TABEL – A Domain Independent and Extensible Framework for Inferring the Semantics of Tables

by

Varish Vyankatesh Mulwad

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

ABSTRACT

Title of Thesis: *TABEL* – A Domain Independent and Extensible Framework for Inferring the Semantics of Tables

Varish Vyankatesh Mulwad, PhD, May, 2015

Thesis directed by: Dr. Tim Finin, Professor
Department of Computer Science and
Electrical Engineering

Tables are an integral part of documents, reports and Web pages, compactly encoding important information that can be difficult to express in text. Table like structures outside documents, such as spreadsheets, CSV files, log files and databases, are widely used to represent and share information. Many scientific and technical domains use tables to compactly depict information which is difficult to express in text. However, tables remain beyond the scope of regular text processing systems which rely on sentence and grammatical structure as well as the context from surrounding words to understand the meaning of text. They also ignore the structure of table, which humans use to both encode and understand the meaning of information inside a table.

This dissertation presents ***TABEL*** – a domain independent & extensible framework to infer the semantics of tables and represent them as RDF Linked Data. *TABEL* captures the intended meaning by mapping header cells to classes, data cell values to existing entities and pair of columns to relations from an given ontology and knowledge base. The core of the framework consists of a module that represents a table as a graphical model to jointly infer the semantics of headers, data cells and relation between headers. We also introduce a novel Semantic Message Passing scheme, which incorporates semantics into message passing, to perform joint inference over the probabilistic graphical model. We present techniques that are both extensible and domain agnostic. Our framework allows the addition of “preprocessing” modules without affecting existing ones, making *TABEL* extensible. It allows the inferred semantics to be represented as RDF triples using the framework’s ontology or a user’s custom ontology. *TABEL* also allows the background knowledge bases to be adapted and changed based on the domains of the table, thus making it domain independent. We also introduce & explore a “human-in-the-loop” paradigm, presenting different models of user interaction with *TABEL* and its impact on the quality of inferred semantics.

We demonstrate the extensibility and domain independence of our techniques by developing an application of *TABEL* in the healthcare domain. Evidence–based Medicine analyzes questions such as efficacy of drug dosages, correlates various medical factors or tries to find a correlation between drugs by performing meta–analyses (i.e., systematic reviews) over evidence and data previously published in scientific literature and clinical trial studies. We develop a proof of concept system that can automatically produce meta–analyses reports to replace the existing manual and tedious process. Tables from medical research reports (*medical tables*) not only pose domain challenges, but also structural ones as they are multi–dimensional in nature. *TABEL* is both extended and adapted by adding a new domain specific preprocessing module and domain specific knowledge bases. We use *TABEL* to infer the semantics of medical tables and build a proof of concept user interactive system which utilizes the inferred semantics to help researchers discover, extract and integrate data from relevant studies to produce meta–analysis reports.

A thorough evaluation with experiments over dataset of tables from the web and medical research reports present promising results. Our experiments also show that limited user feedback can have significant impact on the quality of the semantics inferred by our framework.

Dedicated to my parents

ACKNOWLEDGMENTS

First and foremost, I would like to thank my adviser Dr. Tim Finin who is an excellent research adviser, mentor and a wonderful human being. I would like to thank him for accepting me as his Master's student in the Spring of 2009 as one amongst the many bright students he accepted that semester. As my interest in pursuing a research career piqued, he encouraged and supported me to join the PhD program. It is one of the best decisions I made and has been a truly rewarding and enriching experience. Throughout my Masters and PhD, Dr. Finin provided me the complete freedom to work through my problems, letting me grow as a researcher. He always stepped in when I faced hurdles and his ability to quickly find solutions to complex problems that I stared at for days was amazing. The depth and long term vision in his approach and solutions has always been mind blowing. His hunger to learn new things is matched by none. I thank him for being a great research & life mentor and a constant source of inspiration.

I would also like to thank Dr. Anupam Joshi with whom I have closely collaborated throughout my PhD. His technical depth as well as breadth of knowledge across several areas has been extremely helpful. He encouraged me to work on challenging use–cases and always kept me focused whenever I lost track. I would especially like to thank him for helping me prepare and search for a career after my PhD. I would like to thank him for his guidance and his support which has been invaluable. I would like to thank both Drs. Finin and Joshi for supporting me to travel and present my research at several top conferences around the world. These travels have been an enriching experience in my life.

I would like to thank the rest of my committee members – Drs. Tim Oates, Yun Peng, L. V. Subramaniam and Indrajit Bhattacharya for providing valuable feedback and comments. Dr. Oates's Machine Learning class at UMBC is one of the best classes I ever took. I will like to thank him for being an amazing teacher and providing me the much needed final push to wrap up my work and defend. I would like to thank Dr. Subramaniam for hosting me at IBM Research for a day, allowing me to present and giving early feedback on my work. I will also like to thank Sandeep Suresh and Todd Segal from Microsoft Bing and Dr. Evelyne Viegas from Microsoft Research for providing wonderful internship opportunities and valuable industry exposure.

My colleagues in the Ebiquity Research Lab provided a stimulating environment to perform research. I would like to thank my amazing seniors – Palani, Zareen, Wenjia, Justin, Lushan, Vlad, Kishor and Audumbar for their mentorship in my early days in graduate school. Palani and Kishor, especially helped me make the right choices while selecting courses and doing research in my first year. Zareen provided me with valuable guidance during my Master's research which laid the foundations for my PhD research. I would like to thank all of them for being available for many days after their respective graduation. I would like thank other members of the Ebiquity Lab, the CSEE department and UMBC– Tejas, Krishna, Anand, Will, Ashwini, Amit, Pramod, Mohit, Akshaya, Pradeep, Amey, Anurag, Nikhil, Sumit, Prajit, Dibyajyoti, Aru, Swarna, Anjana, Tejashree, Sushmita, Jennifer, Clare, Abhay, Lisa, Arnav, Ravendar, Sandhya, Deepal, Puneet, Satyajit, Sunil, Primal, Roberto, Piyush and Sudip for the innumerable discussions over coffee, lunches and dinners.

Graduate school would not have been possible with out the support of amazing friends especially Mayank, Niyati, Vivek, Bhushan, Tejas, Tushar, Sanatan, Chaitra, Sayantan, Nikhil, Ashwin, Prajit, Sunil, Primal and Roberto. A lot of credit goes to Mayank, my first roommate who helped me adjust to a new environment and for putting up with all my idiosyncrasies! Thank you for being a constant support and a great friend all through these years. Niyati Chhaya has always been like an elder sister to me. I have discussed with her innumerable questions, problems and she always had answers. The dedication and efforts Vivek and Bhushan put always motivated me to work harder. I would also like to thank them for making me feel at home during my internships in the Seattle area. Tejas, Tushar, Sanatan and Chaitra have been an inspiration in showing me how to be great at studies and yet be very good at other things like music, photography, sports and dance. I thank Tushar for keeping our entire group together and planning our innumerable road trips. I have explored the United States of America all because of his efforts. Sayantan has always been a great listener and helped me solve all my problems, be it research or in life. If I cook better and dress smartly, it's because of him. When things got a little slow and difficult towards the end, Nikhil and Ashwin helped me maintain my sanity. My unending discussions with Nikhil & Ashwin on a variety of topics have been both enriching and stress busters. I thank them for being amazing roommates for the past year and a half. I thank Ashwin for proof reading my entire thesis document. Prajit, Sunil, Primal and Roberto have shown me tremendous support in the final year of my PhD. My friends in graduate school not only made me a better researcher, but also a better person.

I would also like to thank the past and present staff of the CSEE department at UMBC – Jane, Dee Ann, Vera, Donna, Keara, Camilla, Olivia, Geoff and Marie Tyler for all their help and support. They have been extremely kind and patient with me. Foundations for my interests in research and problem solving can be traced back to my undergraduate days, aided by my wonderful friends Mandeep, Purushottam and Vijay and our undergraduate project adviser Anil Joshi. I would like to thank all of them for continuing to encourage me throughout my Masters and PhD. I would also like to thank Drs. Shyam and Shoba Shirali for being like family away from home.

Amol Dhamdhere has been a mentor since my undergraduate days. This PhD simply would not have been possible without Amol *kaka*'s support. Outside my family, he has been my strongest supporter in some of my toughest days. Last, but by no means the least, I want to thank my parents and family. I thank *Aai*, *Baba* for letting their only son to travel abroad to pursue both a Masters and a PhD. Without their strong support, advice, encouragement, this would not have been possible. I thank them for all the sacrifices they made and all that they have done for me till date.

I would like to end this long acknowledgment on a verse from the Bhagvad Gita, which remains one of my guiding principles: "*karmanye vadhikarasthe ma bhaleshu kathachana ma karmabhalahethurbhurma the sanghohasthwakarmani*".

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

The Web has become a primary source of knowledge and information, largely replacing encyclopedias and reference books. Most Web text is written in a narrative form such as news stories, blogs, reports, etc., but significant amounts of information is also encoded in structured forms such as tables embedded in Web pages and documents. It is estimated that the Web contains over 150 million high quality relational html tables (Cafarella *et al.* 2008). Table like structures outside documents, such as spreadsheets, CSV files, databases and log files, are widely used to represent and share information. Tables are used to present and summarize key data and results in documents in many subject areas, including science, medicine, healthcare, finance, and public policy. Governments around the world, as a part of a coordinated open data and transparency initiative, are publishing government data in CSV format on sites such as data.gov.

However, tables remain beyond the scope of regular text processing systems. Notable progress has been made in developing techniques for analyzing natural language text. These techniques, however, do not work well for tables. To understand the meaning of a word or set of words, they often rely on grammatical knowledge and the context provided by surrounding text which is absent in the case of tables. The very structure of tables which adds value and makes it easier for human understanding also makes it harder for machine understanding. Integrating and searching over information encoded in tables benefits from a better understanding of its intended meaning. Early work focused on extracting tables from documents and web pages (Hurst 2006; Embley, Lopresti, & Nagy 2006) with more recent research attempting to interpret their semantics. Existing work in table interpretation either partially infer the semantics (Venetis *et al.* 2011; Wang *et al.* 2012) based on what application is built on top or only focus on a particular domain such as the Web (Limaye, Sarawagi, & Chakrabarti 2010). In this thesis, we present *TABEL* [1] – a domain independent and extensible framework for inferring the semantics of tables and representing them as RDF Linked Data. Our framework, grounded in a probabilistic graphical model, jointly infers the semantics by mapping every header cell to a class from an ontology, data cell values to existing entities and every pair of column to a relation from an ontology. The inferred semantics is represented as RDF triples allowing applications to utilize the recovered knowledge.

1.1 Motivation

A number of problems and applications can benefit from the inferred semantics of tables or table like structures. Search engines do an excellent job of searching over documents and web pages, but poorly when searching over information encoded inside tables. Using the inferred semantics, search engines will be able return tables or web pages that contain tables for queries such as *US president birthdays*, *US cities and mayors*, *coffee production in Africa*.

Table like structures also appear in the network security domain in the form of log files. Security software and devices such as Intrusion Detection and Prevention systems, webservers, firewalls, and routers generate log files recording not only debugging and diag-

[1]TABEL is an acronym for Tables Extracted as Linked Data.

onistic information, but also events and information useful for audit trails and forensics in the event of malicious activities or system attacks. Analyzing log files can provide valuable insights into system (mis)configuration as well as existing vulnerabilities. This information can be used proactively to protect the host network against attacks. Where a particular log file format is well known, existing systems such as Splunk[2] suffice. The challenge arises when we see a log file which is not from a known service or device, in other words, when we do not know what the rows and columns mean, or what relationships exist between them which is common in large heterogeneous networks. An automated system analyzing and reasoning over log files can benefit from its semantic RDF Linked Data interpretation.

In the medical domain, evidence–based medical research can benefit from exploiting information encoded in tables. Evidence–based medicine analyzes the efficacy of drug dosages, correlates various medical factors, or tries to find a correlation between drugs by performing meta-analyses (i.e., systematic reviews) over evidence and data previously published in scientific literature and clinical trial studies. Key information required not only to discover relevant studies, but also to produce an evidence report (or a meta–analysis report) is encoded in tables like the one in Figure 1.1. Tools for generating meta–analyses reports automatically will benefit from the inferred semantics of tables, both during the discovery of relevant studies and integration of data from multiple studies.

Finally, transforming legacy data stored in tables or table like structures such as CSV or spreadsheets into RDF has been a long standing problem in the Semantic Web community. Existing techniques are either manual or semi–automatic often requiring users to provide terms (classes, relations) to be used in the mapping process. Users who are not well-versed with the Semantic Web find this process challenging; however with the growing number of ontologies and datasets, this becomes a harder task, even for expert users.

[2]www.splunk.com

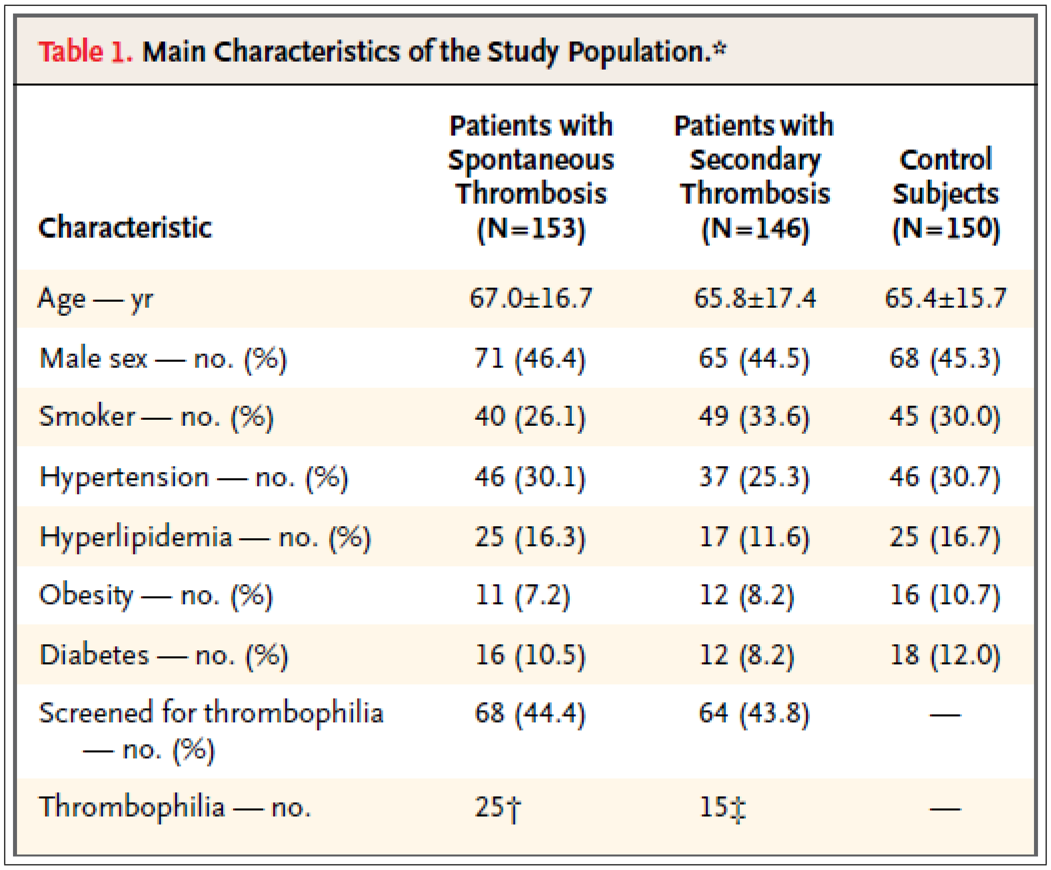


Table 1. Main Characteristics of the Study Population.*

Characteristic	Patients with Spontaneous Thrombosis (N=153)	Patients with Secondary Thrombosis (N=146)	Control Subjects (N=150)
Age — yr	67.0±16.7	65.8±17.4	65.4±15.7
Male sex — no. (%)	71 (46.4)	65 (44.5)	68 (45.3)
Smoker — no. (%)	40 (26.1)	49 (33.6)	45 (30.0)
Hypertension — no. (%)	46 (30.1)	37 (25.3)	46 (30.7)
Hyperlipidemia — no. (%)	25 (16.3)	17 (11.6)	25 (16.7)
Obesity — no. (%)	11 (7.2)	12 (8.2)	16 (10.7)
Diabetes — no. (%)	16 (10.5)	12 (8.2)	18 (12.0)
Screened for thrombophilia — no. (%)	68 (44.4)	64 (43.8)	—
Thrombophilia — no.	25†	15‡	—

FIG. 1.1. Typical tables found in medical research reports include both row and column headers. Information in both header and data cells often encode additional metadata such as units or data cell value type.

Our proposed solution can assist users in automatically transforming tabular data into RDF.

1.2 Inferring the Semantics of Tables

Analyzing tables provides unique challenges. Tables often use idiomatic patterns to represent data. Consider tables found in medical research reports (similar to the one shown in Figure 1.1). Medical tables often exhibit both column and row headers; the data cell represents the value of the relationship between the headers. Furthermore, the content in header and data cells is represented using idiomatic patterns often encoding additional

City	*State*	*Mayor*	*Population*
Baltimore	MD	S.C.Rawlings-Blake	640,000
Philadelphia	PA	M.Nutter	1,500,000
New York	NY	M.Bloomberg	8,400,000
Boston	MA	T.Menino	610,000

FIG. 1.2. A simple table representing information about cities in United States of America

metadata in header cells or representing the literals in data cells as "complex objects". Tables from several domains, be it medical or Web, also use acronyms and abbreviations making disambiguation harder. *TABEL* deals with these challenges via a set of pre–processing modules.

The semantics of a table can be captured by the semantics of its sub–parts such as header cells, data cells and relations between headers. A possible approach could be inferring the semantics of the sub–parts individually or sequentially (Mulwad 2010), header cells followed by data cells and finally the relations between headers. There is a possibility of the individual approach failing to capture fine grained semantics (see the following example) or error percolating in the sequential approach if the semantics of header or data cells was inferred incorrectly. Consider the table shown in Figure 1.2. Examining just the column header strings and mapping them to appropriate classes from an ontology (e.g. mapping *City* to *dbpedia-owl:City*) will help us infer that the table conveys information about cities using attributes such as mayor and population. The semantics of the column header can be further enriched by linking the data cell strings in the column to appropriate entities from a knowledge base. For example, linking the data cells' strings from column 1 in Figure 1.2 to appropriate DBpedia entities (e.g. mapping Baltimore to *dbpedia:Baltimore*, Boston to *dbpedia:Boston*), provides additional fine grained semantics that all cities are located in the United States of America, allowing the column header class to be updated to *yago:CitiesInTheUnitedStates*. Similarly, inferring relationships between columns 1 and 2 in Figure 1.2 helps us infer that the cities are the largest cities in their respective states. It is clear that to infer fine grained semantics, it is necessary to incorporate evidence from the entire table instead of analyzing the individual sub parts.

To overcome the problems of individual and sequential approaches, to infer rich fine grained semantics and to capture the interrelation between the table sub parts, *TABEL* jointly infers the semantics using techniques grounded in probabilistic graphical models. It represents a table as an undirected Markov network, in which the header and data cells are represented as nodes; edges in this network capture interaction or dependence between the nodes. We also introduce a novel inference technique, Semantic Message Passing (SMP), as a method of communication between nodes during the joint inference process. *SMP* allows us to incorporate semantics and background knowledge from Linked Data into regular message passing schemes.

Once the semantics are inferred *TABEL* generates a RDF Linked Data representation of the table. We propose an ontology to represent the raw information as well as inferred semantics associated with the table. While the goal of our research is to automatically infer the semantics, achieving 100% accuracy is very difficult, if not impossible. Certain applications and sensitive domains such as healthcare will benefit from a human-in-the-loop paradigm to improve the quality of inferred semantics. *TABEL* allows various models of human interaction with the system.

1.3 Representing the Intended Meaning

We capture the semantics of tables by using knowledge representation techniques grounded in the Semantic Web (Berners-Lee *et al.* 2001) and Linked Data (Berners-Lee 2006). Projects such as DBpedia (Bizer *et al.* 2009), IBM Watson, Google Knowledge Graph and Microsoft's Satori have all demonstrated both the benefits and impact of knowledge representation grounded in Semantic Web technologies and principles. The principles of Linked Open Data (LOD) encourage users to represent *things* using unique dereferenceable HTTP URIs, providing useful information about the *thing* using standards such as RDF. It also encourages interlinking with related resources and *things* from other datasets. For example, the DBpedia URL for the English football player David Beckham[3], when dereferenced, provides information such as his teams, height, date of birth, place of birth and so on in machine understandable format. Ever since its inception in 2006, Linked Open Data has grown rapidly providing a vast amount of knowledge with 295 datasets and more than 31 billion triples[4] publishing facts related to things in a number of domains such as General Knowledge (Wikipedia), Media, Geography, Publications, and Life sciences. With vast knowledge and proven impact, we believe that the Semantic Web standards such as RDF, OWL, and ontologies and datasets from LOD are appropriate choices to represent the inferred semantics of tables.

1.4 Contributions

This thesis presents *TABEL,* a domain independent and extensible framework for inferring the semantics of tables and representing them as RDF Linked Data. To the best of our knowledge, we are the first to propose to the capture semantics of a table as a mapping of header cells to classes, data cells to entities, and capturing relations between header cells (Syed *et al.* 2010). This thesis also presents a probabilistic graphical model to jointly infer the semantics of header, data cells and relations between headers. It also introduces a novel inference technique, *Semantic Message Passing (SMP)*, which incorporates semantics and background knowledge from Linked Data into message passing schemes. We also incorporate a human-in-the-loop paradigm to improve the quality of the inferred semantics and

[3]http://dbpedia.org/resource/David_Beckham

[4]Information as of Sepetember 2011 – http://lod-cloud.net/state/

discuss different models of human interaction with the framework. We present a thorough evaluation of *TABEL* performed against a dataset of tables extracted from the Web and Wikipedia.

We also present an application of this work in the healthcare domain. We develop a system that can automatically produce meta–analyses reports, replacing the existing tedious and largely manual process. We present an evaluation of the quality of inferred semantics from tables in medical research reports and a proof-of-concept interactive interface that will help researchers discover, extract and integrate data from relevant studies to produce meta–analysis reports.

Chapter 2

RELATED WORK

Our work is related to two threads of research, one focused on pragmatically generating RDF from databases, spreadsheets and CSV files (section 2.1), and a more recent one that addresses inferring the implicit semantics of tables (section 2.2).

2.1 Databases to RDF

Several systems have been implemented to generate semantic web data from databases (Sahoo *et al.* 2009; Vavliakis, Grollios, & Mitkas 2010; Polfliet & Ichise 2010), spreadsheets (Han *et al.* 2008; Langegger & Wob 2009) and CSV files (Ding *et al.* 2010; Ermilov, Auer, & Stadler 2013). Most approaches are manual or partially automated requiring users to specify the mapping to be used in the translation process. Automated solutions follow the model of mapping every table as a class, every row as a RDF node and column headers as predicates (i.e. properties) generating local ontology mappings. These approaches fail to reuse classes and properties from existing ontologies; neither do they map data cell values in tables to entities in a knowledge base. Automated techniques that allow users to map data to existing onotologies require the users to specify the mapping to be used. Our approach, on the other hand, maps a table to classes and properties and entities from existing ontologies and knowledge bases (KB) with a goal to generate "Five

Star Linked Data" (Berners-Lee 2006).

2.2 Tables to RDF

Early work in table understanding focused on extracting tables from documents and web pages (Hurst 2006; Embley, Lopresti, & Nagy 2006) with more recent research attempting to understand their semantics.

Wang *et al.* (Wang *et al.* 2012) identify an 'entity column' in the table, and based on its values and the rest of the column headers, maps the table to a concept from Probase (Wu *et al.* 2012). Zwicklbauer *et al.* (Zwicklbauer *et al.* 2013) present an algorithm which annotates column headers with classes from the DBpedia ontology and Wikipedia categories. The algorithm generates a candidate set of classes for every column, by taking a union of the classes associated with the candidate entities for each data cell value in the column. The most frequently occurring class is chosen as the final assignment for the column header. Deng *et al.* (Deng *et al.* 2013) present a scalable algorithm for mapping column headers in a table to a set of top *k* classes from a KB. The similarity between a column and a candidate class is determined using fuzzy set similarity between the contents in the column and entities belonging to the candidate class.

