrap a supervised classifier using weak supervision. We present arguments, why random forests are especially useful for this task, and discuss the hyperparameters choices most suitable when training from labeled data that is likely noisy.

Open Issues and Future Research

Regarding weak supervision for long-tail entity matching, there are multiple opportunities for future research. First of all and similar to the discussion on long-tail entity extraction above, an evaluation of our weak supervision approach on a larger number of classes could provide more insight into its universality and usefulness.

As the bold rules are based on the schema of the knowledge base, they could potentially be mined from the knowledge base itself. In this case, no human experts are required to provide supervision. Additionally, a large number of more diverse rules could be minded, potentially improving the quality of the labeled data.

Given the importance of the unsupervised model, future research could focus on improving it. We have outlined in Chapter 10 multiple approaches on how this could be done. This includes ensembling multiple versatile unsupervised models, dynamic thresholding, automatic weighting, and using multiple iterations.

Similarly, one could investigate alternative approaches to ensembling. Currently, we ensemble the unsupervised model with only one of the rules from the set, i.e. the one fired with the highest confidence. A more sophisticated approach could potentially lead to better results, by considering all fired rules.

Future research could also investigate the use of alternative supervised machine learning algorithms during bootstrapping. This would allow a confirmation that random forests are especially suitable for training on noisy training data.

Finally, our bootstrapping approach could be extended using an iterative process. In our experimental results in Chapter 10, we already found that by using bootstrapping and a supervised model trained using strong supervision, we could train an even more effective model. This resembles a semi-supervised approach. We could similarly use a model bootstrapped from weak supervision as a labeling function to iteratively bootstrap a potentially more effective model.

11.4 Research Impact

The methods researched in this thesis form the end within a long chain of data integration tasks. Using web tables for knowledge base enrichment begins with crawling websites and extracting web tables and continues with parsing and normalizing these tables to then match them to a knowledge base schema. Finally, we apply identity resolution and fusion methods to perform either slot filling or entity expansion. This thesis focuses on these last two steps in the long list of methods required to enrich a knowledge with long-tail knowledge from web tables.

This chaining of methods necessarily leads to the compounding of errors, increasing the difficulty of tasks at the end of the chain. This also possibly explains why problems like time-aware fusion and long-tail entity extraction are less studied in related work, when compared to web table matching. This thesis closes this gap by extensively researching both tasks and enabling the actual enrichment of cross-domain knowledge bases with long-tail knowledge using web tables.

The significance of web tables for knowledge base enrichment is highlighted by the fact that our work on web table profiling [Ritze et al., 2016] is highly cited. Similarly, two survey papers on web tables have recently been published [Cafarella et al., 2018,Zhang and Balog, 2020], showing the interest in research on web tables.

Existing works on time-aware fusion view the temporal scope as an additional semantic annotation, like unit and scale [Zhang and Chakrabarti, 2013]. However, in the context of fusion, unit and scale fall into the aspect of correctness, whereas the identification of the correct temporal scope falls in the aspect of validity. Our research is the first to explicitly introduce and define the task of targeted slot-filling (see Sections 6.1 and 7.1), while differentiating temporal from snapshot-based knowledge bases and temporal knowledge from listing data. We additionally introduce the Time-Dependent Ground Truth, which could enable a large range of research for tasks that consider the temporal aspect of data.

To the best of our knowledge, we are the first to research the task of long-tail entity extraction. There is currently one paper [Zhang et al., 2020] in preprint on long-tail entity extraction to be presented at the World Wide Web Conference. Unfortunately, the authors seem to be unaware of our research. However, this shows the increasing importance of the topic of long-tail entity extraction. We have compared their work to ours in Section 8.4. With the T4LTE dataset, we have also introduced the first benchmark for long-tail entity extraction from web tables.

Finally, regarding our work on weak supervision for entity matching, there exists a recently published relevant work that also cites our research [Primpeli et al., 2020]. The authors use an unsupervised model from which they bootstrap a supervised model to alleviate the cold-start problem of active learning. The unsupervised model, similar to ours, uses a weighted average of normalized feature scores. However, the authors introduce and employ a dynamic thresholding method. Similar to us, the authors also use random forests when bootstrapping and training supervised models. The suggested approach is able to counteract the cold-start problem, while additionally improving overall F1 and stability.

List of Figures

List of Tables

Bibliography

[Abiteboul, 1997] Abiteboul, S. (1997). Querying semi-structured data. In *Database Theory - ICDT '97*, proceedings of the 6th International Conference on Database Theory, ICDT '97, pages 1–18, Berlin, Germany. Springer-Verlag.

[Ailon et al., 2008] Ailon, N., Charikar, M., and Newman, A. (2008). Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5):23:1–23:27.

[Alexe et al., 2014] Alexe, B., Roth, M., and Tan, W.-C. (2014). Preference-aware integration of temporal data. *Proceedings of the VLDB Endowment*, 8(4):365–376.

[Auer et al., 2007] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). DBpedia: A nucleus for a web of open data. In *The Semantic Web*, proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference, ISWC/ASWC '07, pages 722–735, Berlin, Germany. Springer-Verlag.

[Bach et al., 2019] Bach, S. H., Rodriguez, D., Liu, Y., Luo, C., Shao, H., Xia, C., Sen, S., Ratner, A., Hancock, B., Alborzi, H., Kuchhal, R., Ré, C., and Malkin, R. (2019). Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 362–375, New York, NY, USA. Association for Computing Machinery.

