T2LD - An automatic framework for extracting, interpreting and representing tables as Linked Data

by

Varish 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
M.S. Computer Science
2010

ABSTRACT

Title of Thesis: *T2LD* - An automatic framework for extracting, interpreting and representing tables as Linked Data

Varish Mulwad, M.S. Computer Science, August 2010

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

We present an automatic framework for extracting, interpreting and generating linked data from tables. In the process of representing tables as linked data, we assign every column header a class label from an appropriate ontology, link table cells (if appropriate) to an entity from the Linked Open Data cloud and identify relations between various columns in the table, which helps us to build an overall interpretation of the table. Using the limited evidence provided by a table in the form of table headers and table data in rows and columns, we adopt a novel approach of querying existing knowledgebases such as Wikitology, DBpedia etc. to figure the class labels for table headers. In the process of entity linking, besides querying knowledgebases, we use machine learning algorithms like support vector machine and algorithms which can learn to rank entities within a given set to link a table cell to entity. We further use the class labels, linked entities and information from the knowledgebases to identify relations between columns. We prototyped a system to evaluate our approach against tables obtained from Google Squared, Wikipedia and set of tables obtained from a dataset which Google shared with us.

T2LD - An automatic framework for extracting, interpreting and representing tables as Linked Data

by

Varish 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
M.S. Computer Science
2010

Dedicated to "aajji", my late grandmother

Tungabai.H.Mulwad

(June 24, 1924 - October 31, 2008)

ACKNOWLEDGMENTS

Words are inadequate to express my deep gratitude to Dr. Tim Finin for providing me an opportunity to work under him and supporting me as a Research Assistant in the Ebiquity Research Lab at UMBC. I would like to thank him for the constant encouragement, support and valuable guidance throughout the progress of my Master's Thesis. I would like to thank Dr.Anupam Joshi, Dr. Tim Oates and Dr. Evelyne Viegas for agreeing to be on my thesis committee and providing feedback and suggestions.

I would like to thank Zareen Syed, member of the Ebiquity Lab for her guidance, valuable suggestions and inputs throughout the progress of my work. Thanks to Palani, Wenjia and all other members of the Ebiquity Research Lab for their time and support. I would also like to thank Ebiquity and UMBC alumni Kishor Datar, for proving me guidance and mentoring me through my first year at UMBC. I would like to thank my three evaluators Mayank, Vivek and Bhushan for completing the evaluations in quick time and also for their valuable suggestions on how we could improve the evaluations.

For the first time in my life, I had to stay away from my parents and family. Life would have been difficult if it wasn't for Mayank, Niyati, Vivek, Bhushan, Krishna (KV), Pradeep Guin, Tejas, Tushar, Gitika, and Dr. Shyam and Dr. Shoba Shirali. I want to thank them for making me feel at home in Baltimore and at UMBC.

The success of my work was not possible without the extreme support and love of my parents and the rest of my family, who are always a source of inspiration to me.

TABLE OF CONTENTS

LIST OF FIGURES

List of Algorithms

Chapter 1

INTRODUCTION

In Nineteen Eighty Nine, Sir Tim Berners-Lee proposed and introduced a distributed hypertext system (Berners-Lee 1989) , which eventually led to the birth of the World Wide Web. The World Wide Web truly changed the way we do computing and the way we share information and services. If the 1980s' and the 1990s' were the eras of desktop computing, the 21st century belongs to the Web. The web we know today is a web of hyperlinked documents in which one document is connected to another document via a hyperlink, which in turn is connected to another document and so on. This web of hyperlinked documents is good for humans to navigate and understand, but it makes no sense for machines and software agents.

The original idea of the web also contained seeds for another web - a smarter and intelligent web. Today we identify that web as the Semantic Web. The W3C's Semantic Web Activity group[1] defines the Semantic Web as a platform providing a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. One of the primary goal and vision of the Semantic Web is to create a web of data instead of a web of documents, a web in which data items present on the web are connected to each other. The goal of the Semantic Web is also to make the web a platform

[1] http://www.w3.org/2001/sw/

for sharing data, information and knowledge.

Sir Tim Berners-Lee also introduced the notion of linked data (Bizer 2009) in which he outlined the best practices for exposing and sharing structured data on the web. The principles of linked data state that:

- Every data item should be identified by an URI
- The URI should be an http URI which can be dereferenced
- The dereferenced page should provide useful information about the thing
- Every data item should also point to URIs of other related things

For this web of data vision to be realized, we need data to be available in a standard format suitable for the Semantic Web. There are two key points a) availability of data and b) data in a standard and structured format. The first point has been already taken care of as there is a huge amount raw data already present on the Web. In July 2008, Google software engineers Jesse Alpert and Nissan Hajaj announced that Google had indexed 1 trillion unique URLs[2] , which meant one trillion unique documents on the World Wide Web. The World Wide Web continues to grow, thus as of today we must be having more than a trillion documents on the Web. A lot of data stored in these documents is actually stored in html tables. Google researchers (Cafarella *et al.* 2008) estimated that there are about 14.1 billion html tables, of which 154 million contain high quality relational data.

There is a lot of data available that is not a part of the web. A wide variety of interesting domains such as health-care, biotechnology, finance store data in tabular form, either as spreadsheets, CSV files or database tables. As a part of the Linked Open Data initiative, the US and UK governments have also shared publicly available government data in tabular

[2] *We knew the web was big...* - http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

City	State	Mayor	Population
Baltimore	MD	S.Dixon	640,000
Philadelphia	PA	M.Nutter	1,500,000
Washington	DC	A.Fenty	595,000
New York	NY	M.Bloomberg	8,400,000
Boston	MA	T.Menino	610,000

FIG. 1.1. A simple table about cities in the United States. The column header represents the type of data stored in columns; values in the columns represent instances of that type.

forms. These tables can be thought of as a collection or a gigantic database of information and knowledge which should be utilized for enriching our knowledge.