Venetis *et al.* (Venetis *et al.* 2011) associate multiple class labels (or concepts) with columns in a table and identify relations between the 'subject' column and the rest of the columns in the table. Concept and relation identification is based on a maximum likelihood hypothesis, i.e., the best class label (or relation) is the one that maximizes the probability of the values given the class label (or relation) for the column. Zhang (Zhang 2014c; 2014a; 2014b) presents *TableMiner*, a system that maps column headers to classes and data cell values to entities from a KB using a 'two phase bootstrapping' approach. The first phase learns an initial interpretation using partial data from the table whereas the second phase uses the initial interpretation as constraint to interpret the rest of the semantics.

Muñoz, Hogan, & Mileo (Muñoz, Hogan, & Mileo 2014; Munoz, Hogan, & Mileo 2013) use existing entity links in tables found on Wikipedia and relations between them to enrich Linked Data KBs. The primary goal of their work is identify existing relations between entities and suggest that the same relations hold true for other entities in the same columns. KARMA (Knoblock *et al.* 2012; Szekely *et al.* 2013; Knoblock *et al.* 2013) is a user interactive tool which maps structured data such as tables, spreadsheets and CSV into RDF. KARMA generates an RDF representation of a table by mapping columns to semantic types (OWL class or range of a data property) and identifying relations between those types (i.e. relation column headers). It uses a conditional random field to suggest semantic types allowing users to correct incorrect types along the way. Once the semantic types are identified, it further refines the model and identifies relations between those types. While linking data cells is not the primary focus of this work, Szekely *et al.* extend KARMA to build a custom module to link artist names from the Smithsonian American Art Museum dataset to existing DBpedia entities (Szekely *et al.* 2013).

Limaye, Sarawagi, & Chakrabarti (Limaye, Sarawagi, & Chakrabarti 2010) use a graphical model that maps every column header to a class from a known ontology, links table cell values to entities from a KB and identifies relations between columns, relying on Yago for background knowledge. Their graphical model defines node potentials over column headers and data cells and clique potentials over a subset of related variables, namely, column headers and data cells; pair of column headers and the relation between them; and a pair of data cells and relation between them. The joint probability over all the variables is computed as a product of the node and clique potential functions. Both potentials are computed as a dot product between a feature vector over the variables and a weight vector (learned from labeled data) for each node/clique. They use standard message passing during inference to identify an assignment that maximizes the probability.

The majority of the table interpretation systems focus only on part of the table, inferring the semantics of either the column headers, data cells, relations between headers or a combination of any of the two. Limaye, Sarawagi, & Chakrabarti and to certain extent KARMA come close to our work and infer the complete semantics of a table. No system barring KARMA generates a formal RDF Linked Data representation from the inferred semantics. *TABEL* not only infers the complete semantics, but also generates a RDF Linked Data representation of a table, allowing other applications to exploit the encoded information. Our RDF representation also captures a table's structural information allowing it to be reconstructed. The joint inference module (Chapter 3, section 3.3) in our framework uses a factor node representation reducing the number of potential functions as compared to the model presented by Limaye, Sarawagi, & Chakrabarti. Instead of standard inference techniques, we introduce a novel Semantic Message Passing algorithm, which incorporates semantics and background knowledge from Linked Data sources into message passing. Our human-in-the-loop paradigm (Chapter 3, section 3.5) not only allows users to provide feedback or update inaccurate assignment, but also allows the framework to request specific input from the user, making it a two way interaction. Our work on medical tables (Chapter 5) demonstrates that our framework adapts to tables from different domain and tables with different structures.

Chapter 3

FRAMEWORK

We develop *TABEL,* a *domain independent* and *extensible* framework for inferring the semantics of tables and representing it as RDF Linked Data (Figure 3.1). An input table first goes through a *preprocessing* phase which includes modules to handle a number of pragmatic issues such as dealing with idiomatic patterns in medical tables, recognizing and expanding acronyms and stylized literal values, and recognizing commonly encoded data such as addresses, telephone numbers, zip codes, etc. These modules are developed independently and can run either in parallel or sequentially. Newer modules can be added to the framework without hampering the work flow, providing the option of *extensibility* in the future.

The table is then processed by the *Query and Rank* module which queries background Linked Data sources to generate an initial ranked lists of candidate assignments for headers, data cells and relations between headers. The Linked Data sources or the knowledge bases for generating candidates can be adapted and changed based on the domain of the table, providing *domain independence* to the framework.

Once candidate assignments are generated, the *joint inference* module simultaneously infers the semantics of headers, data cells and relations between headers by representing a table as a probabilistic graphical model to capture correlation between sub–parts of a

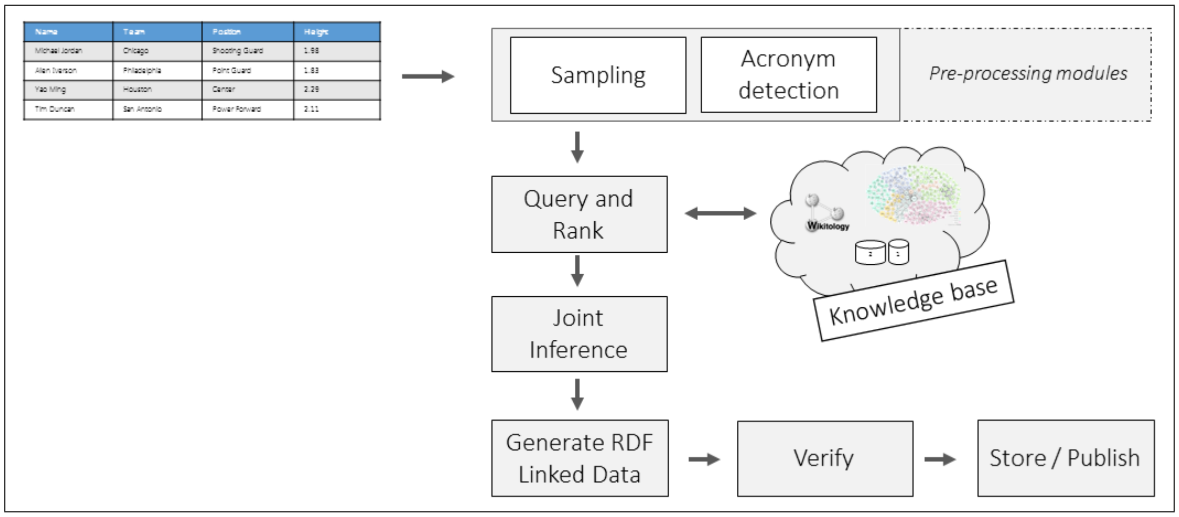


FIG. 3.1. *TABEL* relies on a *joint inference* module to generate a representation of the meaning of the table as a whole.

table and performing inference over the model. After the semantics are inferred, RDF Linked Data triples are generated. Although our goal is to develop an automated system achieving a high level of accuracy, we recognize that practical systems will benefit from or even require human input. We develop different models of human interaction with the framework; Figure 3.1 shows one possible model for a human-in-the-loop paradigm. Users have the option of verifying the inferred semantics and make corrections if necessary by choosing new values from the list of candidates. Finally, the generated linked data can be stored in a knowledge base (KB) or published to the Linked Data cloud. In the rest of this chapter we describe each of the modules in detail.

3.1 Preprocessing

Tables present several pragmatic challenges such as use of acronyms and stylized literal values, tables with large number of rows and columns, the use of idiomatic patterns (especially in medical tables), etc. It is often better to preprocess the table and address such challenges using custom modules, each focusing on a specific task before it passes through the rest of the modules. For instance, a preprocessing module can solely focus on detecting and expanding acronyms and abbreviations, while another can address idiomatic patterns used in medical tables.

We design a modular preprocessing block. Each preprocessing module operates independently of the others working directly on the input table or the output of the previous module. This allows addition of newer modules without affecting existing ones or the rest of the framework. It also allows the option of selecting modules to be used during the preprocessing phase. To demonstrate the modular nature of the preprocessing block, we implement modules to deal with acronyms, abbreviations, idomatic patterns (described in Chapter 5) and commonly encoded data in tables such as telephone numbers, zip codes, etc.

Acronym and Abbreviation Detector: As the name suggests, the Acronym and Abbreviation Detector (AAD) identifies if the header or data cell string is an acronym or an abbreviation and replaces it with the expanded form. AAD is developed on a dictionary based approach maintaining a key value pair of acronym/abbreviation and its expansion (e.g. MD, Maryland). For every string mention in a table cell, AAD checks if the string matches one of the keys. If a match is found, the respective string is replaced with its expanded version.

The current dictionary based approach can be replaced with a more automatic and semantic based approach. Several ontologies have begun encoding acronym and abbreviation information. For example, DBpedia provides information about postal abbreviations for states located in the United States of America using the *postalabbreviation* property. Similarly medical ontologies such as UMLS (Schuyler *et al.* 1993) and SNOMED CT (Stearns *et al.* 2001) also provide both acronyms and synonyms for each medical concept. In the future, it would be possible to construct acronym and abbreviation dictionaries automatically using such ontologies.

Identifying Commonly Encoded Data: Several tables often consist of commonly occurring data such as Social Security Numbers, zip codes, etc. Puranik (2012) developed 'a specialist approach' to identify if a table column consists of either Social Security Numbers, zip codes, airport codes, stock tickers, phone numbers, dates or addresses. The techniques used by the 'specialists' included regular expressions, dictionaries and machine learning based approaches. The module was developed independently and integrated into the framework.

3.2 Query and Rank

The Query and Rank module generates an initial set of candidate assignments for headers, data cells and relations between headers using appropriate Linked Data sources. The choice of the source as a KB depends on the domain of the table. Linked Data KBs such as DBpedia and Yago (Suchanek, Kasneci, & Weikum 2007) provide excellent coverage for general purpose topics such as places, organizations, music, movies, politics, and sports. These can be complimented or replaced with domain specific ones; for example SNOMED CT and UMLS can be used as a compliment or replacement in the case of medical tables. For the rest of chapter, we use the table from Figure 1.2 as an example and DBpedia and Yago as our reference LOD sources.

3.2.1 Generating candidates for data cells

We generate an initial set of candidate entities for every string mention in the data cells in a table using Wikitology (Syed & Finin 2011), a hybrid KB combining unstructured information from Wikipedia with structured information from a number of sources including

```
Input: data cell string (QueryString): {Baltimore}
       row values (RowData): {MD, S.C.Rawlings-Blake, 640,000}
       header string (HeaderString): {City}

Query = wikiTitle: {Baltimore}     (or)
        redirects: {Baltimore}     (or)
        firstSentence: {Baltimore}, {City}     (or)
        types: {City}     (or)
        categories: {City}     (or)
        contents: ({Baltimore}) ^ 4.0, {MD, S.C.Rawlings-Blake, 640,000}     (or)
        linkedConcepts: ({Baltimore}) ^ 4.0, {MD, S.C.Rawlings-Blake, 640,000}     (or)
        propertiesValues: {MD, S.C.Rawlings-Blake, 640,000}

Output: Top "N" matching instances from KB (TopN): {Baltimore, Baltimore_Ravens,
John_Baltimore, ...}
```

FIG. 3.2. Wikitology query to generate candidate entities for data cells in a table

DBpedia and Yago. Wikitology allows queries against several fields of a Wikipedia article including title, first sentence, infobox, content, and categories.

We formulate the Wikitology query for generating candidate entities for data cell values as follows. The data cell value string is used as the query string, along with the contents of the column header and other row values as context in the query. The data cell string is mapped to the title, redirects and the first sentence fields in Wikitology. The redirects field helps in identifying right candidates in cases of misspellings and pseudo names. The column (or row) header string to which the data cell belongs is mapped to the first sentence, types and categories fields, since the header often describes the type of instances present in the column (or row).

The data cell string (with a Lucene query weight boost of 4.0) along with the rest of the values in the row are mapped to contents and the linked concepts fields. Values in a single row are likely to be correlated; hence we map the row data to the linked concept field which captures correlation between Wikipedia articles. The row values (excluding the data cell string) are also mapped to the property values field, since they can possibly represent values of properties associated with the candidate instance. Figure 3.2 describes the Wikitology query using data cell *Baltimore* from the table in Figure 1.2 as an example.

Wikitology returns a ranked list of entities for every query, along with the Wikipedia article page length and approximate Google Page Rank for each entity. We additionally query and obtain DBpedia and Yago classes for each entity. The query for *Baltimore* returns entities such as *Baltimore*, *Baltimore_Ravens*, *John_Baltimore* along with DBpedia and Yago classes such as *City*, *PopulatedPlace*, *Place* and *CitiesInMaryland*, *GeoclassPopulatedPlace* for *Baltimore* and *Person* and *American_conductors_(music)* for *John_Baltimore*.

We re–rank the results returned by Wikitology using an *entity ranker*. Figure 3.3 shows the architecture. We train a Naive Bayes classifier that predicts how likely a candidate entity is to be a correct assignment for a given query string. The classifier is trained on a set of string similarity and popularity metrics as its features, an approach adapted from Dredze *et al.* (2010).

The string similarity metrics provide a syntactic comparison, whereas the popularity metrics helps in cases where disambiguation is difficult due to large numbers of entities having the same or similar names. In the latter case, the most popular entity is often the correct entity. We use the Levenshtein Distance (Levenshtein 1966) and Dice Score (Salton & Mcgill 1986) for each query string – candidate entity pair as the string similarity features and the candidate entity's Wikitology index score, predicted Google Page Rank, and Wikipedia article length as the popularity features.

For each query's *TopN* candidates, the classifier generates a score indicating how likely each entity is the correct assignment. Using this score, the entity ranker re–orders the candidate entity set to produce a final ranked list. We add an additional N.A. (no annotation) entity to each candidate entity set to handle cases where a data cell cannot be mapped to

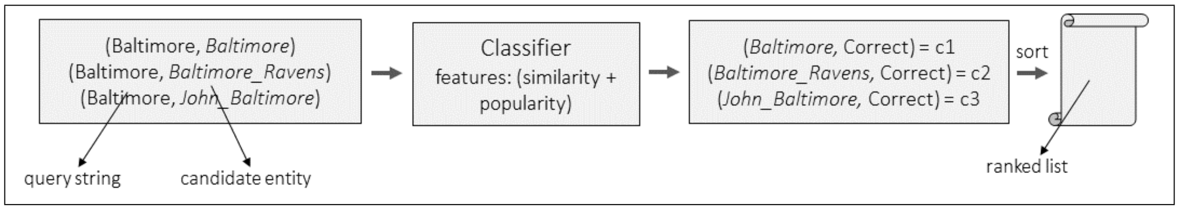

(Baltimore, *Baltimore*)
(Baltimore, *Baltimore_Ravens*)
(Baltimore, *John_Baltimore*)
Classifier
features: (similarity + popularity)
(*Baltimore*, Correct) = c1
(*Baltimore_Ravens*, Correct) = c2
(*John_Baltimore*, Correct) = c3
sort
query string
candidate entity
ranked list

FIG. 3.3. The entity ranker architecture

any of the existing entities in the KB.

3.2.2 Literal Constants

Tables often include columns that contain literal constants like numerical data. Literals cannot be linked to entities from a KB; but rather they represent values of properties. The properties themselves can be associated with other entities in the table. For example, the values in the column *Population* in Figure 1.2 represent values of the property *dbpedia-owl:populationTotal* which in turn is associated with entities of type *dbpedia-owl:City*.

Existing techniques map strings to entities in a KB using textual features (similar to the ones described in section 3.2.1). Similarly, we create a KB encoding properties along with the distributions of their values. Using numerical values as input, we further query against this KB to generate a ranked list of property values for each numerical data cell. We first describe how we generate this KB and follow it up with a description of the query technique.

NumKB – Knowledgebase of Numbers: We capture the distribution associated with values for *datatype* properties in the DBpedia ontology. We begin by identifying the domains associated with each property. Although domains can be obtained by looking up *rdfs:domain*, this information is not available for several properties in DBpedia. A list of domains for every property is enumerated by obtaining the types (i.e.*rdf:type*) associated with the subjects of the triples in which the property is a predicate. Consider *dbpedia-owl:height*; the subjects for this property include *dbpedia:Michael_Jordan*, *dbpedia:David_Beckham*, *dbpedia:Brooklyn_Bridge* and so on. The types associated with these instances, *dbpedia-owl:Athlete*, *dbpedia-owl:SoccerPlayer*, *dbpedia-owl:ArchitecturalStructure*, become the domains for the property. We rank the domains based on the number of the instance–property pairs, assigning a $domainRank$ of 1 to the one with the highest number, 2 to the next and so on. We use these ranks to compute a domain score as follows: $1/\sqrt{domainRank}$. Domain scores help us identify the most popular domains for every property.

We create a property–domain duplet by pairing a property with each of its domain (i.e. height–SoccerPlayer, height–ArchitecturalStructure). We further obtain a list of property values and generate distributional features for each duplet. We sort and ignore the tail (2.5% at each end) to create a subset of the original values. We compute mean (μ) and standard deviation (σ) for the subset and use them to create multiple ranges for each duplet. Ranges are created based on the 68–95–99.7 rule or the three sigma rule. The first range consists of values between $\mu - \sigma$ and $\mu + \sigma$; second range between $\mu - 2\sigma$ and $\mu + 2\sigma$, excluding values from the first range and so on until all the values in the subset are exhausted. We also record the % of the total values present in each range.

Generating candidate properties for literals: We query against *NumKB* to generate candidate properties for literal values (i.e. numerical data) in a column. We use the numerical data cell value along with the column header string as query parameters. NumKB returns a list of property–domain duplet ranges whose property name match the column header string or whose range includes the data cell value. We further re–rank the results using an equally weighted combination of domain score and the % of total values that belong to the range, thus returning a ranked list of properties along with domain information for every numerical data cell value.

3.2.3 Generating candidates for header cells

The initial candidate classes for column (or row) headers are generated from its data cell values. Each data cell is associated with a set of candidate entities; each entity is also associated with a set of classes. For example, we know the classes *City* and *PopulatedPlace* are associated with the entity *Baltimore*. The candidate classes for a header are generated by taking the union of all the classes associated with the candidate entities for all the data cells. For example, the candidate classes for the column header *City* include *City*, *SportTeam*, *Person*, *PopulatedPlace* etc. We generate two separate candidate sets – one for classes from the DBpedia ontology and another from the Yago ontology.

3.2.4 Generating candidates for relation between header cells

Identifying relations between header cells is an important part of table understanding and is modeled by finding appropriate predicates (or properties) from the reference KBs. We generate candidate relations for every pair of column headers in the table based on the entities associated with respective data cell value pairs in those columns. Each data cell has a set of candidate entities, which in turn may be linked to other entities in the reference KB via a set of properties. For example, the DBpedia entities *Baltimore* and *Maryland* are linked via the properties *isPartOf* and *subdivisionName*.

We use these links to generate candidate relations. For a pair of data cells in the same row between two columns, the candidate entity sets for both cells are obtained. For each possible pairing between the entities in both the candidate sets, we query DBpedia and Yago to obtain relations in both direction, i.e., *entityrow1 someproperty1 entityrow2* and *entityrow2 someproperty2 entityrow1*. This gives us a candidate set between pairs of data cell values (e.g., {*someproperty1*, *someproperty2*, *someproperty3* ...}). The candidate set of relations for the entire column pair is generated by taking a union of the set of candidate relations between individual pairs of data cell values. Thus for example, the candidate relations between columns *City* and *State* include *isPartOf*, *capitalCity*, *bornIn* etc. We generate two sets of candidate relations, one from the DBpedia ontology and other from the Yago ontology.

3.3 Joint Inference

Once the initial sets of candidate assignments are generated, the joint inference module assigns values to headers and data cells, and identifies relation between the headers. In our previous work (Mulwad 2010; Mulwad *et al.* 2010), we explored a sequential approach which mapped column headers to classes, mapped data cells to entities using the predicted class as additional knowledge, and finally used both the class and entity assignments to map every pair of columns to a relation from the given ontology. However, this sequential approach led to percolation of error through the stages. An incorrect assignment for the header or data cell would lead to further erroneous assignments. From our previous discussion (section 1.2), it is also evident that rich fine grained semantics can be inferred by capturing correlations and jointly inferring semantics of the sub parts of a table.

Probabilistic Graphical Models (PGMs) (Koller & Friedman 2009) provide a powerful and convenient framework for capturing correlation between variables in a system to compute joint probability over all the variables or a joint assignment of values to all the variables. As the name suggests, the variables are represented as nodes in a graph, with the edges capturing the interaction between them. Constructing a graphical model representation typically involves (i) identifying variables in the system (ii) choosing a graphical model representation and capturing the interaction between the variables and (iii) choosing an appropriate technique to perform inference over the graph. We discuss each of these steps in the context of constructing a graphical model for tables, which is the core of the joint inference module.

3.3.1 Variables

The headers, data cells, and relations between headers are represented as nodes (or variables) in a graphical model representation of a table. Each node in associated with a set of values, which is generated as described in section 3.2. Each data cell is initially mapped to the top ranked entity from its candidate set.

3.3.2 Graphical Representation

There are, broadly speaking, three different techniques for graphical representations in PGMs – directed models (Bayesian network), undirected models (Markov network) and partially directed models. We choose the undirected Markov network model to represent the interactions in a table. In Markov networks, undirected edges indicate symmetrical interaction between the nodes. In the context of tables, an undirected model is a good fit since, as our following discussion will indicate, interaction between the headers, data cell values and relations between headers are symmetrical in nature.

We capture various interactions between sub parts of a table. In a typical well–formed table, each column contains data of a single syntactic type (e.g., strings) that represent entities or values of a common semantic type (e.g., people). For example, the column header *City* represents the semantic type of values in the column and *Baltimore*, *Boston*, and *Philadelphia* are instances of that type. Thus, knowing the type of the column header, influences the decision of the assignment of entities to the data cells in that column and vice-versa. To capture this interaction, we insert an edge between the column header variable and each of the data cell variables in that column. Note that this interaction is symmetrical; i.e. the entities assigned to the data cells equally influence the column header class

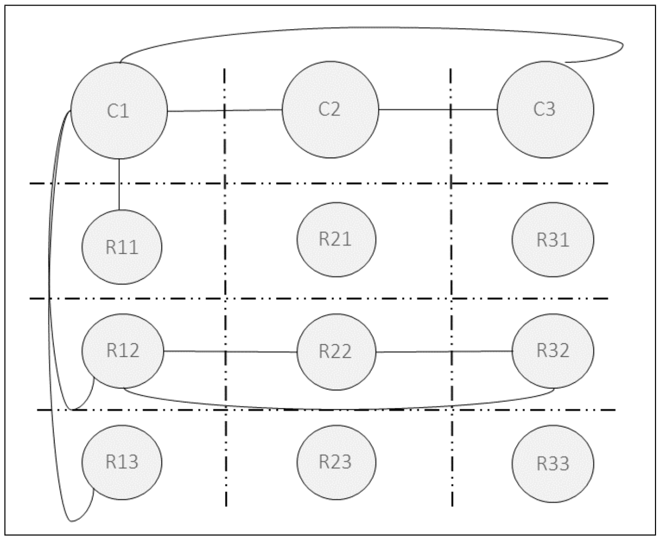


FIG. 3.4. Graph representing interactions between the variables in a simple table. Only some of the connections are shown to keep the figure simple and easy to understand.

as much as the class influences the entity assignments.

Data cells across a row are also related. Consider a data cell with a value *Beetle*; it might refer to an insect or a car. Suppose an adjacent cell has a string value *Red*, which is a reference to a color, and another cell in the same row has the string value *Gasoline*, which is a type of fuel source. As a collection, the data cell values suggest that the row represents values of a car rather than an insect. Thus, the interpretation of each data cell is influenced by the interpretation of others in its row. This correlation when considered between pairs of data cell values between two columns can also be used to identify relations between table columns. To capture this context, we insert edges between all the table cells in a given row.