[Balakrishnan et al., 2015] Balakrishnan, S., Halevy, A. Y., Harb, B., Lee, H., Madhavan, J., Rostamizadeh, A., Shen, W., Wilder, K., Wu, F., and Yu, C. (2015). Applying webtables in practice. In *Seventh Biennial Conference on Innovative Data Systems Research*, CIDR '15. Online Proceedings, cidrdb.org.

[Bansal et al., 2004] Bansal, N., Blum, A., and Chawla, S. (2004). Correlation clustering. *Machine Learning*, 56(1):89–113.

[Bao et al., 2014] Bao, J., Duan, N., Zhou, M., and Zhao, T. (2014). Knowledge-based question answering as machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1:*

Long Papers), ACL '14, pages 967–976, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Beckett, 2014] Beckett, D. (2014). RDF 1.1 n-triples. W3C Recommendation 25 February 2014, W3C. https://www.w3.org/TR/2014/REC-n-triples-20140225/.

[Bellini et al., 2017] Bellini, V., Anelli, V. W., Di Noia, T., and Di Sciascio, E. (2017). Auto-encoding user ratings via knowledge graphs in recommendation scenarios. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, DLRS '17, pages 60–66, New York, NY, USA. Association for Computing Machinery.

[Bernstein et al., 2011] Bernstein, P. A., Madhavan, J., and Rahm, E. (2011). Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701.

[Bhagavatula et al., 2015] Bhagavatula, C. S., Noraset, T., and Downey, D. (2015). Tabel: Entity linking in web tables. In *The Semantic Web – ISWC 2015*, proceedings of the 14th International Semantic Web Conference, ISWC '15, pages 425–441, Cham, Switzerland. Springer International Publishing.

[Bilke and Naumann, 2005] Bilke, A. and Naumann, F. (2005). Schema matching using duplicates. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 69–80, Los Alamitos, CA, USA. IEEE Computer Society.

[Bing et al., 2013] Bing, L., Lam, W., and Wong, T.-L. (2013). Wikipedia entity expansion and attribute extraction from the web using semi-supervised learning. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 567–576, New York, NY, USA. Association for Computing Machinery.

[Bizer, 2009] Bizer, C. (2009). The emerging web of linked data. *IEEE Intelligent Systems*, 24(5):87–92.

[Bizer et al., 2009] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165.

[Bizer et al., 2019] Bizer, C., Primpeli, A., and Peeters, R. (2019). Using the semantic web as a source of training data. *Datenbank-Spektrum*, 19(2):127–135.

[Blanco et al., 2013] Blanco, R., Cambazoglu, B. B., Mika, P., and Torzec, N. (2013). Entity recommendations in web search. In *The Semantic Web – ISWC 2013*, proceedings of the 12th International Semantic Web Conference, ISWC '13, pages 33–48, Berlin, Germany. Springer-Verlag.

[Bollacker et al., 2008] 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*, SIGMOD '08, pages 1247–1250, New York, NY, USA. Association for Computing Machinery.

[Bonchi et al., 2014] Bonchi, F., Garcia-Soriano, D., and Liberty, E. (2014). Correlation clustering: From theory to practice. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1972–1972, New York, NY, USA. Association for Computing Machinery.

[Bordes et al., 2014] Bordes, A., Chopra, S., and Weston, J. (2014). Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP '14, pages 615–620, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Bordes et al., 2013] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems 26*, proceedings of the 27th Annual Conference on Neural Information Processing Systems, NIPS '13, pages 2787–2795, Red Hook, NY, USA. Curran Associates, Inc.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

[Brin and Page, 1998] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107 – 117.

[Bryl et al., 2016] Bryl, V., Bizer, C., and Paulheim, H. (2016). Gathering alternative surface forms for dbpedia entities. In *Proceedings of the Third NLP&DBpedia Workshop (NLP & DBpedia 2015)*, NLP & DBpedia '15, pages 13–24, Aachen, Germany. CEUR Workshop Proceedings, RWTH.

[Bühmann et al., 2014] Bühmann, L., Usbeck, R., Ngonga Ngomo, A.-C., Saleem, M., Both, A., Crescenzi, V., Merialdo, P., and Qiu, D. (2014). Web-scale extension of rdf knowledge bases from templated websites. In *The Semantic Web – ISWC 2014*, proceedings of the 13th International Semantic Web Conference, ISWC '14, pages 66–81, Cham, Switzerland. Springer International Publishing.

[Cafarella et al., 2018] Cafarella, M., Halevy, A., Lee, H., Madhavan, J., Yu, C., Wang, D. Z., and Wu, E. (2018). Ten years of webtables. *Proceedings of the VLDB Endowment*, 11(12):2140–2149.

[Cafarella et al., 2009a] Cafarella, M. J., Halevy, A., and Khoussainova, N. (2009a). Data integration for the relational web. *Proceedings of the VLDB Endowment*, 2(1):1090–1101.

[Cafarella et al., 2008a] Cafarella, M. J., Halevy, A., Wang, D. Z., Wu, E., and Zhang, Y. (2008a). Webtables: Exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549.

[Cafarella et al., 2008b] Cafarella, M. J., Halevy, A. Y., Zhang, Y., Wang, D. Z., and Wu, E. (2008b). Uncovering the relational web. In *Proceedings of the 11th International Workshop on Web and Databases*, WebDB '08.