Similar interactions exist between the headers. By itself, the column header City suggests that columns cells might refer to city instances. However, if the other columns appear to refer to basketball players, coaches and basketball divisions, we can infer that the cities column refers to a team itself. This is an example of metonymy, in which an entity (i.e., the team) is referenced by one of its significant properties (i.e., the location of

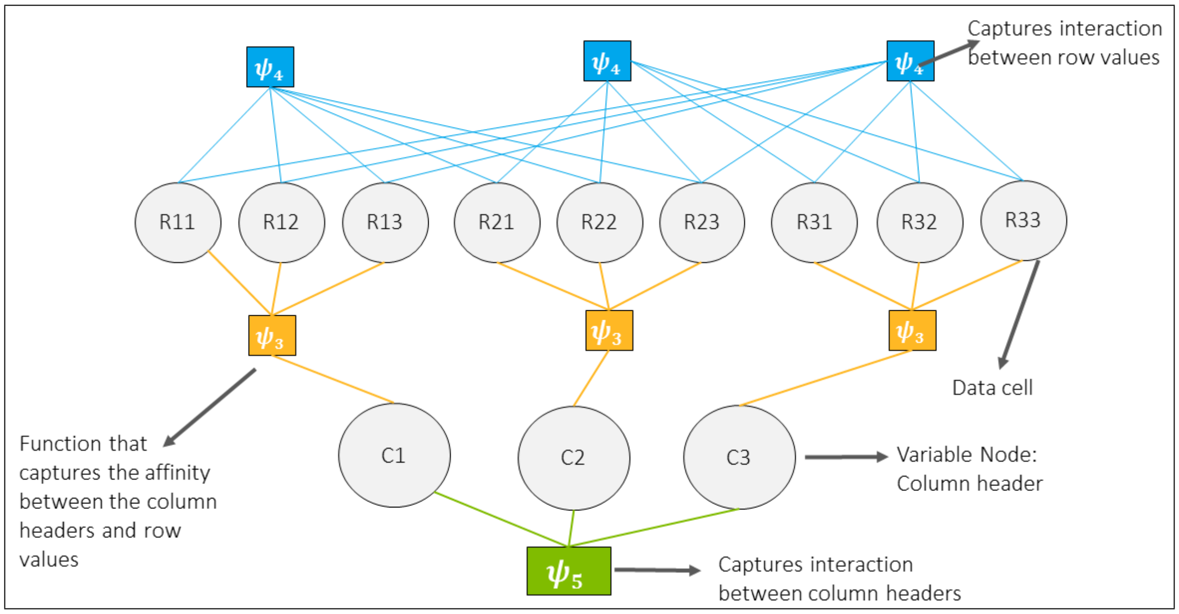

ψ_4
ψ_4
ψ_4
Captures interaction between row values
R11
R12
R13
R21
R22
R23
R31
R32
R33
ψ_3
ψ_3
ψ_3
Data cell
Function that captures the affinity between the column headers and row values
C1
C2
C3
Variable Node: Column header
ψ_5
Captures interaction between column headers

FIG. 3.5. Parameterized Factor Graph for a table.

its base). This interaction is captured by inserting edges between column header nodes. Figure 3.4 shows interactions between the various nodes; column headers are represented by C_i where $i \in \{1,2,3\}$ and data cells are represented by R_{ij} where $i, j \in \{1,2,3\}$.

3.3.3 Inference

To perform any meaningful inference over the graph, it needs to be parameterized. Markov networks are parameterized using factor nodes. Figure 3.5 shows the parameterized factor graph for a table. The graph's square nodes represent what are known as '*factor nodes*', which compute and capture affinity or agreement between interacting variable nodes. The factor node ψ_3 computes the agreement between the class assigned to column header and entities linked to the data cells in that column; ψ_4 between the entities linked to the data cells for a given pair of columns and ψ_5 between classes assigned to the column headers. Typical inference techniques such as Message Passing or Belief Propagation group all nodes that are correlated using common nodes between the groups as a

Algorithm 1 *Semantic Message Passing*

```
1: Let $Vars$ be the set of variable nodes and $Factors$ be the set of factor nodes in the graph.
2: for all $v$ in $Vars$ do
3:     Let $F'$ be the set of factor nodes $v$ is connected to.
4:     for all $f'$ in $F'$ do
5:         $v$ sends its current assignment (value) to $f'$.
6:     end for
7: end for
8: for all $f$ in $Factors$ do
9:     Compute agreement between the received assignments.
10:    Identify variable nodes that may have sent an incorrect assignment.
11:    Send a NO-CHANGE message to all nodes that are in agreement, as determined by $f$.
12:    Send CHANGE message and semantics of expected assignment to all the nodes that have an incorrect assignment, as determined by $f$.
13: end for
14: for all $v$ in $Vars$ do
15:    Let $Messages$ be the set of messages received by $v$.
16:    If all $m \in Messages$ are NO-CHANGE, do nothing.
17:    If few or all $m \in Messages$ are CHANGE, update the current assignment by choosing a new one from the candidate set which satisfies the semantics sent by the factor nodes.
18: end for
19: Repeat till convergence.
```

mode of communication or for passing messages between groups. While we retain the idea of grouping correlated nodes in a factor, we modify existing techniques to incorporate the semantics and background knowledge from Linked Data into the inference process. We propose and implement a new technique, *Semantic Message Passing (SMP)*, an algorithm conceptually similar to the idea of Message Passing/Belief Propagation, but one that factors in semantics into messages.

Algorithm 1 gives a high level overview of *Semantic Message Passing*. The variable nodes in the graph send their current assignment to all the connected factor nodes. For example, R_{11} sends its current assignment to the factor nodes ψ_3 and ψ_4 (see Figure 3.5).

Once the factor nodes receive values from all connected variable nodes, they compute agreement between the values. Thus, in one of the iterations, ψ_3 might receive the class *City* and entity assignments *Baltimore_Ravens*, *Philadelphia*, *New_York* and *Boston*. The goal of ψ_3 is to determine if all the assignments agree and, if not, identify the outliers.

In this case ψ_3 identifies *Baltimore_Ravens* as an outlier and sends a CHANGE message to R_{11}, together with its semantic preferences for a new, alternate assignment that R_{11} should choose. In our example, ψ_3 informs R_{11} of its preference for an update to an entity of type *City*. To the rest of the variable nodes, ψ_3 sends a NO-CHANGE message. This process is performed by all factor nodes. Once a variable node receives messages from all of its connected factor nodes, it decides whether to update its assignment.

If a variable node receives a message of NO-CHANGE from all its connected factor nodes, it indicates that the current assignment is in agreement with others assignments. If it receives a CHANGE message from some or all factor nodes, it updates its current assignment, taking into consideration the semantic preferences provided by the factor nodes. The entire process repeats until convergence, i.e., agreement over the entire graph is achieved. A hard convergence metric could be to repeat the process until no variable node receives a CHANGE message.

Our *Semantic Message Passing* algorithm, thus, not only detects individual variable nodes that have incorrect assignments, but provides the nodes with guidance on choosing a new value, thus incorporating *semantics* into messages. This capability requires defining *semantically-aware* factor nodes that can perform such functions. We describe and implement two such semantically-aware factor nodes, ψ_3 and ψ_4. We also describe the process by which a variable node updates its values based on the messages received and our metric for convergence in this graph.

ψ_3 **– Column header and data cell value agreement function.** Algorithm 2 gives an

Algorithm 2 ψ_3 – Column Header – Row Values Agreement

```
1: Let $Yago$ and $DBp$ be the column header $C_i$'s candidate class sets from Yago & DBpedia respectively and $RowVals$ be the set of data cell values in $C_i$.
2: **for all** $r$ in $RowVals$ **do**
3:     Let $e$ be the currently assigned entity to $r$ and $t_y$, $t_d$ be the set of Yago and DBpedia classes for $e$.
4:     Majority Vote Score: For each class in $Yago$, increment the score by 1 if it is present in $t_y$. Similarly, using $t_d$, increment the scores for classes in $DBp$.
5: **end for**
6: **for all** $y$ in $Yago$ **do**
7:     Get the "granularity" score for $y$.
8: **end for**
9: Order the classes in $Yago$ in descending, first by vote scores and then by granularity scores. Let $top_y$ be the Yago class with maximum vote score and best "granularity score".
10: Order the classes in $DBp$ in descending by vote scores. The class at the top is the one with maximum votes. Let this class be $top_d$.
11: Let $topClassScore_y$ and $topClassScore_d$ be scores for $top_y$ and $top_d$ respectively, where $topClassScore = numberOfVotes/numberOfRows$
12: If the scores are below threshold, send LOW-CONFIDENCE and NO-CHANGE messages to all the data cell values in $RowVals$. Update $C_i$'s class annotations to NO-ANNOTATION.
13: If the scores are above threshold, update $C_i$'s class annotations to $top_y$ and $top_d$. Send appropriate CHANGE or NO-CHANGE messages to data cell values in $RowVals$ as shown next.
14: **for all** $r$ in $RowVals$ **do**
15:     Let $e$ be the currently assigned entity to $r$ and $t_y$, $t_d$ be the set of Yago and DBpedia classes for $e$.
16:     If $t_y$, $t_d$ for $e$, do not contain either $top_y$ or $top_d$, send CHANGE message. Send $top_y$, $top_d$ as the classes for the entity that $r$ should update to; else send the NO-CHANGE message.
17: **end for**
```

overview of the column header and data cell value agreement function. The factor node ψ_3 computes agreement between the class assigned to the column and the entities assigned to its cell values i.e. agreement between the column assigned type *City* and candidate cell assignments *Baltimore_Ravens*, *Philadelphia*, *New_York* and *Boston*.

Recall that at the end of the *Query and Rank* phase, every data cell value has an initial entity assignment and every column header has a set of candidate classes. Every column header C_i maintains two separate sets of candidate classes – one from the Yago ontology and other from the DBpedia ontology. Each cell value in a column is mapped to an initial entity e which is also associated with a set of Yago and DBpedia classes. The initial entities assigned to a column's cell values perform a majority vote over the Yago and DBpedia class sets to pick a top class from each set. Each entity votes and increments the score of a class from the candidate set by 1 if it is also present in the class set associated with e.

The Yago and DBpedia candidate class sets are ordered by votes. ψ_3 computes the top score for each of the top classes. The top class score is equal to the number of votes for the top class divided by the number of rows in the column. Ideally, we want to pick more specific classes (e.g., City) over general classes (e.g., Place) when making an assignment to the column headers. The *granularity* score is computed by simply dividing the number of instances that belong to the class by the total number of instances in the KB and subtracting the result from one. This assigns a higher score to specific classes and a lower score to general classes. If multiple Yago classes get voted as the top class, ψ_3 uses the *granularity* score as tie-breaker, choosing the class with a higher score.

Once the top class(es) are identified and their scores computed, ψ_3 determines if they can be used in the process of identifying cell values with incorrect assignments. It checks whether the top scores for the classes are below a certain threshold. If so, it implies lower confidence and low agreement between the entities assigned to the data cell values and that the top classes cannot be relied upon. In such scenarios, ψ_3 sends a message of LOW-

CONFIDENCE and NO-CHANGE to the variable nodes and also maps the column header class to NO-ANNOTATION.

If scores for both the top Yago and DBpedia classes are above the threshold, ψ_3 assigns both the classes to the column header and uses them in the process of identifying cell values with incorrect assignments. However if either class is below threshold, it checks if the classes are aligned. We define the two classes to be aligned if either the DBpedia class is a subclass of the Yago class or vice-versa. The subclass relation between the DBpedia and Yago classes is obtained via the *PARIS* project (Suchanek, Abiteboul, & Senellart 2011).

If the classes are aligned, ψ_3 picks both the classes as the top Yago and top DBpedia assignments for the column header respectively. If the classes are not aligned, ψ_3 discards the top class which has a top class score below the threshold. ψ_3 further uses the top classes to identify variable nodes with incorrect assignments.

All variable nodes whose currently assigned entity e include the top class(es) in their class set are sent a message of NO-CHANGE. The variable nodes whose current entity assignment diagree with the top class(es) are sent a CHANGE message. These variable nodes are also provided with the top class(es) as semantic preferences for their next entity assignment. ψ_3 also sends the top class scores as a confidence score to all the variable nodes.

ψ_4 **– Relations between pair of columns.** Algorithm 3 gives an overview of the function that identifies relations between pairs of columns in a table. The goal of the factor node ψ_4 is to discover if a relation exists between a pair of columns, say *City* and *State*, and, if so, to use it as evidence to uncover any incorrect entity assignments in the columns' cells. At the end of the *Query and Rank* phase, every data cell value has an initial entity assignment e and the pair of column headers has a set of candidate relations. The initial assigned pair of entities in the two columns performs majority voting to select the best possible relation from the candidate relation set. Each pair of entities in every row between the two columns

Algorithm 3 ψ_4 – Relation between pair of columns

```
1: Let $YagoCandidateRels$ and $DBpCandidateRels$ be the set of Yago and DBpedia candidate relations between the columns $c_i$ and $c_j$.
2: Let $numberOfRows$ be the total number of rows in the table.
3: **for** k = 1 **to** $numberOfRows$ **do**
4:     Let $r_{i,k}$ and $r_{j,k}$ be the data cell values from columns $i$ and $j$ at row $k$.
5:     Let $e_{i,k}$, $e_{j,k}$ be the currently assigned entities to $r_{i,k}$ and $r_{j,k}$.
6:     **for all** $rel$ in $YagoCandidateRels$ **do**
7:         Majority Voting : If $< e_{i,k} >< rel >< e_{j,k} >$ or $< e_{j,k} >< rel >< e_{i,k} >$ is true, increment the score of $rel$ by 1
8:     **end for**
9:     Repeat the same majority voting process for candidate relations from $DBpCandidateRels$
10: **end for**
11: Sort $YagoCandidateRels$ and $DBpCandidateRels$ in descending order. The relations at the top of the sets are the ones with maximum votes. Let $topRel_y$ and $topRel_d$ be the top Yago and DBpedia relations respectively. Let $topRelScore_y$ and $topRelScore_d$ be scores for $topRel_y$ and $topRel_d$, where $topRelScore$ = $numberOfVotes$ / $numberOfRows$
12: **if** $topRelScore_y$ and $topRelScore_d < relThreshold$ **then**
13:     Send message LOW-CONFIDENCE and NO-CHANGE to the data cell values in columns $c_i$ and $c_j$.
14:     Update relation annotation to NO-ANNOTATION.
15: **end if**
16: **if** $topRelScore_y >= relThreshold$ **then**
17:     Update the top Yago relation between the columns $c_i$ and $c_j$ to $topRel_y$.
18: **end if**
19: Similarly update the top DBpedia relation between columns $c_i$ and $c_j$.
20: **for** k = 1 **to** $numberOfRows$ **do**
21:     Let $e_{i,k}$, $e_{j,k}$ be the currently assigned entities for $r_{i,k}$ and $r_{j,k}$.
22:     If $< e_{i,k} >< topRel >< e_{j,k} >$ and $< e_{j,k} >< topRel >< e_{i,k} >$ is false, send CHANGE message to the data cell value. Send top relation the data cell entity assignment should participate in. ($topRel$ can be either $topRel_y$ or $topRel_d$)
23:     Else, send NO-CHANGE message to both $r_{i,k}$ and $r_{j,k}$.
24:     Send $topRelScore$ as confidence score associated with the message.
25: **end for**
```

votes for a relation rel. The candidate relation rel's score is incremented by 1 if $< e_{i,k} >$ $< rel >$ $< e_{j,k} >$ or $< e_{j,k} >$ $< rel >$ $< e_{i,k} >$ is true (*here i, j refer to two columns and k refers to entities assigned to data cells from the k th row between the two columns*). Factor node ψ_4 queries Yago and DBpedia separately to check if the relations exists. ψ_4 also maintains two separate sets of candidate relations (one for Yago and the other for DBpedia), ordered by votes.

ψ_4 further computes $topRelScore$ for both the top Yago and DBpedia relations by dividing the number of votes for the top relation with the number of cells in the column. If the score is below a certain threshold, the relations are discarded. ψ_4 sends LOW-CONFIDENCE and NO-CHANGE messages to all of the data cell values in both columns and updates the relation between the columns to NO-ANNOTATION. If the scores are above threshold, the top Yago and DBpedia relations between the two columns are updated. ψ_4 then revisits the pair of entities from the columns to discover possible incorrect assignments. For every currently assigned entity e in the two columns, ψ_4 checks if e appears as a subject or object of the top relations (depending upon the relation direction; either Yago or DBpedia). If e satisfies this constraint, a NO-CHANGE message is sent to the data cell; a CHANGE message is sent otherwise along with the relation information as characteristics the data cell should use for picking the next entity assignment. $topRelScore$ is sent as confidence score associated with the message to all the data cells.

Once all the semantically–aware factor nodes send their messages variable nodes pick a new assignment and the entire process repeats. We next describe the process of updating the an assignment.

Updating entity annotations for row cell value variables. Every data cell r in the table receives messages from two types of factor nodes – column header factor nodes (ψ_3) and relation factor nodes (ψ_4). While r will receive only one message from the column header (since r belongs to only one column), it might receive multiple messages from relation factor nodes if its column is related to several columns in the table. For example, the column *City* is associated with columns *State*, *Mayor* and *Population*. r can also receive conflicting messages – some factor nodes might send a CHANGE message, while others a NO-CHANGE message.

If all of the messages received by r are NO-CHANGE, r does not update. If all messages received by r are CHANGE, it decides to update its assignment. In the case of conflicting messages, r uses the confidence score sent by each factor node along with the message to compute the average score associated with the CHANGE messages and compares it against the average score associated with the NO-CHANGE messages. If the average CHANGE message score is higher, r updates its current assignment, otherwise it does not.

When r chooses to update its current assignment, it picks a new assignment based on the semantic preferences sent by the factor nodes. For example, the data cell value in the first column of the table in Figure 1.2 might receive messages to update to an entity which will have a type *City* and is the subject of relations *isPartOf* and *hasMayor*. The row cell value r iterates through its ranked list of candidate entity set and picks the next best entity satisfying all the semantic preferences specified in the message. In cases where r cannot find an entity that satisfies all of the constraints, it orders them based on the confidence scores associated with the respective messages and attempts to pick an entity assignment that satisfies the highest rank combination. For example, if there were three preferences, ranked *1*, *2* and *3*, r will first attempt to find an entity that satisfies*[1,2,3]* followed by *[1,2]* ; *[1,3]*; *[2,3]*; *[1]*; *[2]* and so on. If r is unable to find an entity that satisfies any of the preferences, then it updates its current assignment to NO-ANNOTATION.

An exception to this process occurs when the candidate entities for the data cell all have low confidence (i.e., below the threshold ($index\_threshold$)). This is typically the case if the entity is absent from the knowledge base. In such cases, the algorithm maps the data cell value to NO-ANNOTATION rather than linking to any candidate. If the column header is mapped to NO-ANNOTATION, the row cell values retain the top ranked entity assignment as suggested by the entity ranker.

Halting condition. Once the data cell values update and send their new assignments to the factor nodes, the entire process repeats. Ideally, the process should be repeated until the best possible assignments are achieved; i.e., repeat until no variable node receives a CHANGE message or none of the variable nodes select a new assignment. Practically, this a hard convergence metric and it is often not achieved. We define an alternate convergence metric. At the end of every iteration, *Semantic Message Passing*, checks the number of variables that have received a CHANGE message. If the number of variables is lower than the threshold required to update the column header or relation annotation, the process is stopped, else the process continues. In cases where the convergence metric fails, the inference process is stopped after ten iterations. The assignments at the end of the final iteration are chosen as final values.

3.4 RDF Linked Data

We develop the *TABLES* ontology (Figure 3.6) to model and represent a table and its inferred semantics as a set of RDF Linked Data triples. Every table cell (header and data cell) is represented via an instance of one of the classes in the ontology. Header cells are represented as instances of either the *ColumnHeader* or *RowHeader* class; data cells are represented as instances of the *DataCell* class.

The raw string value of every table cell is captured via the *cellLabel* property. The domain of this property is the class *TableCell* (which is the super class of *HeaderCell* and

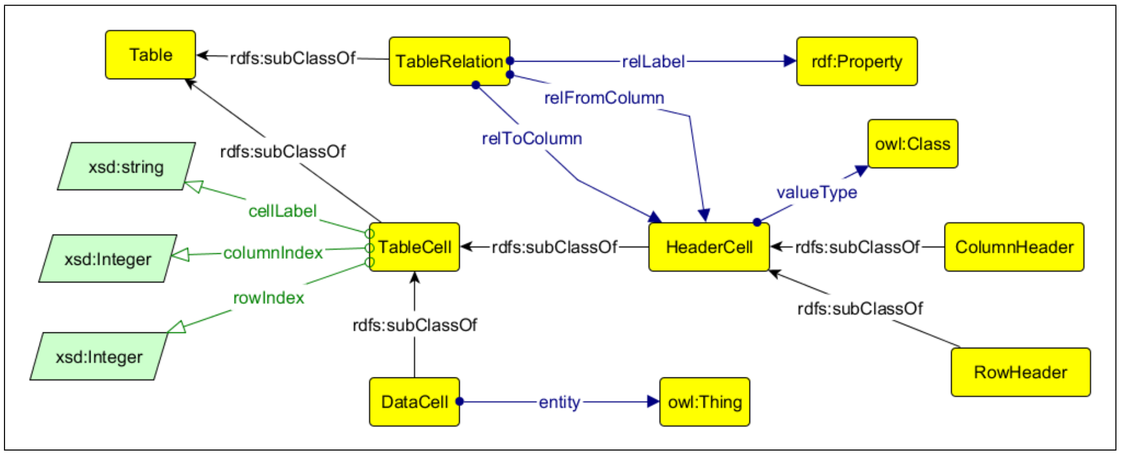


FIG. 3.6. Visualization of the *Tables* ontology. The prefix 'xsd' is for XML schema, 'rdfs' is for the RDF Schema, 'owl' is for the OWL Ontology and 'rdf' is for the RDF vocabulary.

DataCell) and its range is string. We also capture the original position of the table cell via the *columnIndex* and *rowIndex* properties allowing the table to be reconstructed. Instances of the *ColumnHeader* class are associated with the *columnIndex* property; instances of the *RowHeader* are associated with *rowIndex* property and instances of *DataCell* with both the *columnIndex* and *rowIndex* properties. Both properties have the domain *TableCell* and range as integer. Every header cell is mapped to an appropriate class from an ontology; this class is captured by the *valueType* property. Entity assignments for data cells are captured using the *entity* property.

We create an additional class *TableRelation* to capture inferred relations between header cells. The properties *relFromColumn* and *relToColumn* capture the header cells participating in the relation. The domain for both properties is *TableRelation* and the range is the class *HeaderCell* (which is the super class of both *ColumnHeader* and *RowHeader*). The property *relLabel* captures the inferred relation. Figure 3.7 shows RDF triples for a subset of the table from Figure 1.2.

```
#Triples representing the first column header
tab:cell_01 a tab:ColumnHeader;
   tab:cellLabel "City"^^xsd:String;
   tab:columnIndex "1"^^xsd:Integer;
   tab:valueType dbpedia-owl:City.
#Triples representing the second column header
tab:cell_02 a tab:ColumnHeader;
   tab:cellLabel "State"^^xsd:String;
   tab:columnIndex "2"^^xsd:Integer;
   tab:valueType dbpedia-owl:AdministrativeRegion.
tab:cell_11 a tab:DataCell;
   tab:cellLabel "Baltimore"^^xsd:String;
   tab:columnIndex "1"^^xsd:Integer;
   tab:rowIndex "1"^^xsd:Integer;
   tab:entity dbpedia:Baltimore.
tab:HeaderRelation_12 a tab:TableRelation;
   tab:relFromColumn tab:cell_01;
   tab:relToColumn tab:cell_02;
   tab:relLabel dbpedia-owl:isPartOf.
```

FIG. 3.7. Subset of RDF for data in Figure 1.2. The prefixes 'tab' is for our *Tables* Ontology; 'dbpedia' and 'dbpedia-owl' is for DBpedia ontology and 'xsd' is for XML schema.

3.5 Human-in-the-loop

Although our goal is to develop an automated system achieving a high level of accuracy, we recognize that practical systems will benefit from or even require human input. We discuss three possible models of human interaction with the framework and how *TABEL* incorporates user feedback into the inference process. A user can interact *before* the inference begins, *during* the inference process and *after* the inference process. Figure 3.1 shows user interaction *after* the inference is complete.

User interaction *before* inference: There are two different scenarios of user feedback before the inference process. In the first scenario, user feedback is during the preprocessing phase and in the form of highlighting relevant parts of the table. User could mark a column in the table to be ignored (for example, a column that includes serial numbers *1–10* used to order elements) if no relevant semantics can be inferred. In cases of large tables, users could sample and reduce the size by selecting smaller numbers of rows or columns which *TABEL* should use during the inference process. User feedback during the preprocessing phase will help the framework focus on relevant sections of the table and thus perform a better job at inferring its semantics.

The second scenario of user feedback is after the Query and Rank phase, before the Joint Inference module. A user can provide initial assignments for header and data cells and relations between headers by choosing a value from the ranked candidate set. If the value is missing in the candidate set, an interface allows the user to search the KB and add the new desired value to the set. The user assignments are treated as *true* values by *TABEL* during the joint inference process. These assignments are not changed; all other assignments are selected such that they are compatible to the user assignments.

User assignment for a column header impacts the behavior of factor node ψ_3. For example, user could assign the class *dbpedia-owl:City* to the first column from table in Figure 1.2. The joint inference module will make all other assignments compatible to *dbpedia-owl:City*. Before *Semantic Message Passing* starts, data cell variables nodes in the graph update their assignment to the highest ranked entity of the user assigned class from the candidate set. In this case, the variable nodes will choose an entity of type *dbpedia-owl:City*.

During *Semantic Message Passing*, data cell variable nodes may update their assignment based on messages received from the other factor node ψ_4. However higher preference will be given to the user assigned class when choosing a new entity assignment. For example, ψ_4 might send the relation *dbpedia-owl:isPartOf* as a semantic preference for a new entity assignment to the variable node *Batlimore*. The variable node will attempt to pick a new entity assignment that satisfies both *dbpedia-owl:City* and *dbpedia-owl:isPartOf*; dropping the latter as constraint if it does not find such a entity. That is, the variable node will always stick to an assignment whose type is the same as a user assigned class.

User assignment for relations between header cells impacts the behavior of the factor node ψ_4. Consider the case where a user assigns the relation *dbpedia-owl:isPartOf* between the columns *City* and *State*. As in the case of user assigned column header classes, the data cell variable nodes of both the columns will update their assignment to the highest ranked entity which participates in the user assigned relation (in our example it will be *dbpedia-owl:isPartOf*). During *Semantic Message Passing* variable nodes may update their assignment based on messages received from the column header factor node ψ_3. However, in this case preference will be given to relation semantics over the column header class when the variable node selects a new entity assignment. Variable nodes will always select an entity that satisfies the user specified relation constraint.

User assignment for a data cell impacts the behavior of both ψ_3 and ψ_4. For example, the user might map the data cell *Seattle* to the entity *dbpedia:Seattle*. The respective candidate header class set and relation set are reduced to include only classes and relations of the user assigned entity. In cases where user assigns entities to multiple data cells, the classes and relation from those entities are merged to produce a reduced size candidate class and relation set. The rest of the *Semantic Message Passing* process remains the same; albeit now operating on a smaller size candidate set.

A user is not restricted to a single type of assignment; i.e., the user might simultaneously make assignments for column header, data cells and relation between columns. The behavior of the factor nodes and variable nodes is impacted based on these assignments. There are four possible combinations for user assignments – (i) class and entity assignment, (ii) relation and entity assignment, (iii) class and relation assignment and (iv) class, relation and entity assignment.

(i) Class and Entity assignment: A user simultaneously assigns a class to a column header and entity to a data cell in the same column. For example, a user might map the column header *City* to *dbpedia-owl:City* and the data cell *Seattle* to *dbpedia:Seattle*. The behavior of the factor node ψ_3 is the same as in the case of a user assigned class (as described above). Data cells in the column will always select an entity of the user assigned class. The behavior of ψ_4 is impacted by the user assigned entity. As described above, the size of the candidate relation set is reduced for all pair of columns in which the entity is present.

(ii) Relation and Entity assignment: This scenario is similar to (i); the user assigns a relation between a pair of columns. A user might specify the relation *dbpedia-owl:isPartOf* between the columns *City* and *State* along with the entity assignment *dbpedia-Seattle* for the data cell *Seattle*. The behavior of factor node ψ_4 is the same as in the case of user assigned relation and the candidate class set for the column in which the entity is assigned is reduced as described above.

(iii) Class and Relation assignment: The user assigns a class and a relation to the same column. The behavior for ψ_3 and ψ_4 is the same as described above in the cases of user assigned class and relation respectively. Data cells will select an entity assignment that satisfies both the semantic class and relation constraints.

(iv) Class, Relation and Entity assignment: After the class and relation assignment, an entity assignment in the same column does not offer additional benefits to the inference process. The behavior of the factor and data cell variable nodes is the same as in (iii).

User interaction *during* inference: User interaction during the inference process not only helps the user guide the process, but also provides *TABEL* an opportunity to request specific inputs from the user. At the end of every iteration during *Semantic Message Passing*, the framework highlights assignments for headers, data cells and relations between headers with low confidence. It uses threshold scores from the factor nodes to determine low confidence assignments. For example, *TABEL* might fail to infer the class to be assigned to the column *Mayor* and ask the user to select an appropriate class. Another scenario where the framework requests user input is in the case of stubborn variable nodes. *TABEL* highlights variable nodes that changes it assignment in every iteration, never settling down on a value. Values assigned by the user during inference process are also treated as *true* values. Besides correcting assignments highlighted by the framework, users can update assignments for other variable nodes, in the same manner as they would have before the inference process began. Once the user updates assignments, *Semantic Message Passing* alters its behavior for the remaining iterations as described above. The user also has the option of halting the inference process, if he/she is satisfied with the inferred semantics.

User interaction *after* inference: Finally, there is the option of user's interaction with the framework after the inference process is complete. At this point, the user can correct any inaccurate assignments by selecting a new value from the respective candidate set or new value by directly searching the KB. The RDF Linked Data representation is accordingly updated.

Chapter 4

EVALUATION

We begin by describing the datasets and the experimental setup used for evaluation followed by sections on column header, relation and data cell annotation quality. We end the chapter by briefly describing the impact of user feedback (human in the loop) on our framework.

4.1 Dataset

We use a dataset of tables obtained from the Web and Wikipedia shared by Limaye, Sarawagi, & Chakrabarti (2010). The original dataset consists of four distinct subsets as shown in Figure 4.1. The **Wiki_Manual** subset consists of non infobox tables extracted from Wikipedia articles. The column headers, data cells and relations between columns

Dataset	*Col & Rel*	*Cell Value*	*Avg.[Col,Row]*
Wiki_Manual	25	39	[4,35]
Web_Manual	150	371	[2,36]
Web_Relation	28	–	[4,67]
Wiki_Link	–	80	[3,16]

FIG. 4.1. Number of tables and the average number of columns (col) and rows in the sets used for column header, cell value and relation annotation. Average is over the number of tables used in cell value annotation.

are manually annotated with class, entity and relations using the Yago ontology. The **Web_Manual** subset includes tables extracted from web pages; column headers, data cells and relations between columns are manually annotated using the Yago ontology. Limaye, Sarawagi, & Chakrabarti point out that header and data cell text tends to be more noisy in this subset.

The **Web_Relation** subset includes tables extracted from web pages, but only relations between table columns are annotated with relations from the Yago ontology. The **Wiki_Link** subset includes non infobox tables extracted from Wikipedia articles. Tables in which more than 90% of the data cells were linked to a Wikipedia article were included. The links are used as ground truth for data cell value annotation; column header and relations between column headers were not annotated.

While the original tables include ground truth annotations in the form of Yago classes and properties, we could not use these assessments for column headers and table relations because we use data from both the DBpedia and Yago ontologies. We retain ground truth for data cell annotations and develop our gold standard for column header classes and header cell relations with the help of human evaluators. We begin by running *TABEL* with lower threshold values in factor nodes ψ_3 and ψ_4, generating a large number of possible column header classes and relations between column headers. Any class or relation that received at least 5% of the votes was included in the evaluator's set (i.e. the threshold values in ψ_3 and ψ_4 were set to 0.05).

We presented these candidates along with the raw tables to human evaluators, who marked each class and relation as *vital*, *okay* or *incorrect*, a strategy similar to the one adopted by Venetis *et al.* (2011). For example, annotators could mark the class *dbpedia-owl:City* as *vital*, *dbpedia-owl:PopulatedPlace* as *okay* and *dbpedia-owl:Person* as incorrect for the column *City* from the table in Figure 1.2. Thus each column class and relation can have multiple *vital* and *okay* labels as per evaluator judgment. The task of generating gold standard evaluation was split between eight human evaluators who were graduate students in Computer Science at the University of Maryland, Baltimore County.

Figure 4.1 gives a summary of the tables used in our experiments. We evaluated column header and relation annotation over 203 tables and data cell annotation over 490 tables. Figure 4.1 also reports the average size of the tables in each of the subsets.

4.2 Experimental Setup

For the evaluation that follows, *TABEL* was initialized with the following values: for every data cell we chose the top 25 candidate entities from the entity ranker; the column header top score threshold used by ψ_3 and the relation top score threshold used by ψ_4 were set to 0.5; *index_threshold* used by a data cell to determine low confidence entities was set to 10. The number of joint inference model iterations is set to 5. For every iteration after the 5th, the halting condition decides whether the inference process should continue or halt. If the halting condition fails, the inference process is hard stopped at the end of the 10th iteration.

4.3 Column Header Annotations

We use *TABEL* to generate a ranked list of at most ten class annotations for every column, with NO-ANNOTATION as the single value if no appropriate class was found. We compared the set of system generated classes to those obtained from human evaluators. We first measure the quality of predicted class labels by calculating the fraction of *vital* and *okay* class labels at rank 1.

More than 78% of the top ranked class labels (from the DBpedia ontology) for Wiki_Manual, Web_Manual and Web_Relation are relevant labels, i.e., either *vital* or *okay* (see Figure 4.2). A further look at the distribution between *vital* and *okay* labels indicate a

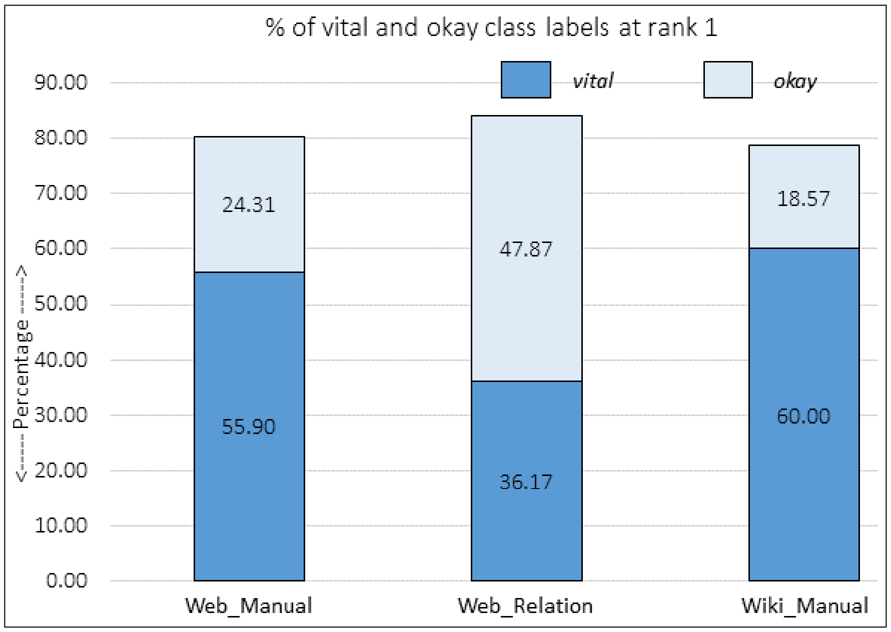


FIG. 4.2. Percentage of *vital* and *okay* DBpedia class labels at rank 1.

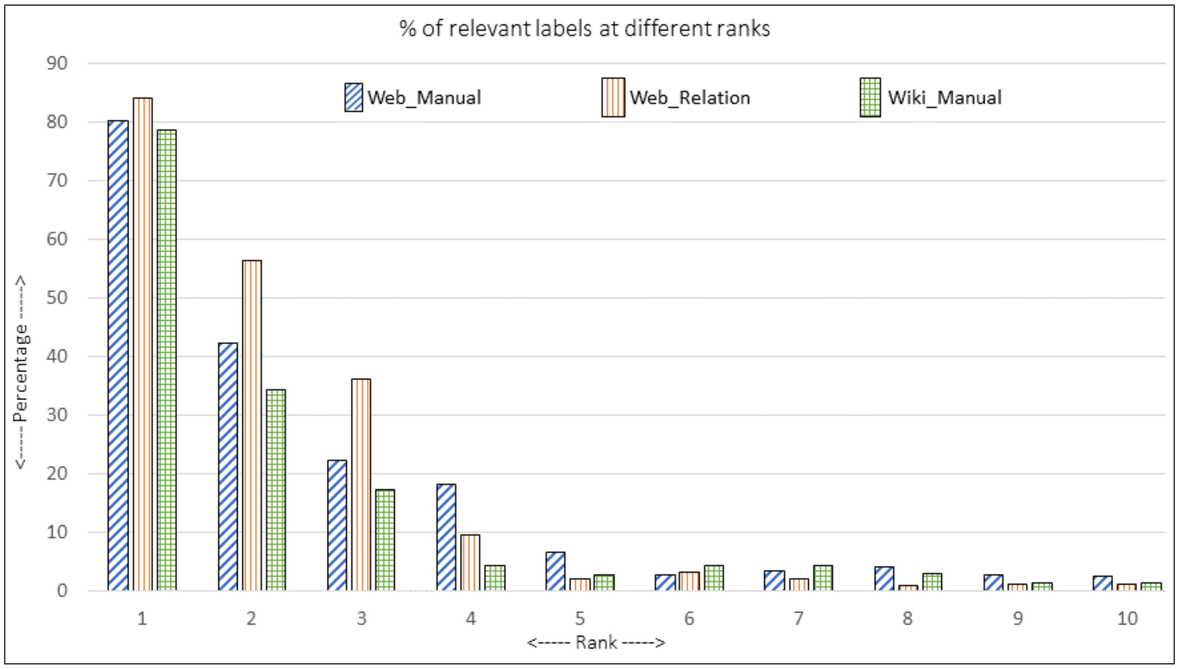


FIG. 4.3. Percentage of relevant class labels at ranks 1–10.

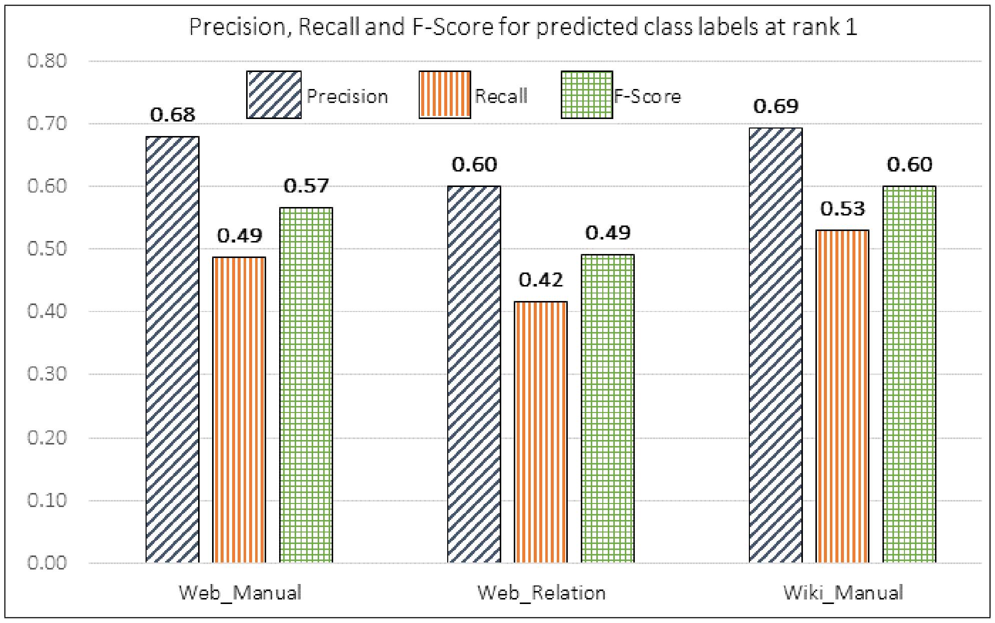


FIG. 4.4. Percentage of relevant class labels at ranks 1–10.

higher percentage of *vital* labels for Wiki_Manual and Web_Manual with 55.90% and 60% respectively. The distribution is skewed in favor of *okay* for tables in the Web_Relation set due to combination of a larger number of rows in those tables coupled with missing information in KBs leading to a lack of higher agreement for *vital* class labels.

Figure 4.3 shows the fraction of relevant class labels at different ranks. The majority of the relevant labels appear in the top five positions. Approximately 56% of the columns have two or more relevant labels in the ground truth in the Web_Relation dataset, resulting in a higher percentage of relevant labels at ranks 2 and 3.

We also compute precision and recall at each k between 1 to 10. For precision, we assign a score of 1 for every *vital* class and 0.5 for every *okay* class identified by the framework. For recall, we assign 1 for every *vital* and *okay* class identified. This evaluation scheme is similar to the one used by Venetis *et al.* (2011). Figure 4.4 shows the annotation results for class labels at rank 1 across three different sets. A significant percentage of the

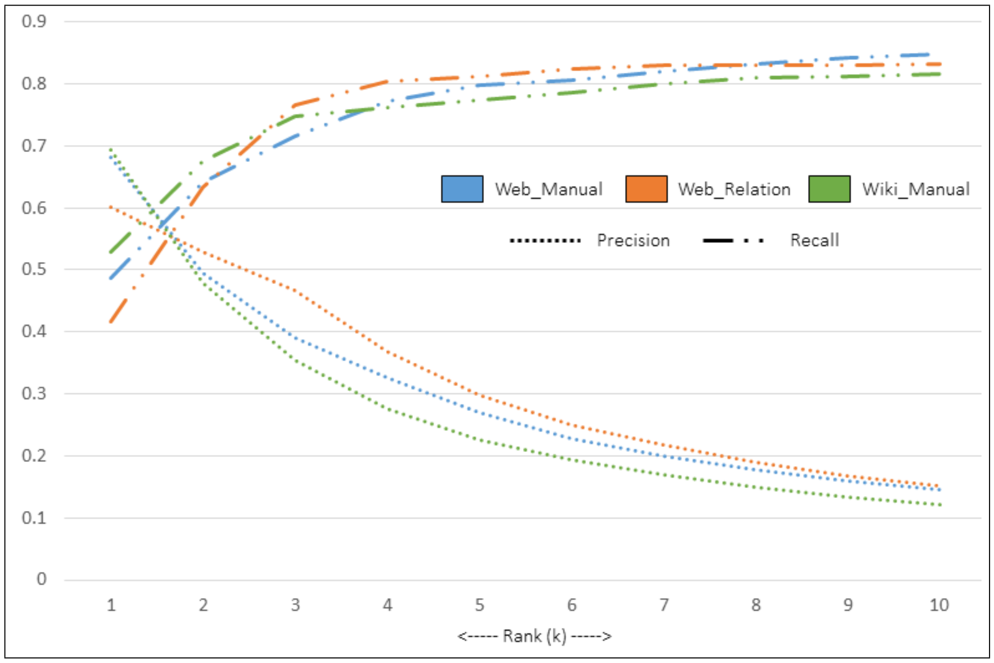


FIG. 4.5. Precision and recall for class labels at ranks 1–10.

columns had two or more expected relevant labels, which explains the lower recall and F–score at rank 1 for all the three sets. Figure 4.5 further confirms this. As expected, recall rapidly increases with *k*, stabilizing at k = 5, since most of the relevant labels appear between ranks 1 and 5.

A direct comparison between our framework and related work (Limaye, Sarawagi, & Chakrabarti 2010; Venetis *et al.* 2011) is not possible since we use a subset of the tables used in their evaluations and due to the difference in ground truth as pointed out in Section 4.1. Our F–score is measured over classes from the DBpedia ontology, while the others report over the Yago ontology. The idea behind the following comparison is to help readers get a perspective of our performance.

Our F–scores for *Web_Manual (0.57)* and *Wiki_Manual (0.60)* are better than the previously reported scores of 0.43 and 0.56 in (Limaye, Sarawagi, & Chakrabarti 2010). We fare slightly poorer for both datasets against scores of 0.65 and 0.67 as reported in (Venetis

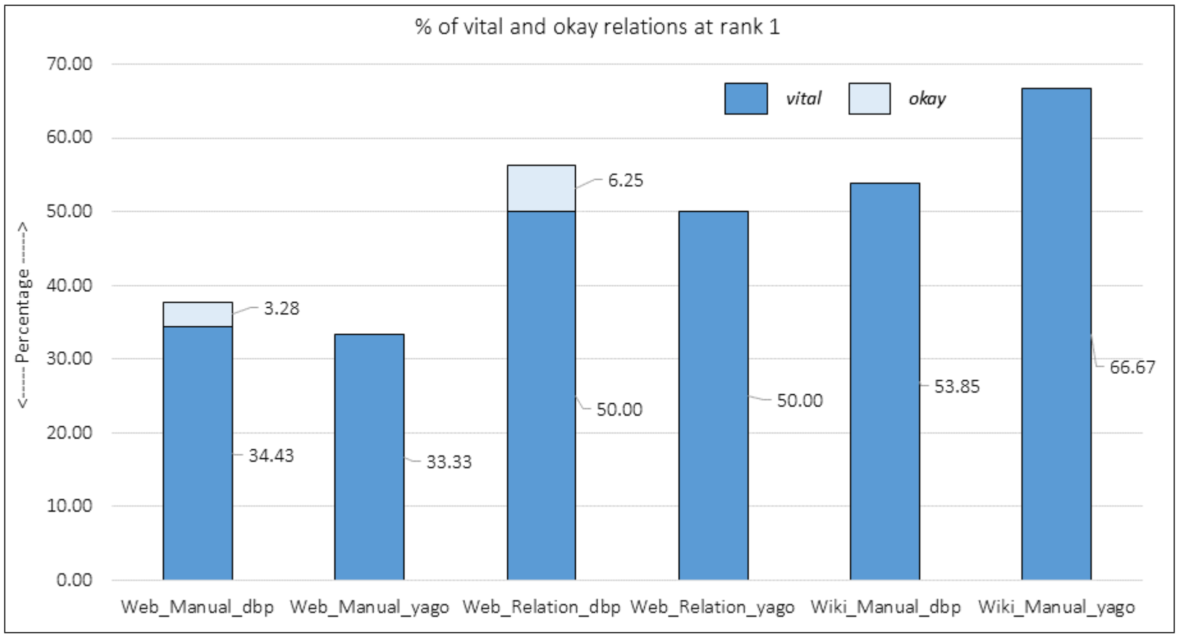


FIG. 4.6. Quality of relations at rank 1. Dataset names followed by a *_dbp* are results for DBpedia relations; whereas the ones followed by *_yago* are for Yago relations.

et al. 2011). The original ground annotations typically have one expected class for every column header leading to better recall at rank 1. Moreover, the system from Venetis *et al.* (2011) does not focus on entity linking. It predicts classes for column headers and relations between the "primary column" (e.g., a key) and other columns in the table independently. Our framework, in contrast, attempts to jointly map column header, cell values, relation between columns to appropriate assignments which in certain cases can lead to incorrect assignments.

4.4 Relation Annotations

TABEL also generates a set of top ten relation annotations for every pairs of column in a table with NO-ANNOTATION as the top assignment if it failed to find an appropriate relation. Our framework generated two separate sets of relations – one from the DBpedia ontology and another from Yago. We compared them against the DBpedia and Yago relations obtained from human evaluators.