[Cafarella et al., 2009b] Cafarella, M. J., Madhavan, J., and Halevy, A. (2009b). Web-scale extraction of structured data. *SIGMOD Record*, 37(4):55–61.

[Cao et al., 2020] Cao, E., Wang, D., Huang, J., and Hu, W. (2020). Open knowledge enrichment for long-tail entities. In *Proceedings of The Web Conference 2020*, WWW '20, pages 384–394, New York, NY, USA. Association for Computing Machinery.

[Carlson et al., 2010a] Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E. R., and Mitchell, T. M. (2010a). Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI '10, pages 1306–1313, Menlo Park, CA, USA. The AAAI Press.

[Carlson et al., 2010b] Carlson, A., Betteridge, J., Wang, R. C., Hruschka, Jr., E. R., and Mitchell, T. M. (2010b). Coupled semi-supervised learning for information extraction. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 101–110, New York, NY, USA. Association for Computing Machinery.

[Christen, 2008] Christen, P. (2008). Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 151–159, New York, NY, USA. Association for Computing Machinery.

[Christen, 2012] Christen, P. (2012). The data matching process. In *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, Data-Centric Systems and Applications. Springer, Heidelberg, Germany.

[Christophides et al., 2015] Christophides, V., Efthymiou, V., and Stefanidis, K. (2015). *Entity Resolution in the Web of Data*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers.

[Crestan and Pantel, 2010] Crestan, E. and Pantel, P. (2010). Web-scale knowledge extraction from semi-structured tables. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1081–1082, New York, NY, USA. Association for Computing Machinery.

[Crestan and Pantel, 2011] Crestan, E. and Pantel, P. (2011). Web-scale table census and classification. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 545–554, New York, NY, USA. Association for Computing Machinery.

[Daiber et al., 2013] Daiber, J., Jakob, M., Hokamp, C., and Mendes, P. N. (2013). Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems*, I-SEMANTICS '13, pages 121–124, New York, NY, USA. Association for Computing Machinery.

[Demaine et al., 2006] Demaine, E. D., Emanuel, D., Fiat, A., and Immorlica, N. (2006). Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2):172 – 187.

[Derczynski et al., 2017] Derczynski, L., Nichols, E., van Erp, M., and Limsopatham, N. (2017). Results of the WNUT2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, WNUT '17, pages 140–147, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Doan et al., 2012a] Doan, A., Halevy, A., and Ives, Z. (2012a). *Principles of Data Integration*. Morgan Kaufmann, Boston, MA, USA, 1st edition.

[Doan et al., 2012b] Doan, A., Halevy, A., and Ives, Z. (2012b). String matching. In *Principles of Data Integration*, pages 95 – 119. Morgan Kaufmann, Boston, MA, USA, 1st edition.

[Dong et al., 2014a] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014a). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610, New York, NY, USA. Association for Computing Machinery.

[Dong et al., 2009] Dong, X. L., Berti-Equille, L., and Srivastava, D. (2009). Truth discovery and copying detection in a dynamic world. *Proceedings of the VLDB Endowment*, 2(1):562–573.

[Dong et al., 2013] Dong, X. L., Berti-Equille, L., and Srivastava, D. (2013). Data fusion: Resolving conflicts from multiple sources. In *Web-Age Information Management*, proceedings of the 14th International Conference on Web-Age

Information Management, WAIM '13, pages 64–76, Berlin, Germany. Springer-Verlag.

[Dong et al., 2014b] Dong, X. L., Gabrilovich, E., Heitz, G., Horn, W., Murphy, K., Sun, S., and Zhang, W. (2014b). From data fusion to knowledge fusion. *Proceedings of the VLDB Endowment*, 7(10):881–892.

[Dong et al., 2015] Dong, X. L., Gabrilovich, E., Murphy, K., Dang, V., Horn, W., Lugaresi, C., Sun, S., and Zhang, W. (2015). Knowledge-based trust: Estimating the trustworthiness of web sources. *Proceedings of the VLDB Endowment*, 8(9):938–949.

[Dong et al., 2016] Dong, X. L., Kementsietsidis, A., and Tan, W.-C. (2016). A time machine for information: Looking back to look forward. *SIGMOD Record*, 45(2):23–32.

[Dong and Srivastava, 2015a] Dong, X. L. and Srivastava, D. (2015a). *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael, CA, USA.

[Dong and Srivastava, 2015b] Dong, X. L. and Srivastava, D. (2015b). Data fusion. In *Big Data Integration*, Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael, CA, USA.

[Dong and Srivastava, 2015c] Dong, X. L. and Srivastava, D. (2015c). Record linkage. In *Big Data Integration*, Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael, CA, USA.

[Dutta, 2016] Dutta, A. (2016). *Automated Knowledge Base Extension Using Open Information*. PhD thesis, University of Mannheim, Mannheim, Germany.

[Dutta et al., 2015] Dutta, A., Meilicke, C., and Stuckenschmidt, H. (2015). Enriching structured knowledge with open information. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 267–277, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Eberius et al., 2015] Eberius, J., Braunschweig, K., Hentsch, M., Thiele, M., Ahmadov, A., and Lehner, W. (2015). Building the dresden web table corpus: A classification approach. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 41–50, Los Alamitos, CA, USA. IEEE Computer Society.