The percentage of relevant relations (i.e., vital and okay) at rank 1 vary with both the dataset as well as the ontology (see Figure 4.6). More than 50% of relations at rank 1, both from DBpedia and Yago, for Web_Relation and Wiki_Manual are vital relations. *TABEL* performs poorly on the Web_Manual dataset with only 34% of vital labels at rank 1. A lack of agreement between the entities in column pairs is a major reason for these average results. For more than 64% of column pairs, for all the datasets and both the ontologies, *TABEL* predicted NO-ANNOTATION at rank 1. The lack of agreement can also be attributed to incorrect entity assignments to the data cell values. Entity linking evaluation shows better results for the the Wiki_Manual dataset as compared to the Web_Manual dataset. A similar trend is reflected in relation annotation evaluation with results for Wiki_Manual being better than Web_Manual. Finally, missing relations in KBs (which is more common than missing classes) also contributes to the average results.

Figure 4.7 shows the distribution of relevant relations at different ranks. The majority of them appear between ranks 1 and 3. While *TABEL* performs poorly in predicting a relevant relation at rank 1 for the Web_Manual dataset, it is compensated by a larger percentage of relevant relations appearing at ranks 2 and 3.

We also compute precision, recall and F–score as described in section 4.3. Figure 4.8 presents the results for relations at rank 1. Besides errors in predicting relevant labels at rank 1, lower recall and f–scores can be attributed to the larger size of the expected relation set. Human evaluators identified at least two relevant relations for most column pairs. Web_Manual_yago had the least such column pairs with 2.3% having two relevant relations, while Web_Manual_dbpedia had most column pairs with 24.2% having two or more relevant relations.The typical size of the expected relation set identified by human evaluators was smaller in comparison to the set of relevant class labels. Fewer column

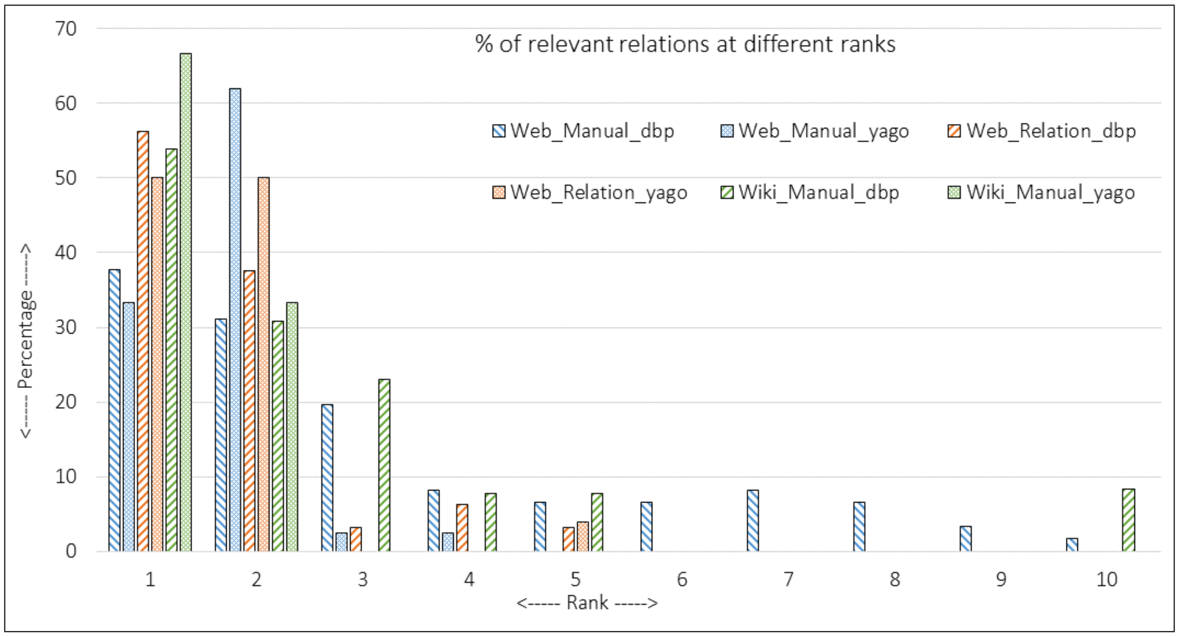


FIG. 4.7. Percentage of relevant relations at different ranks.

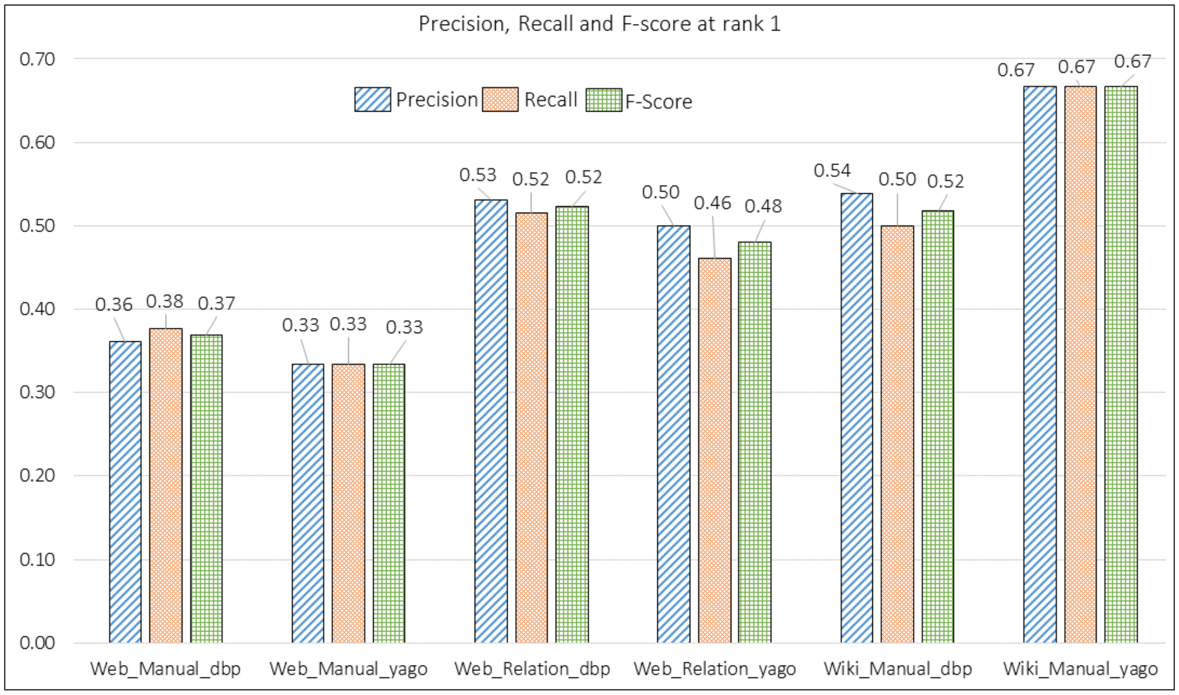


FIG. 4.8. Precision, Recall and F–score for relations at rank 1.

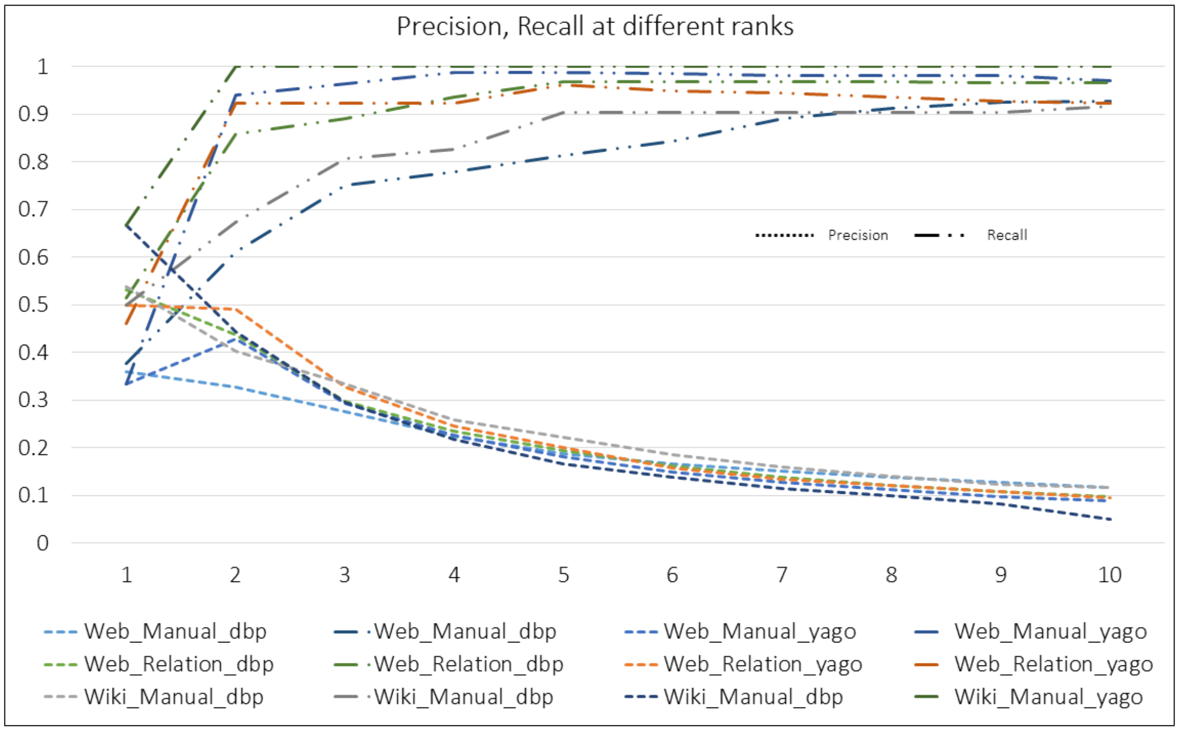


FIG. 4.9. Precision and Recall for relation annotation at different ranks.

pairs had more than two relevant relations in their expected set.

Figure 4.9 shows precision and recall at different ranks. Recall rapidly converges towards 1 (and conversely precision towards zero) for most datasets. Recall scores show two distinct trends in the graph – i) gradual convergence and ii) a rapid convergence towards one. Datasets with larger expected relevant relations converge gradually, recall scores steadying around ranks 4–6, while ones with a smaller set converge much quickly, scores steadying around ranks 2–3. As mentioned in section 4.3, direct comparison between our work and Limaye, Sarawagi, & Chakrabarti (2010) is not possible, but we report their scores to give perspective to our results. They report f–scores of 0.51 for Web_Manual, 0.63 for Web_Relation and 0.68 for Wiki_Manual for the relation annotation tasks. Relations in their ground truth belonged to the Yago ontology and the typical size of expected relation set was one.

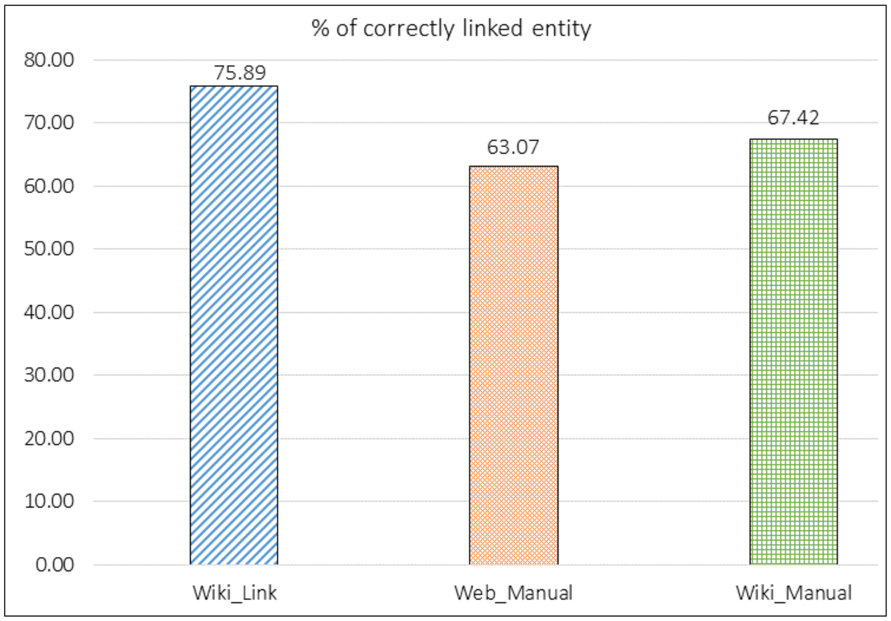


FIG. 4.10. Percentage of data cells with correctly linked entity.

4.5 Data Cell Annotations

We compared the entity links generated by our framework to those obtained as ground truth from the original dataset (Limaye, Sarawagi, & Chakrabarti 2010). Our framework linked a data cell value to an entity from DBpedia wherever possible, else to NO-ANNOTATION. If the predicted entity link matched with ground truth, we considered it as a correct prediction, else incorrect. We obtained an accuracy of 75.89% over the Wiki_Links dataset; 67.42% over the Wiki_Manual dataset and 63.07% over the Web_Manual dataset (Figure 4.10).

One of the reasons for lower accuracy is missing data in the KBs. Even if *TABEL* discovers the correct class and relation, the framework will fail to identify if an entity is correct if these annotations are missing for the entity. For example, even if we discover the Yago class *YagoGeoEntity* for a column of places, the entity *dbpedia:Berlin* does not have that class and thus can lead to an incorrect assignment. Lower accuracy also stems

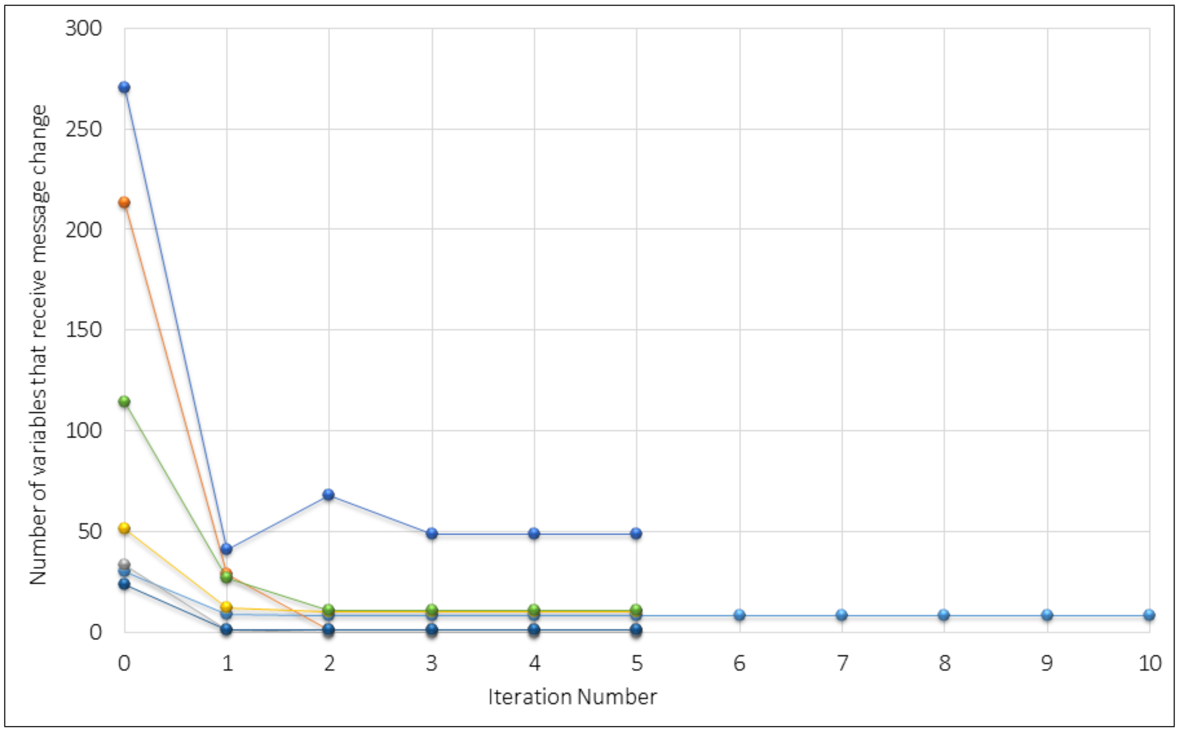


FIG. 4.11. X axis represents the iteration number and Y represents the number of variables that received CHANGE message in that iteration. The value at x=0 is the number of cells in the table.

from the size of the candidate entity set. We restricted the size of the candidate entity set to 25; thus it is possible that the correct assignment could be outside this set. We also note certain discrepancies in the ground truth annotations. We noticed several cases where we were able to discover a correct entity annotation, whereas the ground truth said it was NO-ANNOTATION, which led to counting our annotation as incorrect.

4.6 Convergence

Figure 4.11 gives insight into how quickly Semantic Message Passing (SMP) converges. Each line in the graph represents a table; every point on the line represents the number of variable nodes in the table that received a CHANGE message at the end of the

Class	*Precision*	*Recall*	*F-Score*
0	0.959	0.849	0.901
1	0.871	0.966	0.916

FIG. 4.12. Precision, recall and F–score for the Naive Bayes model.

given iteration. For most tables, the number of variable nodes that receive a CHANGE message stabilize after the first iteration. We had few cases where the variable count fluctuated, i.e., increasing and decreasing as iterations increased. We also noticed cases where the variable count does not go to zero. Some number of "stubborn" variables keep receiving a CHANGE message at the end of every iteration, but cannot find a new value. However, we noticed that the number of stubborn variables is less as compared to the original number of variables in the table. Recollect, we let SMP run for five iterations before our halting condition checks if it can be stopped. In most tables the inference stops after five iterations. We present results for eight tables for visual purposes; the results are representative of the rest of the tables in the dataset. The average time required for the inference model across all tables was 3.4 seconds.

4.7 Entity Ranker

The *entity ranker* uses a Naive Bayes classifier to produce likelihoods that strings should be linked to entities. We choose Naive Bayes because the features, the string similarity and popularity metrics, are fairly independent of each other. The training and test sets were generated using the ground truth for entity annotations from the Wiki_Links dataset. For every string mention in the table, we queried Wikitology to get candidate entities and then computed feature values for the string similarity and popularity metrics for each mention/entity pair.

A class label of 1 was assigned if the candidate entity was the correct assignment

numberOfDoctoralStudents	*numberOfGraduateStudents*	*numberOfPostgraduateStudents*
numberOfStudents	*numberOfUndergraduateStudents*	*populationMetro*
populationRural	*populationTotal*	*populationTotalRanking*
populationUrban	*totalPopulation*	*fuelCapacity*
capacity	*postalCode*	*seatingCapacity*

FIG. 4.13. List of DBpedia properties used to generate NumKB.

(available via ground truth in the dataset) and a 0 otherwise. The training set included 600 instances, evenly split between positive and negative instances. The test set included in all 681 instances with 331 positive and 350 negative instances. Out of the 681 instances, the model was able to correctly classify 619 instances with an accuracy of *90.9 %*. The precision, recall and F–score are presented in Figure 4.12.

4.8 Literal Constants

We evaluate the quality of header cell annotations for literal value columns. The properties used to represent literal constants are also used to represent the header string; i.e., we map the header string *Population* in Figure 1.2 to the property *dbpedia-owl:populationTotal*. We choose 16 tables, with 17 literal value columns from the Wiki_Link dataset. We generate a candidate property set for each column by querying against NumKB generated from the DBpedia properties listed in Figure 4.13. The rest of the experimental setup remains the same (as described in section 4.2).

We obtain ground truth by manually annotating each literal value column with an appropriate DBpedia property. Figure 4.14 shows the distribution of the expected properties at different ranks. Our approach shows promising results with a majority of the expected properties appearing at ranks 1, 2 and 3 (76.47%). The slightly lower % at rank 1 can be attributed to the ambiguous properties *totalPopulation* and *populationTotal*. When we treat either of the properties to be a correct prediction for a column of population values, the %

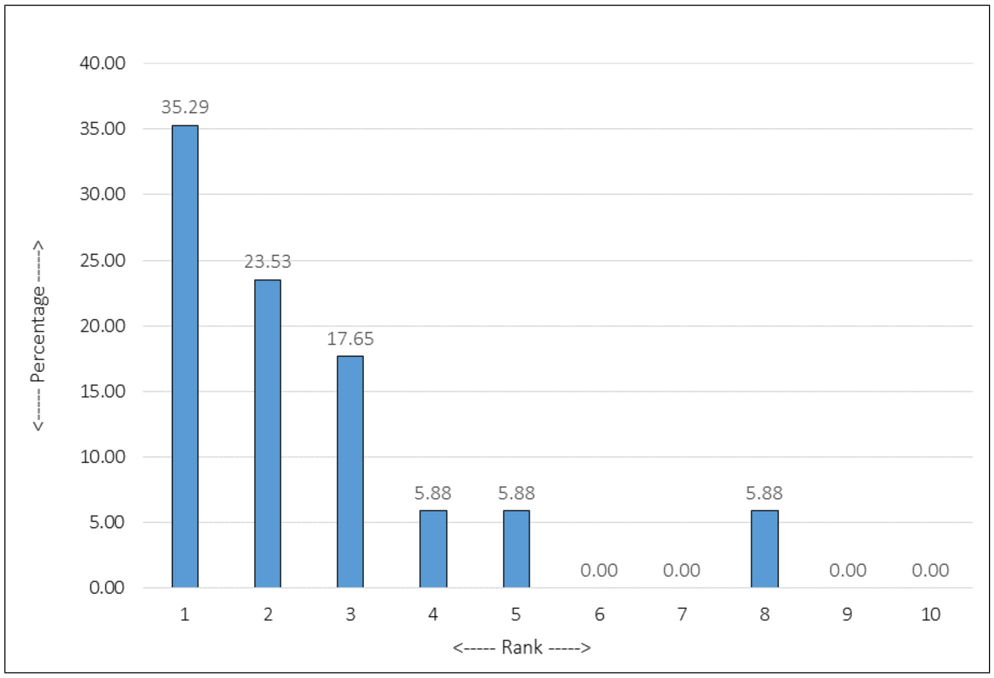


FIG. 4.14. Distribution of expected properties at different ranks. Majority of them appear at ranks 1, 2 and 3.

of correct properties at rank 1 jumps to **64.71**.

4.9 Human in the Loop

We evaluate and quantify the impact of user feedback on our framework by choosing 12 tables from the Wiki_Link dataset and providing class annotations for column headers. A single column was annotated with the correct DBpedia and Yago class for ten of these tables, while two columns were annotated in the remaining two tables. While this feedback was provided before the inference process starts, the same could have been provided during inference. Using these annotations, our framework inferred the rest of the semantics with the modified behavior of Semantic Message Passing as defined in section 3.5.

Column header annotations have a direct impact on the quality of data cell value annotations. Our results show that small (1–2 classes) but accurate feedback can significantly

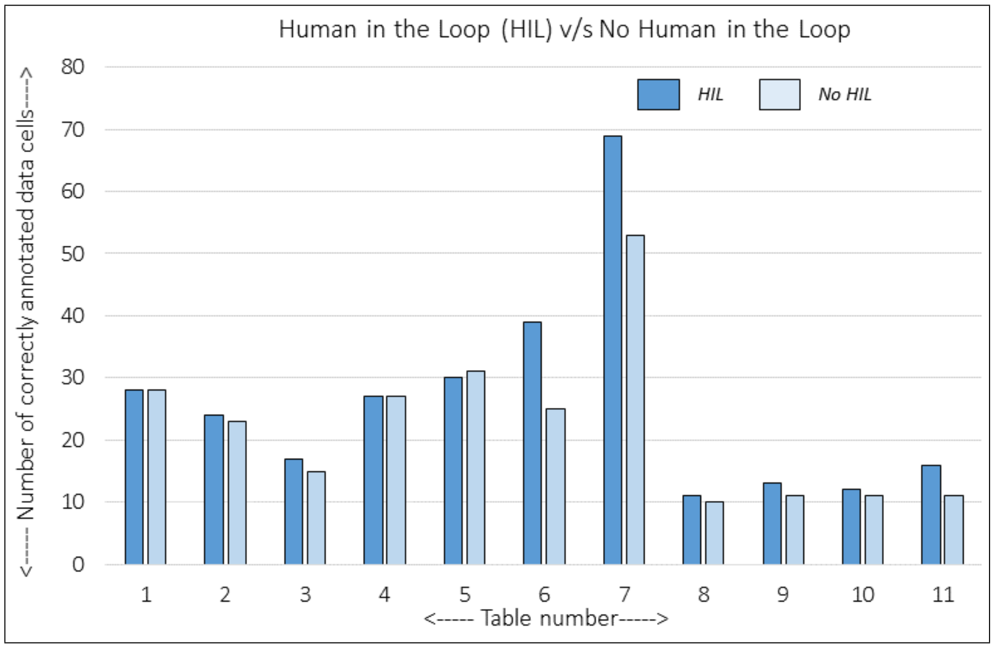


FIG. 4.15. Comparison between data cell annotation quality with and without user feedback.

improve annotation quality. Data cell value annotations improved by **10.2%** from 60.94% to 71.14% with the help of user class annotations. Figure 4.15 shows the impact of annotations on the individual tables. The number of correctly annotated data cells increased in 9 tables, remained the same in 2, and, decreased in 1.

4.10 Comparison with baseline systems

We evaluate the effectiveness of the Query and Rank (Q&R) module and Semantic Message Passing (SMP) by replacing them with naive baseline alternatives. We create a naive alternative for the Query and Rank module, by modifying the Wikitology query for generating candidate entities for data cells. The data cell value string is mapped to the title field in Wikitology. The context provided by the column header string and rest of the row values is not used. The candidate generation process for column header classes and

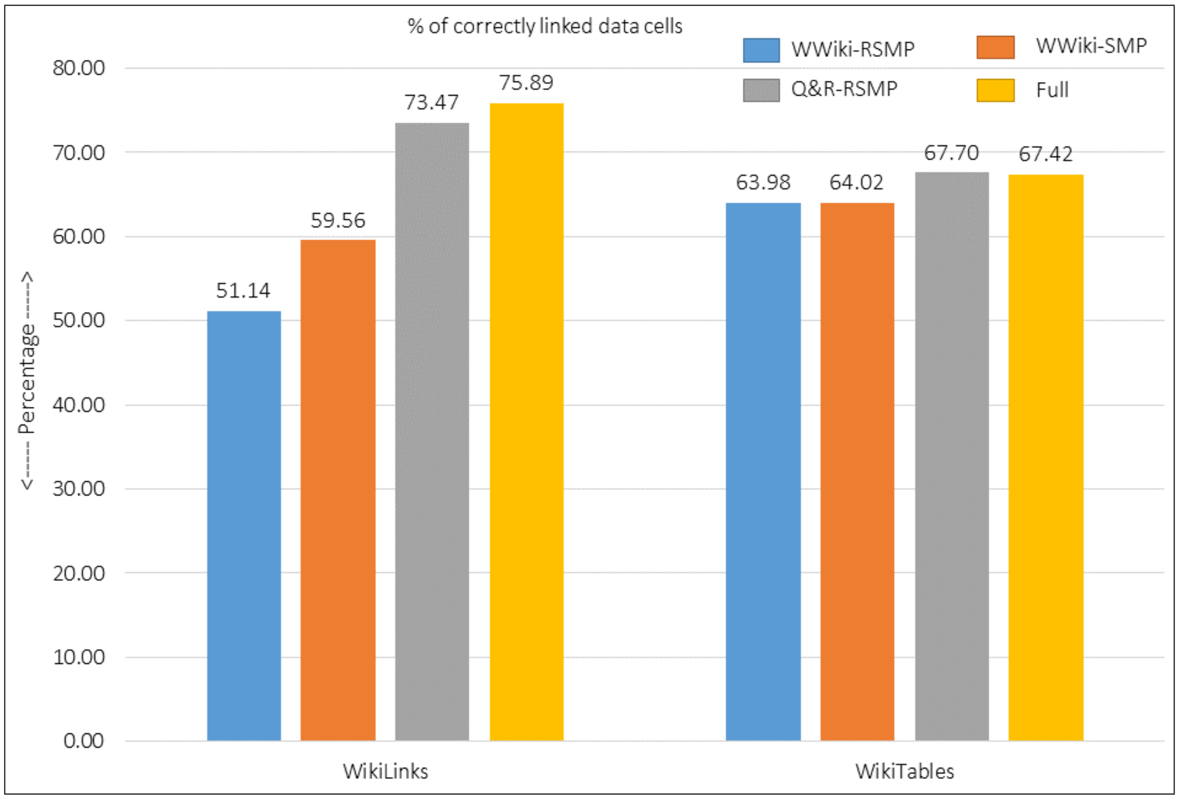


FIG. 4.16. Quality of data cell annotations for all four configurations.

relations between columns remains the same. We refer to this alternative as *WWiki*.

A naive alternative for Semantic Message Passing (*RSMP*) is created as follows: The factor nodes operate as described in section 3.3.3; however in the modified RSMP, they do not send semantics along with the CHANGE message. Variable nodes update their assignments if they receive at least one CHANGE message. With the lack of semantics, variable nodes pick the next highest ranked entity as a new assignment. The rest of process remains the same.

We create three different baselines with configurations WWiki–RSMP, WWiki–SMP and Q&R–RSMP and compare them against our default system which uses Q&R and SMP. Figure 4.16 presents the quality of data cell annotations for all four configurations over the Wiki_Links and Wiki_Manual datasets. A naive Q&R module along with a naive SMP leads to the lowest accuracy for data cell annotations. The strength of our Q&R module is demonstrated by the Q&R–RSMP configuration. Our Q&R module with a naive SMP still leads to good data cell annotations. While SMP individually improves the annotation quality, it still depends on the candidate entities, classes and relations. A naive Q&R leads to poor candidate sets and thus impacts the inference process and annotation quality.

Chapter 5

USE CASE

We demonstrate the extensibility and domain independence of our techniques by developing an application of *TABEL* in the healthcare domain. We first introduce the problem and our application, describe how *TABEL* was extended and adapted for tables from the medical domain and finally evaluate the quality of the inferred semantics as well as its effectiveness for the developed application.

5.1 Introduction

Evidence–based medicine (EBM) is commonly defined as "the conscientious, explicit, and judicious use of current best evidence in making decisions about the care of individual patients" (Sackett *et al.* 1996). EBM analyzes questions such as efficacy of drug dosages, correlation between various medical factors or correlation between drugs by performing meta–analyses (i.e., systematic reviews) over evidence and data previously published in scientific literature and clinical trial studies. The goal is to find, integrate and analyze available high-quality quantitative data to inform clinical and health care related decisions.

EBM has been gaining traction for the past several years. A search on PubMed[1] for publication type "Meta–Analysis" shows, while only 272 meta–analyses reports were

[1]http://www.ncbi.nlm.nih.gov/pubmed

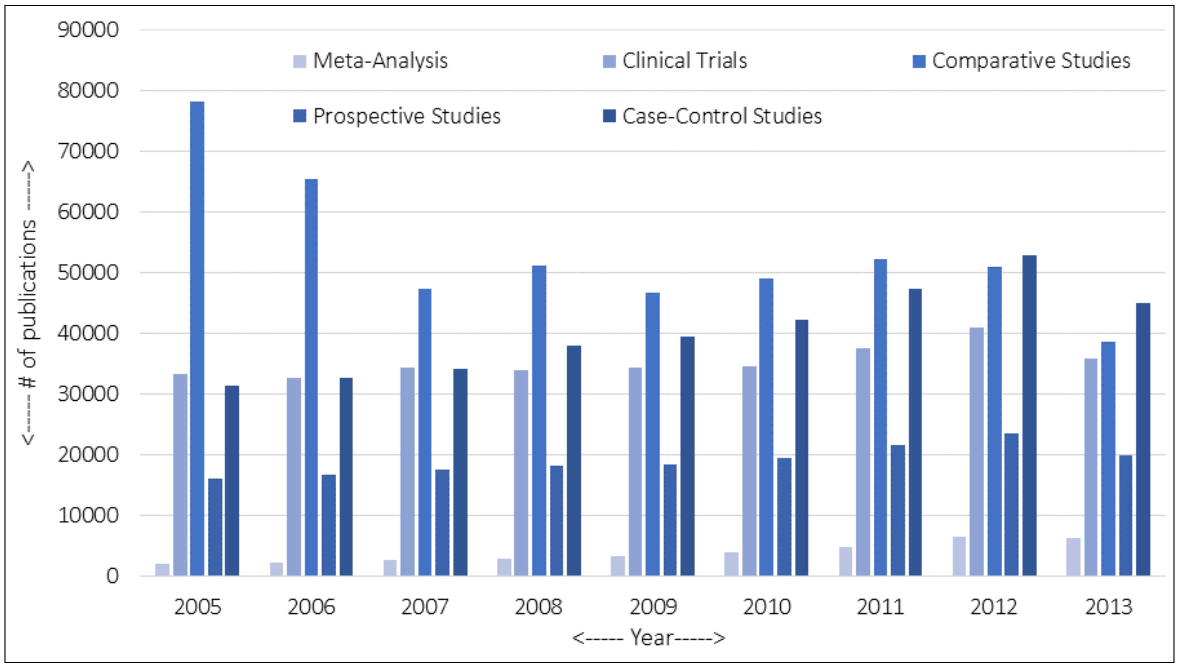


FIG. 5.1. Number of meta–analysis, clinical trials, comparative studies, prospective studies and case–control studies published between 2005 and 2013. Data was obtained from PubMed by querying for publication types "Meta–Analysis","Clinical Trial" "Comparative Studies", "Prospective studies", and "Case-Control Studies"

published in 1990, more than 6600 meta–analyses reports were published in 2013. Organizations such as The Cochrane Collaboration[2] have a dedicated set of medical researchers whose primary goal is to perform and publish systematic reviews on a number of health care related issues and to keep them updated as new medical research findings become available.

The process of generating a meta–analysis report is still largely manual. Medical researchers start with keyword search on systems like MEDLINE[3] which often lead to thousands of initial set of studies. Researchers carefully analyze each study reducing the result set to a few hundred studies or less which are included in the final meta–analysis.

[2]http://www.cochrane.org/

[3]http://nlm.nih.gov/bsd/pmresources.html

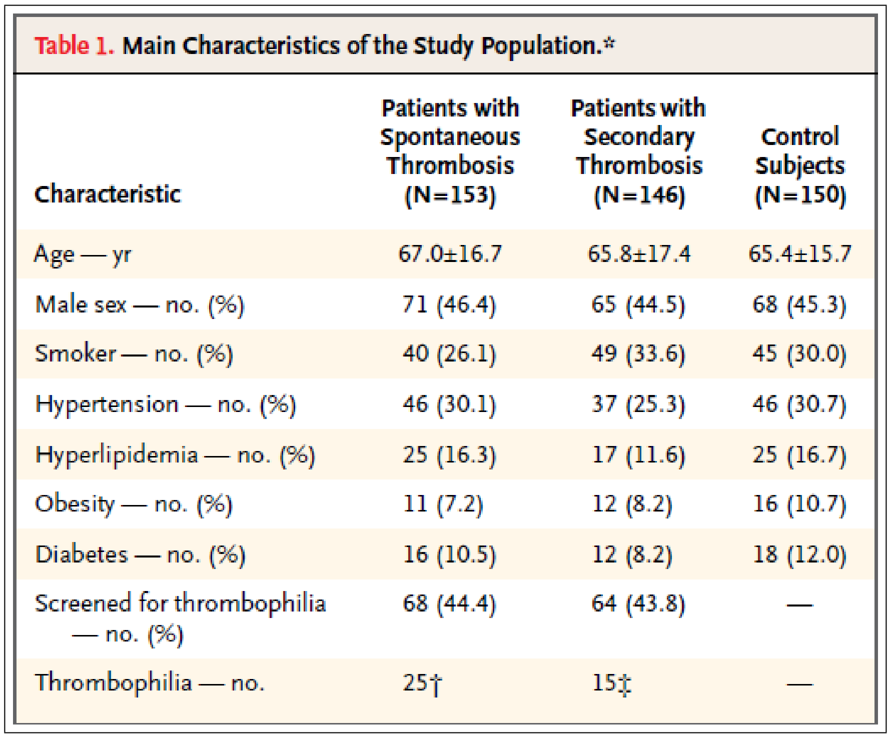


Table 1. Main Characteristics of the Study Population.*

Characteristic	Patients with Spontaneous Thrombosis (N=153)	Patients with Secondary Thrombosis (N=146)	Control Subjects (N=150)
Age — yr	67.0±16.7	65.8±17.4	65.4±15.7
Male sex — no. (%)	71 (46.4)	65 (44.5)	68 (45.3)
Smoker — no. (%)	40 (26.1)	49 (33.6)	45 (30.0)
Hypertension — no. (%)	46 (30.1)	37 (25.3)	46 (30.7)
Hyperlipidemia — no. (%)	25 (16.3)	17 (11.6)	25 (16.7)
Obesity — no. (%)	11 (7.2)	12 (8.2)	16 (10.7)
Diabetes — no. (%)	16 (10.5)	12 (8.2)	18 (12.0)
Screened for thrombophilia — no. (%)	68 (44.4)	64 (43.8)	—
Thrombophilia — no.	25†	15‡	—

FIG. 5.2. An example table typically found in medical research reports.

Often a two stage filtering is done in which studies are accepted or rejected first based on the title and abstract and then later filtered after a close examination. A meta–analysis of the correlation between *cardiovascular risk factors* and *venous thromboembolism* (Ageno *et al.* 2008), for example, started with an initial search yielding 1949 studies which were downselected to just 22 for the final analysis and report. Figure 5.1 gives insight into how tedious the process is. While the number of meta-analysis reports published each year is growing, they are out-paced by the number of clinical trials, comparative studies, prospective studies and case–control studies that potentially provide evidence.

Key information required to produce meta–analyses reports is often obtained from tables like the one shown in Figure 5.2. Consider an analysis of the correlation between *Obesity*, a cardiovascular risk factor and *venous thromboembolism*. Conclusions about the correlation are derived by first identifying relevant studies such as (Prandoni *et al.* 2003;

Paganin *et al.* 2003) and then by extracting and integrating results from these individual studies. In this example, information such as number of individuals that suffer from obesity and venous thromboembolism (23/299) and number of individuals that suffer only from obesity (16/150) is of key interest. Such information, as seen from Figure 5.2, is encoded in tables published in medical studies.

The automatic discovery and interpretation of tables in a medical research report gives strong evidence that (i) the report includes empirical data and (ii) the degree of relevance to the question. The tables themselves provide the raw study data that will eventually be integrated. Inferring the semantics of tables and producing a linked data representation delivers the data in a form that facilitates aggregation, mapping and integration.

5.2 Related Work

This section specifically focuses on related research in the space of evidence–based medicine and the use of ontologies to model clinical trials and other medical research studies.

Recent research has focused on taking steps towards automating evidence–based medicine and generating meta–analysis reports. Cohen et al. (Cohen *et al.* 2010) present a design for end-to-end text-mining pipeline to automate the process of generating and updating meta–analysis reports. Their pipeline consists of searching, classifying, grouping and ranking medical research papers to produce systematic reviews. ExaCT (Kiritchenko *et al.* 2010) is a information extraction system that searches and extracts sentences from clinical trials and other related studies that best match the clinical trial characteristics provided by the user. It aims to facilitate in the process of identifying relevant studies for producing evidence reports.

Researchers have produced systems to create summaries of medical papers (Sarker, Mollá, & Paris 2013) and also from medical paper abstracts (Summerscales *et al.* 2011) to be used in meta–analysis studies. Others have applied machine learning techniques to reduce the number of search query results a medical researcher must analyze to collect relevant studies (Kilicoglu *et al.* 2009; Cohen, Ambert, & McDonagh 2009). However, this entire body of work has focused on analyzing free text; to the best of our knowledge no work has focused on analyzing and inferring information encoded in tables in medical research literature. Our approach can not only find relevant studies, but can also extract and integrate the data to produce meta–analysis reports.

Related work has also focused on using ontologies and other Semantic Web technologies in assisting clinical trial management. Frameworks such as the ObTiMA System (Stenzhorn *et al.* 2010) and Epoch (Shankar *et al.* 2006) allow management of ongoing clinical trial by providing tools to researchers to capture data and represent data as RDF. Both provide ontologies useful for representing data of ongoing clinical trials. ORCe (Sim *et al.* 2013) is a general purpose ontology allowing users to model various aspects related to clinical trials such as study design, study protocol, statistical concepts related to the study analysis. ADDIS (Van Valkenhoef *et al.* 2013) presents an ontology to ground various clinical trials in a common data schema, facilitating search and integration. ADDIS further provides users a semi-automatic decision support software that allows importing studies, representing them in RDF and producing evidence reports. LinkedCT (Hassanzadeh *et al.* 2009) triplifies data sources such as ClinicalTrials.gov; Dameron et al. (Dameron *et al.* 2012) designed an ontology to model and reason about patient eligibility in clinical trials. While existing work has covered the breadth in clinical trial management using Semantic Web technologies, our Medical Tables ontology (section 5.3) focuses on a very specific task – modeling and representing medical tables published in medical research papers.

5.3 Approach

5.3.1 Inferring the semantics of medical tables

We adapt *TABEL* to infer the semantics and generate a linked data representation of medical tables. A table first goes through a pre-processing module *Normalize*, which handles the idiomatic patterns typically found in medical tables. The *Query and Rank* (Q&R) module generates candidate concepts for the header cells as described in section 3.2. We also adapt Q&R to generate candidates from domain specific KBs, namely SNOMED CT and UMLS. Header cells are mapped to the top ranked candidate returned by Q&R before a final linked data representation is produced.

Normalization. Medical tables do not have the simple structure of most tables found on the web (webtables), i.e., a rectangular array of data cells (with optional column headers) where each cell holds a single value. Medical tables often exhibit of both column and row headers, between whom the data is enclosed. Non-header cells are *data cells* that typically represent values for the relationship between the respective column and row headers. For example, the value *46* from the column of *Patients With Spontaneous Thrombosis* and row header *Hypertension* in Figure 5.2 leads to the interpretation that 46 of the 153 patients with *spontaneous thrombosis* also suffer from *hypertension*.

Typically, content in header and data cells in webtables is simple in nature; it either consists of strings that can be directly mapped to a class or a entity or literal values such as numbers that can be mapped as values of a property. However, the content found in header and data cells in medical tables are represented using idiomatic patterns often encoding additional metadata in header cells or representing the literals in data cells as “complex objects”.

Consider the column header *Patients With Spontaneous Thrombosis (N=153)*; not only it represents a concept (*Spontaneous Thrombosis*), but also encodes additional metadata *(N=153)* which is useful for interpreting the values in the column. Similar encoding can be observed in row headers. Data cells are also complex objects. Consider the data cell *46 (30.1)* from the row with header *Hypertension* in Figure 5.2; from the metadata in its row header, one can interpret, *46* represents the raw number and *30* its percentage of the total. Interpreting medical tables thus requires significant pre–processing in which the content in the header cells have to be normalized to extract strings to be mapped to concepts and decompose and interpret data cells to generate an accurate semantic representation.

The *Normalize* module first processes the header cells. The content in column and row header cell can be parsed into two distinct parts: a *query string* denoting a concept such as a disease, drug, or patient characteristic and *metadata* that describes how the data in the column/row should be interpreted, e.g., giving units of measure (*kg*), statistical properties (*avg*) or a schema for non-atomic values (*no. (%)*). For example, the fourth row header *Hypertension – no. (%)* would produce the query string *Hypertension* and metadata *no. (%)*.

We develop a set of regular expression patterns that cover the most common cases observed in published medical data tables to extract query string and metadata. In most medical tables, the query string and metadata are either separated by a hyphen [query string - metadata], a comma [query string , metadata] or the metadata follows the query string in parenthesis [query string (metadata)]. Every column and row header is matched against these three patterns to extract query string and metadata. If the header content does not match any of the patterns, the content is treated as query string. In our current example, the header *Hypertension – no. (%)* will match with [query string - metadata] extracting *Hypertension* as query string and *no. (%)* as metadata.

The extracted metadata can require further processing as it encodes information regarding how values in the respective column or row should be interpreted. For example, the metadata *no. (%)* conveys the message that values of the form *a (b)* in the given row should be interpreted as *a* representing the raw number and *b* representing its percentage of the total. While metadata in medical tables can encode vast variety of information, they can be generalized to a limited set of common patterns such as generalizing *no. (%)* to *a (b)*. The Normalize module further processes metadata by mapping it to a pattern from a set of generalized metadata patterns.

We identify a set of common metadata patterns which include: *a (b) (c)* (e.g. mean (standard deviation) (range)); *a (b)* (e.g. no. (%)); *a +/-b* (e.g. mean standard +/- deviation); *a/b* (e.g. Male/Female). Every metadata extracted from the content in header cells is mapped against one of four metadata patterns. Thus, the extracted metadata from the *Hypertension* row header, *no. (%)* is normalized as *a (b)*. The generalized metadata pattern is further used to interpret the values in the respective columns and rows. Header cells representing patient groups used in the study, encode number of the patients in the group as part of the header metadata. The Normalize module uses an additional rule $n = x$ pattern, where x represents the number of patients in respective patient group.

The content in data cells is processed by the Normalize module by using the generalized header cell metadata patterns. The data cell content is also mapped against one of the four metadata patterns. For example, the data cell content *46 (30.1)* will get mapped to the pattern *a (b)*. Once the data cell is mapped, the Normalize module checks for the same pattern in the respective column or row header. If the same pattern is discovered, it is used to decompose the data cell content. Again, in the case of *46 (30.1)*, its pattern *a (b)* will match with its row header pattern *a (b)* , which is used by the Normalize module to decompose and interpret *46* as *no.* and *30.1* as *%*.

We also implement the Acronym and Abbreviation Detector (AAD) as described in section 3.1. For every query string, AAD checks for a match in it's dictionary. If a match is found, the query string is replaced with its expanded version.

Query and Rank. The Query and Rank (Q&R) module generates a ranked list of candidate classes for every query string in row and column headers. We assess three different knowledge bases (KBs), two domain specific ones, UMLS Metathesaurus (Schuyler *et al.* 1993) and SNOMED CT (Stearns *et al.* 2001), and one general purpose one, DBpedia (Bizer *et al.* 2009). KBs in the health care domain are still maturing and our goal behind assessing different KBs is to compare the coverage and strengths of each in the context of medical tables.

A key challenge during Q&R in the case of medical tables is *context*. Consider the data cell *Baltimore* from the table in Figure 1.2. Rich context is available in the form of column header string *City* and rest of the values in the row *MD, S.Dixon* and *680,000* to disambiguate *Baltimore*. Similar context is not available in medical tables since the values in row and column headers are generally independent of each other and are not useful as context to disambiguate during querying. While disambiguation can be a challenge, vocabulary used in the medical domain is fairly controlled often leading to a exact match if the terms used in medical tables and terms in a knowledge base or an ontology are the same.

SNOMED CT (Systematized Nomenclature of Medicine–Clinical Terms) is a clinical terminology consisting of more than 311,000 medical concepts organized in a hierarchy with more than 1,360,000 links or semantic relationships between them. For every column and row header query string, the Q&R module first executes an exact match query against all concepts and their synonyms in SNOMED CT. If no results are returned, Q&R executes a free text search query over an index of SNOMED CT concepts. The order in which the concepts are returned, both for exact match and free text search queries, is retained by Q&R to rank the concepts. For example, the query string *Age* results in a direct match with two concepts *Age (qualifier value)* and *Current chronological age (observable entity)* as results;

whereas the query string *Diabetes* leads to a text search query with *Diabetic cataract associated with type I diabetes mellitus*, *Diabetic oculopathy associated with type I diabetes mellitus*, *Diabetic retinopathy associated with type I diabetes mellitus*, etc. as results.

UMLS Metathesaurus (Unified Medical Language System Metathesaurus) has of large number of concepts related to clinical, health care, and biomedical domains assembled by combining information from over 150 different clinical, biomedical, health care related terminologies, vocabularies and code–sets, including SNOMED CT. Like SNOMED CT, concepts within the UMLS Metathesaurus are connected via a set of relationships. We use the UMLS API *findConcept* service to generate candidate concepts. For every query string, Q&R module queries the API which returns a ranked list of concepts matching the query string. For example, for the input query *Hypertension*, the API returns concepts such as *Hypertensive disease*, *Hypertension Adverse Event*, *No hypertension*, *Hypertension with complications*, etc. Q&R again retains the ranking order.

We also implement a *Hybrid* query model combining UMLS Metathesaurus and SNOMED CT. Q&R first queries against UMLS Metathesaurus; if no results are returned it further queries against SNOMED CT as described above. The UMLS api fails to return results for certain query strings and *Hybrid* allows us to use SNOMED CT as a fall back option for those cases.