[Ehrlinger and Wöß, 2016] Ehrlinger, L. and Wöß, W. (2016). Towards a definition of knowledge graphs. In *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics*

(SuCCESS'16), SEMANTiCS '16, Aachen, Germany. CEUR Workshop Proceedings, RWTH.

[Elsner and Charniak, 2008] Elsner, M. and Charniak, E. (2008). You talking to me? a corpus and algorithm for conversation disentanglement. In *Proceedings of ACL-08: HLT*, ACL-HLT '08, pages 834–842, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Elsner and Schudy, 2009] Elsner, M. and Schudy, W. (2009). Bounding and comparing methods for correlation clustering beyond ilp. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*, pages 19–27, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Etzioni et al., 2004] Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D., and Yates, A. (2004). Methods for domain-independent information extraction from the web: An experimental comparison. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, AAAI '04, pages 391–398, Menlo Park, CA, USA. The AAAI Press.

[Fan et al., 2014] Fan, W., Geerts, F., Tang, N., and Yu, W. (2014). Conflict resolution with data currency and consistency. *Journal of Data and Information Quality*, 5(1-2):6:1–6:37.

[Färber et al., 2018] Färber, M., Bartscherer, F., Menne, C., and Rettinger, A. (2018). Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web*, 9(1):77–129.

[Färber et al., 2016] Färber, M., Rettinger, A., and El Asmar, B. (2016). On emerging entity detection. In *Knowledge Engineering and Knowledge Management*, proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management, EKAW '16, pages 223–238, Cham, Switzerland. Springer International Publishing.

[Ferrucci et al., 2010] Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., Schlaefer, N., and Welty, C. (2010). Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79.

[Galárraga et al., 2015] Galárraga, L., Teflioudi, C., Hose, K., and Suchanek, F. M. (2015). Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24(6):707–730.

[Galárraga et al., 2013] Galárraga, L. A., Teflioudi, C., Hose, K., and Suchanek, F. (2013). Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 413–422, New York, NY, USA. Association for Computing Machinery.

[Gatterbauer et al., 2007] Gatterbauer, W., Bohunsky, P., Herzog, M., Krüpl, B., and Pollak, B. (2007). Towards domain-independent information extraction from web tables. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 71–80, New York, NY, USA. Association for Computing Machinery.

[Geurts, 2010] Geurts, P. (2010). Bias vs variance decomposition for regression and classification. In Maimon, O. and Rokach, L., editors, *Data Mining and Knowledge Discovery Handbook*, pages 733–746. Springer, Boston, MA, USA.

[Han et al., 2011] Han, J., Pei, J., and Kamber, M. (2011). *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, Boston, MA, USA, 3rd edition.

[Hassanzadeh et al., 2009] Hassanzadeh, O., Chiang, F., Lee, H. C., and Miller, R. J. (2009). Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment*, 2(1):1282–1293.

[Hassanzadeh et al., 2015] Hassanzadeh, O., Ward, M. J., Rodriguez-Muro, M., and Srinivas, K. (2015). Understanding a large corpus of web tables through matching with knowledge bases: an empirical study. In *Proceedings of the 10th International Workshop on Ontology Matching*, OM '15, pages 25–34, Aachen, Germany. CEUR Workshop Proceedings, RWTH.

[He et al., 2007] He, B., Patel, M., Zhang, Z., and Chang, K. C.-C. (2007). Accessing the deep web. *Communications of the ACM*, 50(5):94–101.

[Ho et al., 2018] Ho, V. T., Stepanova, D., Gad-Elrab, M. H., Kharlamov, E., and Weikum, G. (2018). Rule learning from knowledge graphs guided by embedding models. In *The Semantic Web – ISWC 2018*, proceedings of the 17th International Semantic Web Conference, ISWC '18, pages 72–90, Cham, Switzerland. Springer International Publishing.

[Hoffart et al., 2014] Hoffart, J., Altun, Y., and Weikum, G. (2014). Discovering emerging entities with ambiguous names. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 385–396, New York, NY, USA. Association for Computing Machinery.

[Hoffart et al., 2016] Hoffart, J., Milchevski, D., Weikum, G., Anand, A., and Singh, J. (2016). The knowledge awakens: Keeping knowledge bases fresh with emerging entities. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 203–206, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Hoffart et al., 2013] Hoffart, J., Suchanek, F. M., Berberich, K., and Weikum, G. (2013). Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61.

[Jurek et al., 2017] Jurek, A., Hong, J., Chi, Y., and Liu, W. (2017). A novel ensemble learning approach to unsupervised record linkage. *Information Systems*, 71:40–54.

[Kang, 2013] Kang, H. (2013). The prevention and handling of the missing data. *Korean journal of anesthesiology*, 64(5):402–406.

[Kejriwal and Miranker, 2015] Kejriwal, M. and Miranker, D. P. (2015). An unsupervised instance matcher for schema-free rdf data. *Journal of Web Semantics*, 35:102–123.

[Kernighan and Lin, 1970] Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307.

[Keuper et al., 2015] Keuper, M., Levinkov, E., Bonneel, N., Lavoue, G., Brox, T., and Andres, B. (2015). Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proceedings of the 2015 IEEE International Conference on Computer Vision*, ICCV '15, pages 1751–1759, Los Alamitos, CA, USA. IEEE Computer Society.

[Kopliku et al., 2011] Kopliku, A., Pinel-Sauvagnat, K., and Boughanem, M. (2011). Retrieving attributes using web tables. In *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*, JCDL '11, pages 397–398, New York, NY, USA. Association for Computing Machinery.