Finally, we use *DBpedia* as a KB to generate candidates for header cells in medical tables. Unlike SNOMED CT and UMLS Metathesaurus, the row and column headers from medical tables can be mapped to DBpedia instances rather than DBpedia classes. Similar to the original strategy in section 3.2, Q&R generates a ranked list of candidate instances by querying against Wikitology which is adapted to account for lack of context.

Q&R matches the query string against Wikipedia article's title, redirects, first sentence and contents(body). The query for *Hypertension*, for example, returns ranked list of instances *Idiopathic_intracranial_hypertension*, *Pulmonary_hypertension* and *Hypertension*. Q&R module reranks the results returned by Wikitology using the *entity ranker* described in section 3.2.

Joint Inference. The dependencies used to construct the graphical model in section 3.3 are less common in medical tables. Row headers are fairly independent of each other leaving little room to capture any correlation. Column headers in medical tables consists of patient groups and statistical tests comparing the groups. Thus, if a header cell in a medical table gets mapped to a statistical test, then it is likely that other headers are various patient groups. This correlation can be captured by column header factor node ψ_5. Row headers, in certain cases, can be disambiguated with the help of values in the row. Consider the values in the row *Age*: *67*, *65.8*, and *65.4*. One can infer from the range of the values that they represent age of a person. This is analogous to the correlation captured via the factor node ψ_3. However, such cases are less frequent and the problem of inferring and mapping a set of numerical values to a concept remains an open problem. Thus, the joint inference module is skipped and the top ranked concept returned by Q&R is considered as the final assignment.

RDF Generation. Once data cells are normalized and interpreted, and the column and row headers are mapped to appropriate concepts, the *Generate RDF* module generates a RDF linked data representation of the table. Unlike webtables, medical tables have a complex representation with the data cell values representing the relationship between the row and column headers. We develop the *MTO* ontology (Medical Table Ontology 2014) (Figure 5.3) to model and represent information encoded in such medical tables. MTO can be used

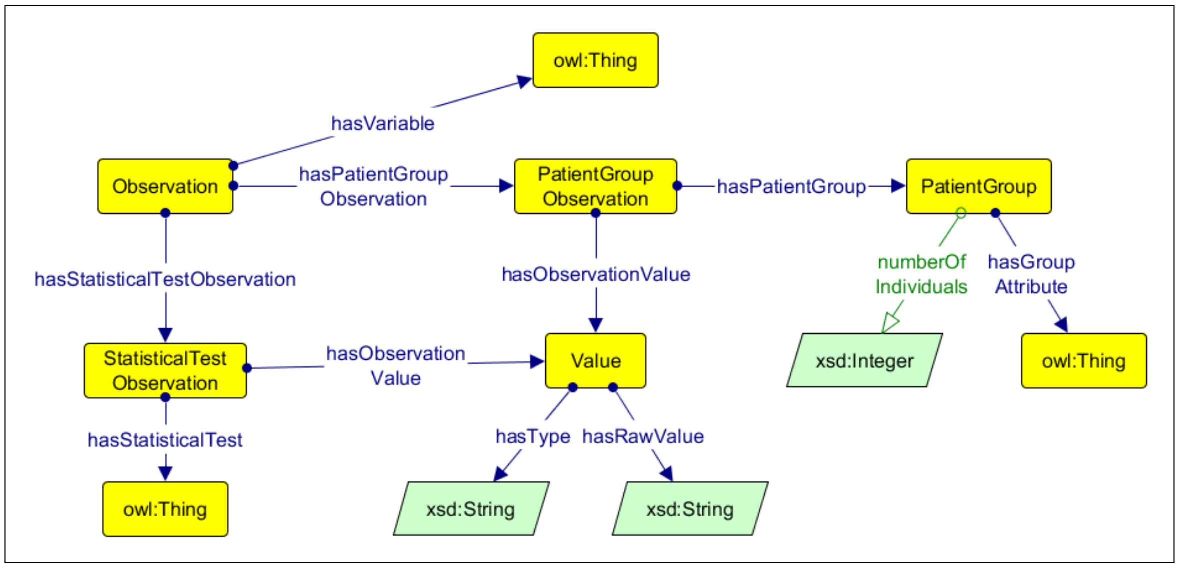


FIG. 5.3. Core of the Medical Tables Ontology. The prefix ‘xsd’ is for XML schema and ‘owl’ is for the OWL Ontology.

in conjunction with the *Tables* ontology or can be used independently. We describe details of MTO followed by its usage together with Tables and its independent usage.

A typical medical table represents information about a set of observations related to the question analyzed in the study. A study of the correlation between *Atherosclerosis* and *Venous Thrombosis* will include medical tables reporting on observations that helps one analyze if the correlation exists or not. Each row in such a table can be interpreted as representing one observation. We add a class **Observation** to represent every row in a medical table. Every observation is typically associated with different aspects of the study such as patient groups, characteristics associated with patients in such groups and statistical analysis tests comparing the groups.

The characteristics associated with patients in the group can vary from information about demographics and habits such as gender, age and smoking habits to information about the diseases patients may or may not be suffering to data on vitals such as blood pressure, sugar level and hemoglobin. We combine all of these into a single group and refer to it as *Variables* associated with the observation. Typically, such variables appear in the row headers of the table. The association between an observation and its variable is captured by the **hasVariable** property.

Patient groups refer to group of individuals with different characteristics used for comparisons in the study. For example, a study comparing the correlation between *venous thrombosis* and *obesity* will consist of group of individuals that suffer from both the diseases and a control group which includes individuals that suffer only from *obesity*. Another study might include patient groups, each on a different drug dosage to study the effect of dosages on a particular disease. The relation between a observation and a patient group is captured via the **hasPatientGroupObservation** property.

Finally, the table often reports statistical analysis performed to compare different groups using tests such as Odds ratio, Hazard Ratio, and p-value. The correlation between an observation and a statistical test is captured via the **hasStatisticalTestObservation** property.

Instances of the class **StatisticalTestObservation** capture test information with two key components: the type of statistical test (e.g., odds ratio, hazard ratio) via the property **hasStatisticalTest** and an associated result or value via the **hasObservationValue** property. The domain of the property hasObservationValue is StatisticalTestObservation or PatientGroupObservation and its range is the class **Value**, whose instances are used to capture the normalized and decomposed content from data cells. The property **hasRawValue** captures the actual value from the table cell and **hasType** captures its interpretation, with both having domain Value and range **string**.

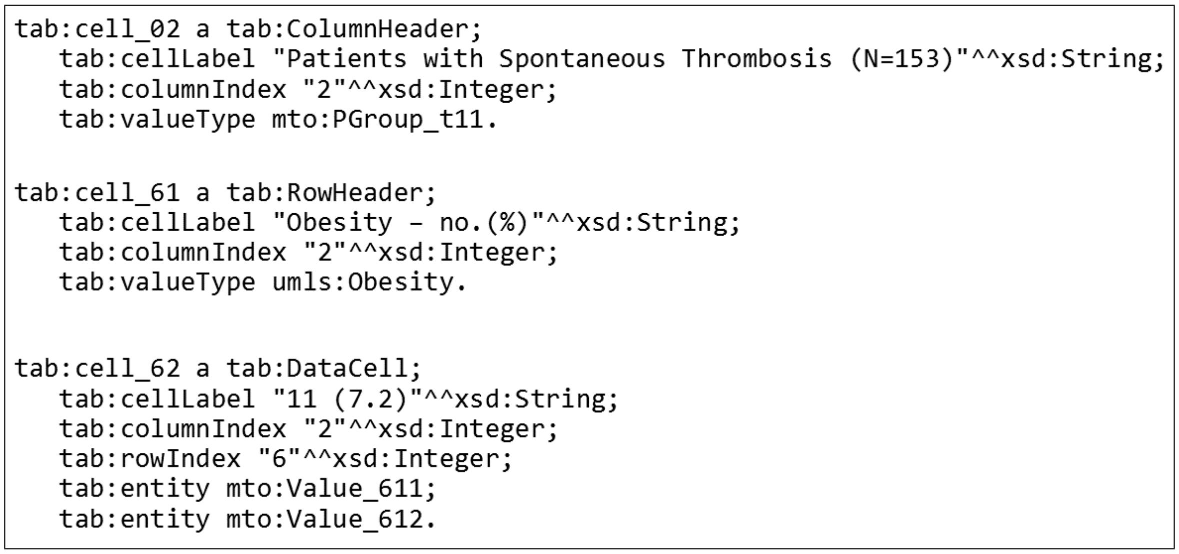
Similarly, instances of the class **PatientGroupObservation** are used to capture observation value associated with the patient group. We define a property **hasPatientGroup** to associate patient group observation with a patient group; and use previously hasObservationValue to associate the actual observation value. The domain of hasPatientGroup is the

```
# Triples representing the sixth row related to Obesity
mto:Observation_6 mto:hasVariable umls:Obesity;
   mto:hasPatientGroupObservation mto:PGroupObs_61;
   mto:hasPatientGroupObservation mto:PGroupObs_62;
   mto:hasPatientGroupObservation mto:PGroupObs_63.
mto:PGroupObs_61 mto:hasPatientGroup mto:PGroup_t11;
   mto:hasObservationValue mto:Value_611;
   mto:hasObservationValue mto:Value_612.

# Triples representing the first patient group (second column header)
mto:PGroup_t11 mto:numberOfIndividuals "153"^^xsd:integer;
   mto:hasGroupAttribute umls:Venous_Thrombosis.


# Triples representing the value in first cell of Obesity row
mto:Value_611 mto:hasRawValue "11"^^xsd:string;
   mto:Type "no."^^xsd:string.
mto:Value_612 mto:hasRawValue "7.2"^^xsd:string;
   mto:Type "%."^^xsd:string.
```

FIG. 5.4. Subset of RDF triples for data in Figure 5.2. The prefixes 'mto' is for our Medical Tables Ontology; 'umls' is for UMLS Metathesaurus and 'xsd' is for XML schema.

class PatientGroupObservation and its range is the class **PatientGroup**. The class PatientGroup is used to capture information related to the group. Depending upon the study, this information can vary from a disease from which individuals suffer (e.g., *Venous Thrombosis*) to drugs that individuals are taking. We define the property **hasGroupAttribute** to capture this information. The domain of this property is the class **PatientGroup**. Additional metadata related the group such as number of individuals in the group is also captured via the instances of class PatientGroup. We define the property **numberOfIndividuals** to capture this metadata, with domain PatientGroup and range **xsd:integer**.

Observation instances are associated with a **Table** through the property **tableObservation**. **Tables** are linked to a study using the property **hasTable**. We also record additional metadata associated with the study such as name of the study, publication information associated with the study such as authors, date of publication and journal. We use the properties



```
tab:cell_02 a tab:ColumnHeader;
   tab:cellLabel "Patients with Spontaneous Thrombosis (N=153)"^^xsd:String;
   tab:columnIndex "2"^^xsd:Integer;
   tab:valueType mto:PGroup_t11.

tab:cell_61 a tab:RowHeader;
   tab:cellLabel "Obesity – no.(%)"^^xsd:String;
   tab:columnIndex "2"^^xsd:Integer;
   tab:valueType umls:Obesity.


tab:cell_62 a tab:DataCell;
   tab:cellLabel "11 (7.2)"^^xsd:String;
   tab:columnIndex "2"^^xsd:Integer;
   tab:rowIndex "6"^^xsd:Integer;
   tab:entity mto:Value_611;
   tab:entity mto:Value_612.
```

FIG. 5.5. Subset of RDF triples for data in Figure 5.2 using the Tables and MTO ontologies.

studyTitle, **studyAuthor**, **publicationDate** and **publicationJournal**.

The *Generate RDF* module uses the MTO ontology to represent inferred semantics of medical tables as RDF linked data. A subset of RDF linked data generated from the table in Figure 5.2 is shown in Figure 5.4. Every row in the table is represented as instance of the *Observation* class. Linking the row header string *Obesity* to *umls:Obesity* provides additional information that the header type is disease. Concepts such as diseases are often characteristics or attributes of a patient group and are linked with a observation using the *hasVariable* property. The identified patient groups in column headers are linked with the observation using the *hasPatientGroup* property. Instances of the *PatientGroup* represent additional information about the group. The data cells are linked with the observation using *hasPatientGroupObservation* property. Every data cell is linked to the observation using instances of the *PatientGroupObservation*. The normalized data cells are represented using instances of the *Value* class and linked to patient group observations using *hasObservation-Value* property.

Figure 5.5 represents the same data using both the MTO and Tables ontology. While the Tables ontology captures the overall structure as well as semantics of the table, the PatientGroup and Value classes from MTO are helpful in capturing fine grained semantics of ‘complex’ header and data cells encountered in medical tables.

5.3.2 Find – Extract – Integrate

Once medical tables are represented as RDF linked data, we can execute SPARQL queries to automate the process of discovery, extraction and integration of data from multiple studies to automatically produce meta–analysis reports. The first step in producing a meta–analysis report is the discovery of relevant studies associated with the question under consideration. For example, the meta–analysis for correlation between cardiovascular risk factors and venous thrombosis would require all individual studies comparing the two.

Figure 5.6 presents the first screen (*Discovery*) in our proof of concept of a user interactive system to produce meta–analysis reports. This screen allows researchers to define parameters used to identify relevant studies. The two important criteria used in the discovery process process include patient groups and characteristics associated with them. In our example meta–analysis, patient groups include patients with Venous Thrombosis and characteristics include cardiovascular risk factors and demographic information. *Discovery* allows researchers to add multiple patient groups by defining an attribute associated with it. This attribute could be different factors associated with the group such as disease, drug dosage etc. Similarly, it allows the addition of multiple characteristics to be used during search.

As researchers type in parameter values, *Discovery* will auto–complete and link them to appropriate UMLS (or source ontology) concepts. The use of a ontology and KB allows researchers to use higher level abstract concepts. For example, instead of listing multiple cardiovascular risk factors such as Obesity, Diabetes, Hypertension individually, they can

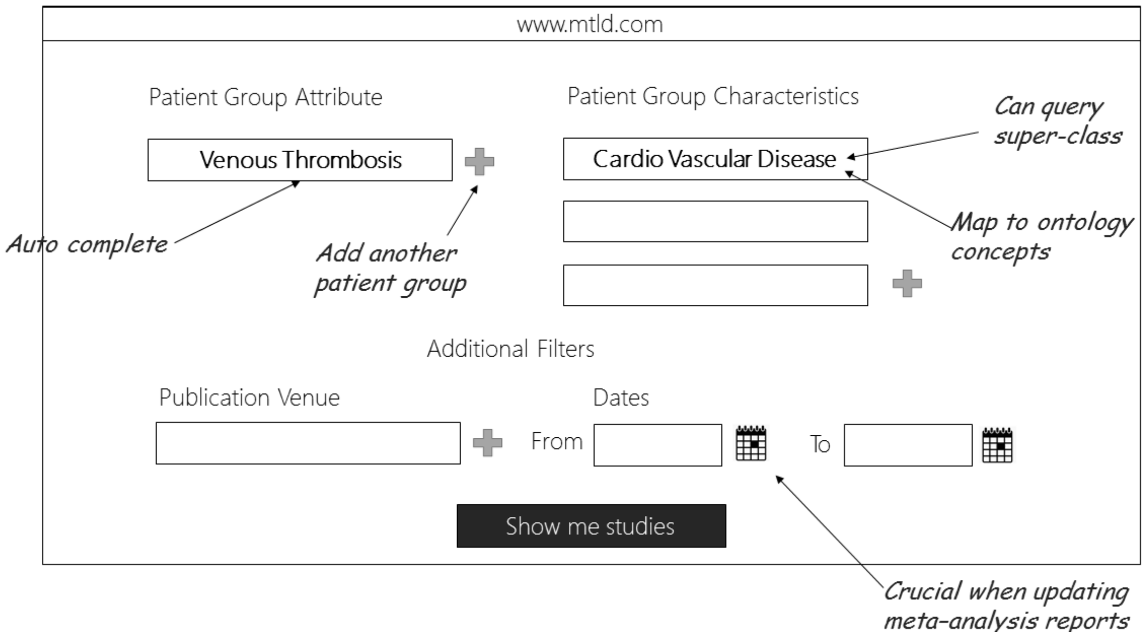


FIG. 5.6. An interactive interface will allow users to define search criteria.

specify a general concept such as ‘Cardiovascular Risk Factors’. Facts such Obesity is a type of Cardiovascular risk factor are encoded in KBs such as UMLS and SPARQL queries use these associations during data retrieval.

Additional parameters can be added to further narrow the search results. Any patient group characteristic with units of measurement can be used. For example, researchers can specify that Age should be greater than 30; Body Mass Index greater than 28 and so on. Such characteristics include (but are not limited to): Age, Blood Pressure, Sugar/Glucose level, Triglyceride level, Body Mass Index. Similar filter can be applied on the size of the patient group. Researchers can specify a range, minimum, or, a maximum value on these attributes. Finally, filters can be applied on publication metadata such as publication date, venue, author and so on.

SPARQL queries are automatically generated based on parameters defined on the *Discovery* screen. Figure 5.7 (a) shows the SPARQL query generated for the parameters de-

(a) Find studies correlating *venous thrombosis* and *obesity*.

```
SELECT ?study ?table WHERE {
  ?patGroup mto:hasGroupAttribute
     umls:Venous_Thrombosis.
  ?patObsGroup mto:hasPatientGroup ?patGroup.
  ?obs mto:hasPatientGroupObservation ?patObsGroup;
     mto:hasVariable umls:Cardio_Vascular_Risk_Factors;
     mto:observationInTable ?table.
  ?table mto:tableInStudy ?study.}
```

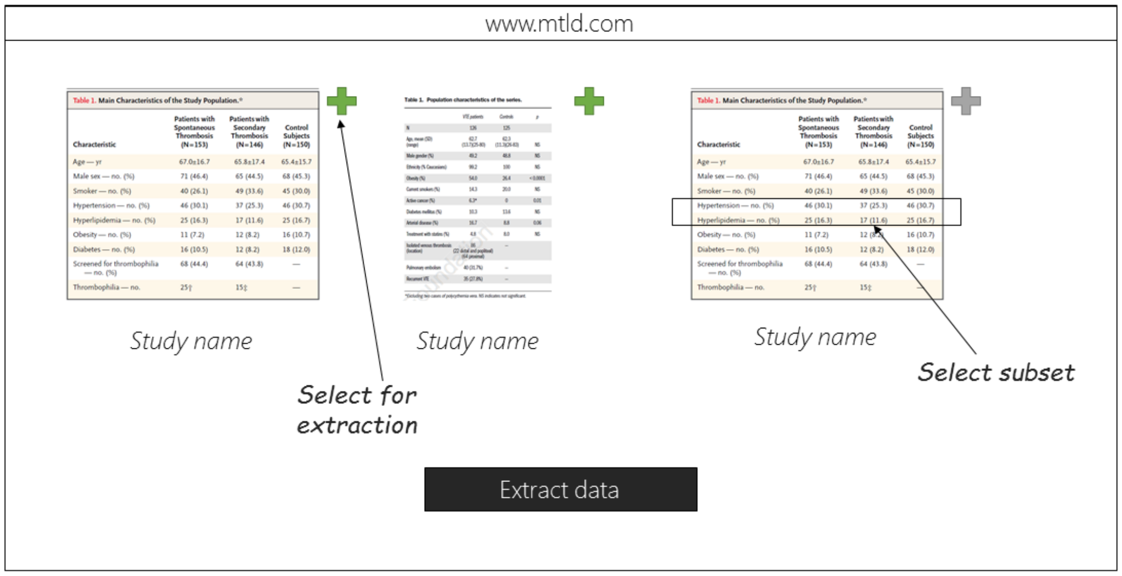
(b) Get all observations for a given table.

```
SELECT ?obs WHERE {
  mto:table3 mto:tableObservation ?obs.}
```

FIG. 5.7. SPARQL query for parameters defined in Figure 5.6.

fined in Figure 5.6. This query identifies all studies with patient group attribute Venous Thrombosis and a cardiovascular risk factor as a characteristic. Since both the query and the tables are grounded in the same ontology, search becomes an efficient process.

The *Extract* screen (Figure 5.8) displays search results for the user query. This screen allows researchers to evaluate the results and select the data to be included for the integration phase. Results are presented in the form of tables along with the studies they belong to. Additional information can be obtained by clicking on the name of the study. Researchers can either select an entire table or as often is the case select a subset of table to be included for the next phase. Figure 5.7 (b) shows the SPARQL query used to extract the data once the tables are selected. This query extracts all observations from a given table. Appropriate SPARQL queries are generated to extract either all or subset of observations from a table. Once the data is extracted, it can be integrated to produce results to be included in the meta–analysis report.



www.mtld.com

Table 1. Main Characteristics of the Study Population.*

Characteristic	Patients with Spontaneous Thrombosis (N=153)	Patients with Secondary Thrombosis (N=146)	Control Subjects (N=150)
Age — yr	67.0±16.7	65.8±17.4	65.4±15.7
Male sex — no. (%)	71 (46.4)	65 (44.5)	68 (45.3)
Smoker — no. (%)	40 (26.1)	49 (33.6)	45 (30.0)
Hypertension — no. (%)	46 (30.1)	37 (25.3)	46 (30.7)
Hyperlipidemia — no. (%)	25 (16.3)	17 (11.6)	25 (16.7)
Obesity — no. (%)	11 (7.2)	12 (8.2)	16 (10.7)
Diabetes — no. (%)	16 (10.5)	12 (8.2)	18 (12.0)
Screened for thrombophilia — no. (%)	68 (44.4)	64 (43.8)	—
Thrombophilia — no.	25†	15‡	—

Study name

Table 1. Population characteristics of the series.

	VTE patients	Controls	p
N	126	125	
Age, mean (SD) (range)	62.7 (13.7)(25-80)	62.3 (11.3)(26-83)	NS
Male gender (%)	49.2	48.8	NS
Ethnicity (% Caucasians)	99.2	100	NS
Obesity (%)	54.0	26.4	< 0.0001
Current smokers (%)	14.3	20.0	NS
Active cancer (%)	6.3*	0	0.01
Diabetes mellitus (%)	10.3	13.6	NS
Arterial disease (%)	16.7	8.8	0.06
Treatment with statins (%)	4.8	8.0	NS
Isolated venous thrombosis (location)	86 (22 distal and popliteal) (64 proximal)	–	
Pulmonary embolism	40 (31.7%)	–	
Recurrent VTE	35 (27.8%)	–	

*Excluding two cases of polycythemia vera. NS indicates not significant.

Study name

Table 1. Main Characteristics of the Study Population.*

Characteristic	Patients with Spontaneous Thrombosis (N=153)	Patients with Secondary Thrombosis (N=146)	Control Subjects (N=150)
Age — yr	67.0±16.7	65.8±17.4	65.4±15.7
Male sex — no. (%)	71 (46.4)	65 (44.5)	68 (45.3)
Smoker — no. (%)	40 (26.1)	49 (33.6)	45 (30.0)
Hypertension — no. (%)	46 (30.1)	37 (25.3)	46 (30.7)
Hyperlipidemia — no. (%)	25 (16.3)	17 (11.6)	25 (16.7)
Obesity — no. (%)	11 (7.2)	12 (8.2)	16 (10.7)
Diabetes — no. (%)	16 (10.5)	12 (8.2)	18 (12.0)
Screened for thrombophilia — no. (%)	68 (44.4)	64 (43.8)	—
Thrombophilia — no.	25†	15‡	—

Study name

FIG. 5.8. Users can select entire table or subset of the table to be integrated for the final meta–analysis report.

5.4 Evaluation

We present an evaluation of how well *TABEL* performs in inferring the semantics of medical tables and its utility in discovering relevant studies associated with a meta–analysis report.