[Kuzey and Weikum, 2012] Kuzey, E. and Weikum, G. (2012). Extraction of temporal facts and events from wikipedia. In *Proceedings of the 2nd Temporal Web Analytics Workshop*, TempWeb '12, page 25–32, New York, NY, USA. Association for Computing Machinery.

[Larose, 2014] Larose, D. T. (2014). *Discovering knowledge in data : an introduction to data mining*. Wiley Series on Methods and Applications in Data Mining. JohnWiley & Sons, Inc, Hoboken, NJ, USA, 2nd edition.

[Lehmann et al., 2015] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al. (2015). DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195.

[Lehmberg, 2019] Lehmberg, O. (2019). *Web table integration and profiling for knowledge base augmentation*. PhD thesis, University of Mannheim, Mannheim, Germany.

[Lehmberg and Bizer, 2016] Lehmberg, O. and Bizer, C. (2016). Web table column categorisation and profiling. In *Proceedings of the 19th International Workshop on Web and Databases*, WebDB '16, pages 4:1–4:7, New York, NY, USA. Association for Computing Machinery.

[Lehmberg and Bizer, 2017] Lehmberg, O. and Bizer, C. (2017). Stitching web tables for improving matching quality. *Proceedings of the VLDB Endowment*, 10(11):1502–1513.

[Lehmberg and Bizer, 2019a] Lehmberg, O. and Bizer, C. (2019a). Profiling the semantics of n-ary web table data. In *Proceedings of the International Workshop on Semantic Big Data*, SBD '19, pages 5:1–5:6, New York, NY, USA. Association for Computing Machinery.

[Lehmberg and Bizer, 2019b] Lehmberg, O. and Bizer, C. (2019b). Synthesizing n-ary relations from web tables. In *Proceedings of the 9th International Conference on Web Intelligence, Mining and Semantics*, WIMS '19, pages 17:1–17:12, New York, NY, USA. Association for Computing Machinery.

[Lehmberg and Hassanzadeh, 2018] Lehmberg, O. and Hassanzadeh, O. (2018). Ontology augmentation through matching with web tables. In *Proceedings of the 13th International Workshop on Ontology Matching*, OM '18, pages 37–48, Aachen, Germany. CEUR Workshop Proceedings, RWTH.

[Lehmberg et al., 2016] Lehmberg, O., Ritze, D., Meusel, R., and Bizer, C. (2016). A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 75–76, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Lehmberg et al., 2015] Lehmberg, O., Ritze, D., Ristoski, P., Meusel, R., Paulheim, H., and Bizer, C. (2015). The mannheim search join engine. *Web Semantics: Science, Services and Agents on the World Wide Web*, 35(P3):159–166.

[Lenat, 1995] Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38.

[Li et al., 2016] Li, Y., Gao, J., Meng, C., Li, Q., Su, L., Zhao, B., Fan, W., and Han, J. (2016). A survey on truth discovery. *SIGKDD Explorations Newsletter*, 17(2):1–16.

[Limaye et al., 2010] Limaye, G., Sarawagi, S., and Chakrabarti, S. (2010). Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2):1338–1347.

[Ling and Weld, 2010] Ling, X. and Weld, D. S. (2010). Temporal information extraction. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI '10, pages 1385–1390, Menlo Park, CA, USA. The AAAI Press.

[Little, 2020] Little, R. J. A. (2020). *Statistical analysis with missing data*. Wiley series in probability and statistics. John Wiley & Sons, Inc, Hoboken, NJ, USA, 3rd edition.

[Lockard et al., 2018] Lockard, C., Dong, X. L., Einolghozati, A., and Shiralkar, P. (2018). Ceres: Distantly supervised relation extraction from the semi-structured web. *Proceedings of the VLDB Endowment*, 11(10):1084–1096.

[Lockard et al., 2019] Lockard, C., Shiralkar, P., and Dong, X. L. (2019). OpenCeres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, NAACL-HLT '19, pages 3047–3056, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Madhavan et al., 2001] Madhavan, J., Bernstein, P. A., and Rahm, E. (2001). Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 49–58, San Francisco, CA, USA. Morgan Kaufmann Publishers.

[Mahdisoltani et al., 2015] Mahdisoltani, F., Biega, J., and Suchanek, F. M. (2015). YAGO3: A knowledge base from multilingual wikipedias. In *Seventh Biennial Conference on Innovative Data Systems Research*, CIDR '15. Online Proceedings, cidrdb.org.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press, New York, NY, USA.

[Meilicke et al., 2019a] Meilicke, C., Chekol, M. W., Ruffinelli, D., and Stuckenschmidt, H. (2019a). Anytime bottom-up rule learning for knowledge graph completion. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, IJCAI '19, pages 3137–3143. Online Proceedings, International Joint Conferences on Artificial Intelligence (www.ijcai.org).

[Meilicke et al., 2019b] Meilicke, C., Chekol, M. W., Ruffinelli, D., and Stuckenschmidt, H. (2019b). An introduction to anyburl. In *KI 2019: Advances in Artificial Intelligence*, proceedings of the 42nd German Conference on Artificial Intelligence, KI '19, pages 244–248, Cham, Switzerland. Springer International Publishing.