5.4.1 Semantics of Medical Tables

Dataset. We choose a meta–analysis report analyzing correlation between *cardiovascular risk factors* and *venous thromboembolism* (Ageno *et al.* 2008) for our evaluation purposes. We further obtain publicly available medical studies used to generate our selected meta–analysis report and manually extract tables embedded in these documents. Figure 5.9 presents details about tables extracted from individual studies. We manually split every header cell content into query string and metadata to evaluate the *Normalize* module. Simi-

Table # 1, 2 from González-Ordóñez *et al.*(González-Ordóñez *et al.* 2003)
Table # 1, 2 from Paganin *et al.*(Paganin *et al.* 2003)
Table # 1 from Prandoni *et al.*(Prandoni *et al.* 2003)
Table # 2 from Frederiksen *et al.*(Frederiksen *et al.* 2004)
Table # 1 from Deguchi *et al.*(Deguchi *et al.* 2005)

FIG. 5.9. Tables numbers extracted from each study.

larly, every header cell is also mapped to appropriate concepts from SNOMED CT, UMLS Metathesaurus, and DBpedia to evaluate quality of header cell annotations.

Experimental Setup. The extracted medical tables go through the *Normalize*, *Query and Rank* and *Generate RDF* modules finally leading to RDF linked data as output. In the absence of the *Joint Inference* step, the top ranked class (or instance) from the ranked list of candidates is assigned to the column and row headers. For each query string, a ranked list of candidate concepts (or instances in case of DBpedia) are generated as described in section 5.3. The size of candidate set is 25 for UMLS and 100 for DBpedia and SNOMED. Each candidate set also includes an additional No–Annotation (N.A.) entity.

Evaluating Normalize. The *Normalize* module accurately splits the header cell content into query string and metadata for 109 out of 123 header cells (88.62%). The errors in the remaining cases (14) were a result of choosing incorrect patterns in cases of multiple of pattern matches.

Evaluating Header Cell Annotations. Figure 5.10 shows the distribution of the correct header cell concept at different ranks for 122 header cells across seven tables. UMLS and Hybrid give the best performance with approximately 60% of the correct concepts at rank 1. DBpedia and SNOMED CT perform poorly with only 53.28% and 44.27% of cor-

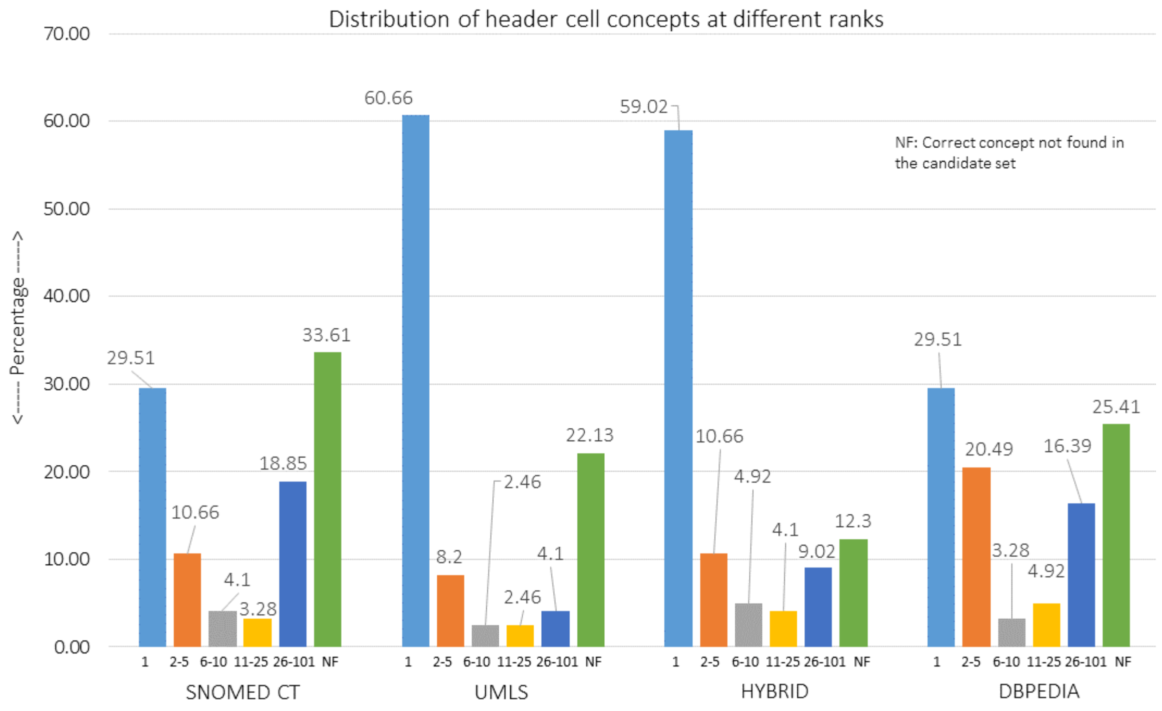


FIG. 5.10. Distribution of header cell concepts at different ranks.

rect concepts in the top ten ranks. UMLS outperforms SNOMED because of its larger coverage of concepts; several expected concepts were absent in SNOMED CT. Since DBpedia is a general purpose KB, its query results often include non medical concepts, leading to its overall decline in performance. Improvements for DBpedia can be obtained by restricting concepts from the medical domain.

Query string that consisted of a modifier (e.g., *no diabetes*, *treatment with statins*, *active cancer*) posed a challenge across all knowledge sources. We also found evidence of overlap between the terms used in sources such as SNOMED CT and terms used in medical tables, with 35.24% of queries resulting in an *exact query match*. While the number of concepts at rank 1 is nearly identical for UMLS and Hybrid, the number of times a concept is not found in the candidate set is far lower for the latter (12.3%) as compared to former (22.13%). Results suggest that a Hybrid strategy for *Query and Rank* would produce better

Risk Factor	*Associated Studies*
Obesity	Table 1 (Paganin *et al.* 2003), Table 1 (Prandoni *et al.* 2003)
Hypertension	Table 1 (Prandoni *et al.* 2003), Table 2 (Frederiksen *et al.* 2004)
Diabetes	Table 1 (Prandoni *et al.* 2003), Table 2 (Frederiksen *et al.* 2004)
Smoking	Table 1 (Prandoni *et al.* 2003), Table 2 (Frederiksen *et al.* 2004)

FIG. 5.11. Table # and individual studies associated with each risk factor.

candidate sets.

5.4.2 Discovering Relevant Studies

Our selected meta–analysis (Ageno *et al.* 2008) analyzed the correlation between seven different cardio vascular risk factors and venous thrombosis namely Obesity (Body Mass Index > 30), Diabetes Mellitus, Smoking, Total Cholesterol, Triglycerides and LDL & HDL Cholesterol. The correlation between each risk factor and venous thrombosis was performed independently, each correlation using a separate set of individual studies. Figure 5.11 shows individual studies associated with four of the risk factor.

We begin by inferring the semantics and generating RDF Linked Data from the seven tables listed in Figure 5.9. We execute a SPARQL query (Figure 5.7 (a)) to identify studies analyzing correlation between Venous Thrombosis and the four risk factors listed in Figure 5.11. Since our current dataset includes studies only related to Venous Thrombosis, we drop the clause associated with patient group attribute and only query using risk factors.

We compute precision and recall using the tables listed in Figure 5.11 as relevant tables or ground truth for each query. We obtain an average precision of **0.79** of and recall of **0.75** of over the four queries. Errors in the retrieval process can be attributed to the incorrect semantics inferred during header cell annotations. In the case of *Diabetes* and *Smoking*, header cells in one of the expected tables were mapped to the concepts *Smoker* and *Diabetes* respectively; whereas the concepts used in the SPARQL query were *Diabetes_Mellitus* and

Smoking. The case of *Smoker* vs. *Smoking* is interesting: the latter is represented as an *individual behavior* in UMLS whereas the former is represented as a *finding*. However both concepts seem accurate to describe whether a person is a smoker. Similar ambiguity arises in the cases like *triglyceride*: is the correct concept for *triglyceride*, *Triglyceride – Biologically Active Substance* or *Triglyceride level – finding*. These challenges will have to be either tackled in *Query & Rank* by disambiguating to similar type of entity (i.e., always prefer type *Finding*) or in retrieval phase by executing multiple queries for related concepts.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Generating an explicit representation of the meaning which is implicit in tabular data will benefit several research areas and applications from multiple domains. In this thesis, we presented *TABEL* – a domain independent and extensible framework to infer the semantics of tables and make it explicit by representing them as RDF Linked Data. To the best of our knowledge, we were the first to define and capture the semantics of a table by mapping header cells to classes, data cells to entities and pair of columns to relations from a given ontology and knowledgebase.

We demonstrate that it is possible to generate high quality linked data from tables by jointly inferring the semantics of header and data cells using techniques from probabilistic graphical models. We introduce a novel inference technique, Semantic Message Passing, which incorporates semantics and background knowledge from Linked Data sources into standard message passing techniques. We introduce techniques to infer the semantics and generate candidate properties for literal values. We also demonstrate the extensibility and domain independence of our techniques by developing an application of *TABEL* for the healthcare domain. We adapt our framework to infer the semantics of tables found in medical literature (medical tables) and develop a proof of concept for an application to generate meta-analysis reports automatically, which is built on top of the semantics inferred from medical tables. Finally, our work explores different models of user interaction with *TABEL* and its impact on the quality of inferred semantics.

A thorough evaluation with experiments over dataset of web and medical tables showed promising results. Our framework adopts a “data–driven” approach; first we generate candidates entities for data cell values and use them to generate candidate classes and relations. The candidates entities are ranked and the top ranked ones are used as initial assignments for variables in the graphical model. A large percentage of variable node assignments stabilize after the first iteration indicating the benefits of this approach. However, our experiments with medical tables also highlight the challenges of such an approach. The data cell values often represent counts of patients which are captured via a relationship between the column and row headers. Limited context in header cells made disambiguation a challenge and the multi–dimensional nature of medical tables did not allow us to exploit correlations (and hence joint inference) typically found in regular tables.

6.2 Future Work

6.2.1 Extending Semantic Message Passing

Our existing implementation of the Joint Inference module instantiates factor nodes ψ_3 and ψ_4. The next step would be to instantiate the factor node ψ_5 which would compute agreement between the classes assigned to the column headers. The additional semantics from ψ_5 will improve the quality of inferred semantics, especially in tables like the ones found in medical research papers in which data cells are not useful during the inference process.

Semantic Message Passing allows data cell variables to exploit semantic messages to update their assignments. Instantiating ψ_5 would also allow the column header and relation variables, both to receive and exploit these messages. Semantic Message Passing can be further made robust if it is able to identify 'problematic' header and data cells and eliminate them during inference. Problematic variables could constitute ones with low confidence assignments or ones that keep changing their assignment in every iteration.

6.2.2 Literal Values

Our preliminary work on *NumKB* (Chapter 3.2.2) show encouraging results to build upon. An immediate next step would be constructing NumKB for all datatype properties from DBpedia (or a simialar KB) and providing support for properties with units. The challenge of units can be handled by either mapping literal values into the units of values as represented in the kb or representing value distributions for each possible unit of properties when the kb is generated. Our existing approach for generating NumKB assumes the properties values to belong to a normal distribution which always may not be true. An alternate approach would be to experiment using non–parametric techniques such as Kernel Density Estimation to estimate distribution of the property values.

6.2.3 Combine Schema and Instance approach

As pointed earlier, our current approach for generating candidates is a "data–driven" technique; we first generate candidate entities, which are then used to generate candidate classes and relations. Additional candidate classes can be independently generated using column header strings. It is also possible to infer the semantics of header strings independently of the data cells using schema based techniques such as the ones developed by Han et al. (Han, Finin, & Joshi 2011). Combining both schema and data–driven approaches can yield better results.

6.3 Concluding Remarks

This thesis presented an an domain independent and extensible framework which demonstrated that it is possible to generate high quality linked data from tables by jointly inferring the semantics of column headers, values (string and literal) in table cells, and relations between columns augmented with background knowledge from open data sources such as the Linked Open Data cloud.

REFERENCES

[1] Ageno, W.; Becattini, C.; Brighton, T.; Selby, R.; and Kamphuisen, P. W. 2008. Cardiovascular risk factors and venous thromboembolism a meta-analysis. *Circulation* 117(1):93–102.

[2] Berners-Lee, T.; Hendler, J.; Lassila, O.; et al. 2001. The semantic web. *Scientific American* 284(5):28–37.

[3] Berners-Lee, T. 2006. Linked data. http://www.w3.org/DesignIssues/LinkedData.html.

[4] 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.

[5] Cafarella, M. J.; Halevy, A. Y.; Wang, Z. D.; Wu, E.; and Zhang, Y. 2008. Webtables: exploring the power of tables on the web. *PVLDB* 1(1):538–549.

[6] Cohen, A. M.; Ambert, K.; and McDonagh, M. 2009. Cross-topic learning for work prioritization in systematic review creation and update. *Journal of the American Medical Informatics Association* 16(5):690–704.

[7] Cohen, A.; Adams, C.; Davis, J.; Yu, C.; Yu, P.; Meng, W.; Duggan, L.; McDonagh, M.; and Smalheiser, N. 2010. Evidence-based medicine, the essential role of systematic reviews, and the need for automated text mining tools. In *1st Int. Health Informatics Symposium*, 376–380. ACM.

[8] Dameron, O.; Besana, P.; Zekri, O.; Bourdé, A.; Burgun, A.; Cuggia, M.; et al. 2012. Owl model of clinical trial eligibility criteria compatible with partially-known information. In *SWAT4LS*.

[9] Deguchi, H.; Pecheniuk, N. M.; Elias, D. J.; Averell, P. M.; and Griffin, J. H. 2005. High-density lipoprotein deficiency and dyslipoproteinemia associated with venous thrombosis in men. *Circulation* 112(6):893–899.

[10] Deng, D.; Jiang, Y.; Li, G.; Li, J.; and Yu, C. 2013. Scalable column concept determination for web tables using large knowledge bases. *Proceedings of the VLDB Endowment* 6(13):1606–1617.

[11] Ding, L.; DiFranzo, D.; Graves, A.; Michaelis, J. R.; Li, X.; McGuinness, D. L.; and Hendler, J. A. 2010. TWC data-gov corpus: incrementally generating linked government data from data.gov. In *Proc 19th WWW*, 1383–1386. ACM.

[12] Dredze, M.; McNamee, P.; Rao, D.; Gerber, A.; and Finin, T. 2010. Entity Disambiguation for Knowledge Base Population. In *Proc. 23rd Int. Conf. on Computational Linguistics*.

[13] Embley, D. W.; Lopresti, D. P.; and Nagy, G. 2006. Notes on contemporary table recognition. In *Document Analysis Systems*, 164–175.

[14] Ermilov, I.; Auer, S.; and Stadler, C. 2013. Crowd–sourcing the large-scale semantic mapping of tabular data. *ACM Web Science Conference*.

[15] Frederiksen, J.; Juul, K.; Grande, P.; Jensen, G. B.; Schroeder, T. V.; Tybjærg-Hansen, A.; and Nordestgaard, B. G. 2004. Methylenetetrahydrofolate reductase polymorphism (c677t), hyperhomocysteinemia, and risk of ischemic cardiovascular disease and venous thromboembolism: prospective and case-control studies from the copenhagen city heart study. *Blood* 104(10):3046–3051.

[16] González-Ordóñez, A. J.; Fernández-Carreira, J. M.; Fernández-Alvarez, C. R.; Obaya, R. V.; Macías-Robles, M. D.; González-Franco, A.; and Garcia, M. A. 2003. The

concentrations of soluble vascular cell adhesion molecule-1 and lipids are independently associated with venous thromboembolism. *haematologica* 88(9):1035–1043.

[17] Han, L.; Finin, T.; Parr, C.; Sachs, J.; and Joshi, A. 2008. RDF123: from Spreadsheets to RDF. In *Proc. 7th Int. Semantic Web Conf.* Springer.

[18] Han, L.; Finin, T.; and Joshi, A. 2011. GoRelations: An Intuitive Query System for DBpedia. In *Proc. Joint Int. Semantic Technology Conf.*, LNCS. Springer.

[19] Hassanzadeh, O.; Kementsietsidis, A.; Lim, L.; Miller, R. J.; and Wang, M. 2009. Linkedct: A linked data space for clinical trials. *arXiv preprint arXiv:0908.0567*.

[20] Hurst, M. 2006. Towards a theory of tables. *IJDAR* 8(2-3):123–131.

[21] Kilicoglu, H.; Demner-Fushman, D.; Rindflesch, T. C.; Wilczynski, N. L.; and Haynes, R. B. 2009. Towards automatic recognition of scientifically rigorous clinical research evidence. *Journal of the American Medical Informatics Association* 16(1):25–31.

[22] Kiritchenko, S.; de Bruijn, B.; Carini, S.; Martin, J.; and Sim, I. 2010. Exact: automatic extraction of clinical trial characteristics from journal publications. *BMC medical informatics and decision making* 10(1):56.

[23] Knoblock, C. A.; Szekely, P.; Ambite, J. L.; Gupta, S.; Goel, A.; Muslea, M.; Lerman, K.; Taheriyan, M.; and Mallick, P. 2012. Semi-automatically mapping structured sources into the semantic web. In *Proceedings of the Extended Semantic Web Conference*.

[24] Knoblock, C. A.; Szekely, P.; Gupta, S.; Manglik, A.; Verborgh, R.; Yang, F.; and de Walle, R. V. 2013. Publishing data from the smithsonian american art museum as

linked open data. In *Proceedings of the ISWC 2013 Posters & Demonstrations Track*, 129–132.

[25] Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

[26] Langegger, A., and Wob, W. 2009. Xlwrap - querying and integrating arbitrary spreadsheets with SPARQL. In *Proc. 8th Int. Semantic Web Conf.*

[27] Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, Soviet Physics Doklady.

[28] Limaye, G.; Sarawagi, S.; and Chakrabarti, S. 2010. Annotating and searching web tables using entities, types and relationships. In *Proc. 36th VLDB*.

[29] 2014. Medical Table Ontology. http://ebiquity.umbc.edu/ontology/mto/v1/.

[30] Mulwad, V.; Finin, T.; Syed, Z.; and Joshi, A. 2010. Using linked data to interpret tables. In *Proc. 1st Int. Workshop on Consuming Linked Data*.

[31] Mulwad, V. 2010. T2LD - An automatic framework for extracting, interpreting and representing tables as Linked Data. Master's thesis, University of Maryland, Baltimore County.

[32] Munoz, E.; Hogan, A.; and Mileo, A. 2013. Triplifying wikipedia's tables. In *1st International Workshop on Linked Data for Information Extraction*.

[33] Muñoz, E.; Hogan, A.; and Mileo, A. 2014. Using linked data to mine rdf from wikipedia's tables. In *7th ACM International Conference on Web Search and Data Mining*, 533–542. ACM.

[34] Paganin, F.; Bourde, A.; Yvin, J.-L.; Genin, R.; Guijarro, J.-L.; Bourdin, A.; and Lassalle, C. 2003. Venous thromboembolism in passengers following a 12-h flight: a case-control study. *Aviation, space, and environmental medicine* 74(12):1277–1280.

[35] Polfliet, S., and Ichise, R. 2010. Automated mapping generation for converting databases into linked data. In *Proc. 9th Int. Semantic Web Conf.*

[36] Prandoni, P.; Bilora, F.; Marchiori, A.; Bernardi, E.; Petrobelli, F.; Lensing, A. W.; Prins, M. H.; and Girolami, A. 2003. An association between atherosclerosis and venous thrombosis. *New England Journal of Medicine* 348(15):1435–1441.

[37] Puranik, N. 2012. A specialist approach for classification of column data. Master's thesis, University of Maryland, Baltimore County.

[38] Sackett, D.; Rosenberg, W.; Gray, J.; Haynes, R.; and Richardson, W. 1996. Evidence based medicine: what it is and what it isn't. *Bmj* 312(7023):71.

[39] Sahoo, S. S.; Halb, W.; Hellmann, S.; Idehen, K.; Thibodeau Jr, T.; Auer, S.; Sequeda, J.; and Ezzat, A. 2009. A survey of current approaches for mapping of relational databases to rdf. Technical report, W3C.

[40] Salton, G., and Mcgill, M. J. 1986. *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc.

[41] Sarker, A.; Mollá, D.; and Paris, C. 2013. An approach for query-focused text summarisation for evidence based medicine. In *Artificial Intelligence in Medicine*. Springer. 295–304.

[42] Schuyler, P. L.; Hole, W. T.; Tuttle, M. S.; and Sherertz, D. D. 1993. The UMLS metathesaurus: representing different views of biomedical concepts. *Bulletin of the Medical Library Association* 81(2):217.

[43] Shankar, R. D.; Martins, S. B.; O'Connor, M. J.; Parrish, D. B.; and Das, A. K. 2006. Epoch: an ontological framework to support clinical trials management. In *Proceedings of the international workshop on Healthcare information and knowledge management*, 25–32. ACM.

[44] Sim, I.; Tu, S. W.; Carini, S.; Lehmann, H. P.; Pollock, B. H.; Peleg, M.; and Wittkowski, K. M. 2013. The ontology of clinical research (ocre): An informatics foundation for the science of clinical research. *Journal of biomedical informatics*.

[45] Stearns, M. Q.; Price, C.; Spackman, K. A.; and Wang, A. Y. 2001. SNOMED clinical terms: overview of the development process and project status. In *AMIA Symposium*, 662. AMIA.

[46] Stenzhorn, H.; Weiler, G.; Brochhausen, M.; Schera, F.; Kritsotakis, V.; Tsiknakis, M.; Kiefer, S.; and Graf, N. 2010. The obtima system-ontology-based managing of clinical trials. *Stud Health Technol Inform* 160(Pt 2):1090–1094.

[47] Suchanek, F. M.; Abiteboul, S.; and Senellart, P. 2011. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *PVLDB* 5(3):157–168.

[48] Suchanek, F. M.; Kasneci, G.; and Weikum, G. 2007. Yago: A Core of Semantic Knowledge. In *16th International World Wide Web Conference*. New York: ACM Press.

[49] Summerscales, R. L.; Argamon, S.; Bai, S.; Huperff, J.; and Schwartzff, A. 2011. Automatic summarization of results from clinical trials. In *International Conference on Bioinformatics and Biomedicine*, 372–377. IEEE.

[50] Syed, Z., and Finin, T. 2011. Creating and Exploiting a Hybrid Knowledge Base for Linked Data. In *Agents and Artificial Intelligence*. Springer. 3–21.

[51] Syed, Z.; Finin, T.; Mulwad, V.; and Joshi, A. 2010. Exploiting a Web of Semantic Data for Interpreting Tables. In *Proceedings of the 2nd Web Science Conference*.

[52] Szekely, P.; Knoblock, C. A.; Yang, F.; Zhu, X.; Fink, E.; Allen, R.; and Goodlander, G. 2013. Connecting the Smithsonian American Art Museum to the Linked Data Cloud. In *Proceedings of the 10th Extended Semantic Web Conference*. Awarded Best In-Use Paper at ESWC 2013.

[53] Van Valkenhoef, G.; Tervonen, T.; Zwinkels, T.; De Brock, B.; and Hillege, H. 2013. Addis: a decision support system for evidence-based medicine. *Decision Support Systems* 55(2):459–475.

[54] Vavliakis, K. N.; Grollios, T. K.; and Mitkas, P. A. 2010. RDOTE- transforming relational databases into semantic web data. In *9th Int. Semantic Web Conf.*

[55] Venetis, P.; Halevy, A.; Madhavan, J.; Pasca, M.; Shen, W.; Wu, F.; Miao, G.; and Wu, C. 2011. Recovering semantics of tables on the web. In *Proc. 37th VLDB*.

[56] Wang, J.; Wang, H.; Wang, Z.; and Zhu, K. Q. 2012. Understanding tables on the web. In *Conceptual Modeling*. Springer. 141–155.

[57] Wu, W.; Li, H.; Wang, H.; and Zhu, K. Q. 2012. Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 481–492. ACM.

[58] Zhang, Z. 2014a. Disambiguating web tables using partial data. In *13th International Semantic Web Conference (Posters & Demos)*, 213–216. CEUR Workshop Proceedings.

[59] Zhang, Z. 2014b. Learning with partial data for semantic table interpretation. *Knowledge Engineering and Knowledge Management* 607–618.

[60] Zhang, Z. 2014c. Towards efficient and effective semantic table interpretation. In *12th International Semantic Web Conference*, 487–502. Springer International Publishing.

[61] Zwicklbauer, S.; Einsiedler, C.; Granitzer, M.; and Seifert, C. 2013. Towards disambiguating web tables. In *12th International Semantic Web Conference (Posters & Demos)*, 205–208. CEUR Workshop Proceedings.