[Mendes et al., 2012] Mendes, P., Jakob, M., and Bizer, C. (2012). DBpedia: A multilingual cross-domain knowledge base. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation*, LREC '12, pages 1813–1817. Online Proceedings, European Language Resources Association (ELRA).

[Meusel, 2017] Meusel, R. (2017). *Web-scale profiling of semantic annotations in HTML pages*. PhD thesis, University of Mannheim, Mannheim, Germany.

[Meusel et al., 2014] Meusel, R., Petrovski, P., and Bizer, C. (2014). The webdatacommons microdata, rdfa and microformat dataset series. In *The Semantic Web – ISWC 2014*, proceedings of the 13th International Semantic Web Conference, ISWC '14, pages 277–292, Cham, Switzerland. Springer International Publishing.

[Meusel et al., 2015] Meusel, R., Vigna, S., Lehmberg, O., and Bizer, C. (2015). The graph structure in the web – analyzed on different aggregation levels. *The Journal of Web Science*, 1(1):33–47.

[Mihalcea, 2004] Mihalcea, R. (2004). Co-training and self-training for word sense disambiguation. In *Proceedings of the Eighth Conference on Computational Natural Language Learning*, CoNLL '04, pages 33–40, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Mika et al., 2014] Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., and Goble, C., editors (2014). *The Semantic Web – ISWC 2014*, Cham, Switzerland. Springer International Publishing.

[Mintz et al., 2009] Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, ACL '09, pages 1003–1011, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Mitchell et al., 2018] Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., and Welling, J. (2018). Never-ending learning. *Communications of the ACM*, 61(5):103–115.

[Mulwad et al., 2010] Mulwad, V., Finin, T., Syed, Z., and Joshi, A. (2010). Using linked data to interpret tables. In *Proceedings of the the First International Workshop on Consuming Linked Data*, COLD '10, Aachen, Germany. CEUR Workshop Proceedings, RWTH.

[Niklaus et al., 2018] Niklaus, C., Cetto, M., Freitas, A., and Handschuh, S. (2018). A survey on open information extraction. In *Proceedings of the 27th International Conference on Computational Linguistics*, COLING '18, pages 3866–3878, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Otsu, 1979] Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66.

[Oulabi and Bizer, 2017] Oulabi, Y. and Bizer, C. (2017). Estimating missing temporal meta-information using knowledge-based-trust. In *Proceedings of the 3rd International Workshop on Knowledge Discovery on the WEB*, KDWEB '17, Aachen, Germany. CEUR Workshop Proceedings, RWTH.

[Oulabi and Bizer, 2019a] Oulabi, Y. and Bizer, C. (2019a). Extending cross-domain knowledge bases with long tail entities using web table data. In *Proceedings of the 22nd International Conference on Extending Database Technology*, EDBT '19, pages 385–396, Konstanz, Germany. OpenProceedings.org.

[Oulabi and Bizer, 2019b] Oulabi, Y. and Bizer, C. (2019b). Using weak supervision to identify long-tail entities for knowledge base completion. In *Semantic Systems. The Power of AI and Knowledge Graphs*, proceedings of the 15th International Conference on Semantic Systems, SEMANTiCS '19, pages 83–98, Cham, Switzerland. Springer International Publishing.

[Oulabi et al., 2016] Oulabi, Y., Meusel, R., and Bizer, C. (2016). Fusing time-dependent web table data. In *Proceedings of the 19th International Workshop on Web and Databases*, WebDB '16, pages 3:1–3:7, New York, NY, USA. Association for Computing Machinery.

[Pal et al., 2012] Pal, A., Rastogi, V., Machanavajjhala, A., and Bohannon, P. (2012). Information integration over time in unreliable and uncertain environments. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 789–798, New York, NY, USA. Association for Computing Machinery.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[Pantel et al., 2009] Pantel, P., Crestan, E., Borkovsky, A., Popescu, A.-M., and Vyas, V. (2009). Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 938–947, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Pasternack and Roth, 2010] Pasternack, J. and Roth, D. (2010). Knowing what to believe (when you already know something). In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 877–885, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Paulheim, 2017] Paulheim, H. (2017). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8(3):489–508.

[Petrovski et al., 2017] Petrovski, P., Primpeli, A., Meusel, R., and Bizer, C. (2017). The wdc gold standards for product feature extraction and product matching. In *E-Commerce and Web Technologies*, proceedings of the 17th International Conference on E-Commerce and Web Technologies, EC-Web '16, pages 73–86, Cham, Switzerland. Springer International Publishing.

[Popescul et al., 2003] Popescul, A., Popescul, R., and Ungar, L. H. (2003). Statistical relational learning for link prediction. In *Proceedings of the Workshop on Learning Statistical Models from Relational Data at IJCAI-2003*, number 162 in Computer Science Department Faculty Publication Series, Amherst, MA, USA. ScholarWorks@UMass Amherst.

[Primpeli and Bizer, 2019] Primpeli, A. and Bizer, C. (2019). Robust active learning of expressive linkage rules. In *Proceedings of the 9th International Conference on Web Intelligence, Mining and Semantics*, WIMS '19, pages 2:1–2:7, New York, NY, USA. Association for Computing Machinery.

[Primpeli et al., 2020] Primpeli, A., Bizer, C., and Keuper, M. (2020). Unsupervised bootstrapping of active learning for entity resolution. In *The Semantic Web*, proceedings of the 17th Extended Semantic Web Conference, ESWC '20, pages 215–231, Cham, Switzerland. Springer Nature Switzerland.

[Pujara et al., 2013] Pujara, J., Miao, H., Getoor, L., and Cohen, W. (2013). Knowledge graph identification. In *The Semantic Web – ISWC 2013*, proceedings of the 12th International Semantic Web Conference, ISWC '13, pages 542–557, Berlin, Germany. Springer-Verlag.

[Rahm and Bernstein, 2001] Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350.

[Ratner et al., 2017] Ratner, A., Bach, S. H., Ehrenberg, H., Fries, J., Wu, S., and Ré, C. (2017). Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282.

[Ratner et al., 2016] Ratner, A. J., Sa, C. D., Wu, S., Selsam, D., and Ré, C. (2016). Data programming: Creating large training sets, quickly. In *Advances in Neural Information Processing Systems 29*, proceedings of the 30th Annual Conference on Neural Information Processing Systems, NIPS '16, pages 3567–3575, Red Hook, NY, USA. Curran Associates Inc.

[Rekatsinas et al., 2014] Rekatsinas, T., Dong, X. L., and Srivastava, D. (2014). Characterizing and selecting fresh data sources. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 919–930, New York, NY, USA. Association for Computing Machinery.

[Ringler and Paulheim, 2017] Ringler, D. and Paulheim, H. (2017). One knowledge graph to rule them all? analyzing the differences between dbpedia, yago, wikidata & co. In *KI 2017: Advances in Artificial Intelligence*, proceedings of the 40th Annual German Conference on Artificial Intelligence, KI '17, pages 366–372, Cham, Switzerland. Springer International Publishing.

[Ristoski et al., 2015] Ristoski, P., Bizer, C., and Paulheim, H. (2015). Mining the web of linked data with rapidminer. *Journal of Web Semantics*, 35(3):142–151.

[Ritze, 2017] Ritze, D. (2017). *Web-scale web table to knowledge base matching*. PhD thesis, University of Mannheim, Mannheim, Germany.

[Ritze and Bizer, 2017] Ritze, D. and Bizer, C. (2017). Matching web tables to dbpedia - A feature utility study. In *Proceedings of the 20th International Conference on Extending Database Technology*, EDBT '17, pages 210–221, Konstanz, Germany. OpenProceedings.org.

[Ritze et al., 2015] Ritze, D., Lehmberg, O., and Bizer, C. (2015). Matching html tables to dbpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, WIMS '15, pages 10:1–10:6, New York, NY, USA. Association for Computing Machinery.

[Ritze et al., 2016] Ritze, D., Lehmberg, O., Oulabi, Y., and Bizer, C. (2016). Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 251–261, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Sekhavat et al., 2014] Sekhavat, Y. A., Paolo, F. D., Barbosa, D., and Merialdo, P. (2014). Knowledge base augmentation using tabular data. In *Proceedings of the Workshop on Linked Data on the Web*, LDOW '14, Aachen, Germany. CEUR Workshop Proceedings, RWTH.

[Settles, 2012] Settles, B. (2012). Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114.

[Shen et al., 2005] Shen, W., Li, X., and Doan, A. (2005). Constraint-based entity matching. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, AAAI '05, pages 862–867, Menlo Park, CA, USA. The AAAI Press.

[Singh et al., 2016] Singh, J., Hoffart, J., and Anand, A. (2016). Discovering entities with just a little help from you. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1331–1340, New York, NY, USA. Association for Computing Machinery.

[Socher et al., 2013] Socher, R., Chen, D., Manning, C. D., and Ng, A. Y. (2013). Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26*, proceedings of the 27th Annual Conference on Neural Information Processing Systems, NIPS '13, pages 926–934, Red Hook, NY, USA. Curran Associates Inc.

[Strötgen and Gertz, 2010] Strötgen, J. and Gertz, M. (2010). Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 321–324, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Strötgen and Gertz, 2015] Strötgen, J. and Gertz, M. (2015). A baseline temporal tagger for all languages. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, EMNLP '15, pages 541–547, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Su et al., 2010] Su, W., Wang, J., and Lochovsky, F. H. (2010). Record matching over query results from multiple web databases. *IEEE Transactions on Knowledge and Data Engineering*, 22(4):578–589.

[Suchanek et al., 2007] Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: A core of semantic knowledge unifying wordnet and wikipedia. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA. Association for Computing Machinery.

[Suchanek et al., 2008] Suchanek, F. M., Kasneci, G., and Weikum, G. (2008). Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217.

[Sun et al., 2016] Sun, H., Ma, H., He, X., Yih, W.-t., Su, Y., and Yan, X. (2016). Table cell search for question answering. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 771–782, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Surdeanu, 2013] Surdeanu, M. (2013). Overview of the tac2013 knowledge base population evaluation: English slot filling and temporal slot filling. In *Proceedings of the Sixth Text Analysis Conference*, TAC '13, Gaithersburg, MD, USA. National Institute of Standards and Technology.

[Tanon et al., 2016] Tanon, T. P., Vrandečić, D., Schaffert, S., Steiner, T., and Pintscher, L. (2016). From freebase to wikidata: The great migration. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 1419–1428, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Varma and Ré, 2018] Varma, P. and Ré, C. (2018). Snuba: Automating weak supervision to label training data. *Proceedings of the VLDB Endowment*, 12(3):223–236.

[Verhagen et al., 2010] Verhagen, M., Saurí, R., Caselli, T., and Pustejovsky, J. (2010). Semeval-2010 task 13: Tempeval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 57–62, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Vrandečić and Krötzsch, 2014] Vrandečić, D. and Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

[Wang et al., 2015] Wang, C., Chakrabarti, K., He, Y., Ganjam, K., Chen, Z., and Bernstein, P. A. (2015). Concept expansion using web tables. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1198–1208, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Wang et al., 2012] Wang, J., Wang, H., Wang, Z., and Zhu, K. Q. (2012). Understanding tables on the web. In *Conceptual Modeling*, proceedings of the 31st International Conference on Conceptual Modeling, ER '12, pages 141–155, Berlin, Germany. Springer-Verlag.

[Wang et al., 2017] Wang, Q., Mao, Z., Wang, B., and Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

[Wang and Cohen, 2007] Wang, R. C. and Cohen, W. W. (2007). Language-independent set expansion of named entities using the web. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ICDM '07, pages 342–350, Los Alamitos, CA, USA. IEEE Computer Society.

[Wang and Hu, 2002] Wang, Y. and Hu, J. (2002). A machine learning based approach for table detection on the web. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 242–250, New York, NY, USA. Association for Computing Machinery.

[Wang and Huang, 2019] Wang, Z. and Huang, Y. (2019). Knowledge base completion by inference from both relational and literal facts. In *Advances in Knowledge Discovery and Data Mining*, proceedings of the 23rd Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD '19, pages 501–513, Cham, Switzerland. Springer International Publishing.

[Wang et al., 2014] Wang, Z., Yan, S., Wang, H., and Huang, X. (2014). An overview of microsoft deep qa system on stanford webquestions benchmark. Technical Report MSR-TR-2014-121, Microsoft Research.

[Wu et al., 2016] Wu, Z., Song, Y., and Giles, C. (2016). Exploring multiple feature spaces for novel entity discovery. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI '16, pages 3073–3079, Palo Alto, CA, USA. The AAAI Press.

[Yakout et al., 2012] Yakout, M., Ganjam, K., Chakrabarti, K., and Chaudhuri, S. (2012). Infogather: Entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 97–108, New York, NY, USA. Association for Computing Machinery.

[Yao and Van Durme, 2014] Yao, X. and Van Durme, B. (2014). Information extraction over structured data: Question answering with Freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL '14, pages 956–966, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Yeo et al., 2016] Yeo, J., woo Park, J., and won Hwang, S. (2016). Understanding emerging spatial entities. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI '16, pages 301–307, Palo Alto, CA, USA. The AAAI Press.

[Yin et al., 2008] Yin, X., Han, J., and Yu, P. S. (2008). Truth discovery with multiple conflicting information providers on the web. *IEEE Transactions on Knowledge and Data Engineering*, 20(6):796–808.

[Yin and Tan, 2011] Yin, X. and Tan, W. (2011). Semi-supervised truth discovery. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 217–226, New York, NY, USA. Association for Computing Machinery.

[Yin et al., 2011] Yin, X., Tan, W., and Liu, C. (2011). FACTO: a fact lookup engine based on web tables. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 507–516, New York, NY, USA. Association for Computing Machinery.

[Yu et al., 2019] Yu, R., Gadiraju, U., Fetahu, B., Lehmberg, O., Ritze, D., and Dietze, S. (2019). Knowmore - knowledge base augmentation with structured web markup. *Semantic Web*, 10(1):159–180.

[Zhang et al., 2016] Zhang, F., Yuan, N. J., Lian, D., Xie, X., and Ma, W.-Y. (2016). Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 353–362, New York, NY, USA. Association for Computing Machinery.

[Zhang et al., 2019] Zhang, L., Wu, T., Xu, L., Wang, M., Qi, G., and Sack, H. (2019). Emerging entity discovery using web sources. In *Knowledge Graph and Semantic Computing: Knowledge Computing and Language Understanding*, proceedings of the 4th China Conference on Knowledge Graph and Semantic Computing, CCKS '19, pages 175–184, Singapore. Springer Singapore.

[Zhang and Chakrabarti, 2013] Zhang, M. and Chakrabarti, K. (2013). Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 145–156, New York, NY, USA. Association for Computing Machinery.

[Zhang and Balog, 2017] Zhang, S. and Balog, K. (2017). Entitables: Smart assistance for entity-focused tables. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 255–264, New York, NY, USA. Association for Computing Machinery.

[Zhang and Balog, 2020] Zhang, S. and Balog, K. (2020). Web table extraction, retrieval, and augmentation: A survey. *ACM Transactions on Intelligent Systems and Technology*, 11(2).

[Zhang et al., 2020] Zhang, S., Meij, E., Balog, K., and Reinanda, R. (2020). Novel entity discovery from web tables. In *Proceedings of The Web Conference 2020*, WWW '20, pages 1298–1308, New York, NY, USA. Association for Computing Machinery.

[Zhang et al., 2013] Zhang, X., Chen, Y., Chen, J., Du, X., and Zou, L. (2013). Mapping entity-attribute web tables to web-scale knowledge bases. In *Database Systems for Advanced Applications*, proceedings of the 18th International Conference on Database Systems for Advanced Applications, DASFAA '13, pages 108–122, Berlin, Germany. Springer-Verlag.

[Zhang, 2017] Zhang, Z. (2017). Effective and efficient semantic table interpretation using tableminer(+). *Semantic Web*, 8(6):921–957.