We have a gigantic database of knowledge and information, but it cannot be exploited on the Semantic Web in its present form. It is possible for humans to convert this massive raw data into a structured and standard format for the Semantic Web, but it is time consuming and difficult to do so. To be able to exploit this information, we need mechanisms and systems that can convert this data into a structured and standard format.

In this research, we focus on the converting the data stored in a tabular form to a structured format. A table can be defined as a two-dimensional presentation of logical relationships between groups of data (Vanoirbeek 1992). The table columns header represents the type of data that is stored in that particular column, whereas the values in the column represent the instances of the type. The values in the same row of the table hold some form of relationship amongst each other.

Consider a simple table shown in Figure 1.1. Given the structure of the table, there is a lot of information that can be extracted from the table. Using the column header, along with the column and row contents, we can query knowledge bases to figure out what data is stored in the table. For instance the values in column 1 could be cities in the US or could be references to football teams of the respective cities. Querying knowledge bases

like Wikipedia, DBpedia (Bizer *et al.* 2009) we can figure out that these are largest cities in their respective states on the east coast of the United States. We can also determine that the column 3 contains people. On further fine grain reasoning we can determine that it's a type of politician or type of mayor. Querying DBpedia, we can figure out that mayor is a property of the ontology that could be used to describe column 1. Finally we can also figure out that the values in the last column of the table are actually values of the property population and the property population is associated with the data stored in column 1.

This thesis presents a mechanism to convert the data stored in tables into linked data. The process of converting data stored in tables to linked data involves the following steps. For every column header in the table, we label it with a class label from an appropriate ontology. Every table cell value is linked to an appropriate entity from the linked data cloud. We also identify and discover relationships that may exist between the columns of the table. All this information is then published as RDF on the Semantic Web. To achieve this task, we use the information from table column headers, table rows and the values in the columns and we query Wikitology (Finin & Syed 2010) knowledge base (KB), which consist information from Wikipedia, DBpedia, Freebase (Bollacker *et al.* 2008), WordNet (Miller 1995) and Yago (Suchanek, Kasneci, & Weikum 2007). Processing the results returned by our KB, we predict the class labels for every table column header. Using this additional evidence, we re-query our KB to complete the step of linking the table cells to entities from the linked data cloud. Once the entity linking is complete, we discover relations between the columns.

So for example, when we convert the data in table in figure 1.1 into linked data, we will assign column header city a class label like dbpedia-owl:city. We also link the table cell value Baltimore to its DBpedia page - http://dbpedia.org/resource/Baltimore. Further we also discover the relation dbpedia-owl:largestCity between columns 1 and 2 of the table.

We have also prototyped a system which implements the mechanism described above

to convert tables into linked data. We test our approach against tables generated from Google Squared[3], tables extracted from Wikipedia pages and tables from a dataset of tables occurring on the web, Google shared with us.

[3] http://www.google.com/squared

Chapter 2

BACKGROUND AND RELATED WORK

In this chapter, we provide some background knowledge about the knowledge bases we use and then describe some related work that has been pursued in the fields of the World Wide Web and Databases in understanding and interpreting tables.

2.1 Wikitology

Wikitology is a hybrid knowledge base (KB) of structured and unstructured information from Wikipedia, which is augmented with structured information from DBpedia, Freebase, WordNet and Yago. Wikitology is comprehensive KB since its base is the English language articles from Wikipedia. Wikipedia is a collaborative encyclopedia developed by web users from around the world. It has around 15 million articles in multiple languages, with more than 3.2 million articles in English [1]. Every Wikipedia article is associated with only one concept (e.g. person, location, organization etc.) and all articles on Wikipedia are organized into categories and sub - categories. All the Wikipedia articles also have links between themselves which leads to the creation of huge graph of concepts on Wikipedia. Most Wikipedia concepts have infobox associated with it, which stores information related to the concept in a semi - structured format. Evaluations have shown that article quality on

[1] http://en.wikipedia.org/wiki/Wikipedia

Wikipedia is pretty high (Hu *et al.* 2007).

Wikitology captures the free text, the concepts graph and the infobox information and integrates this information with the information of the Wikipedia concept from other KB's. Wikitology uses a specialized Lucene (Hatcher & Gospodnetic 2004) IR index to integrate unstructured as well as structured information related to every Wikipedia concept. The IR index enhances and also at the same time simplifies the query process allowing applications and user to make unstructured queries (search using words in a documents), structured queries (search for all *dbpedia-owl:person*) and combination of structured and unstructured terms in the same query (search using words in all articles/concepts of the type *dbpedia-owl:person*).

The version of Wikitology we used uses the information from the March 2008 dump of Wikipedia, DBpedia version 3.4, Freebase dump from 2009 and WordNet version 3.0.

2.2 DBpedia

The DBpedia project aims to extract structured information from Wikipedia. DBpedia makes Wikipedia data available in RDF, so as to allow sophisticated and structured queries over Wikipedia. The DBpedia knowledge base (as of April 2010) contains more than 3.4 million things. DBpedia has its own Ontology to describe data stored in DBpedia. The Ontology has over 259 classes and more than 1200 properties. DBpedia provides access to its data via periodically released dumps or a public SPARQL (Prud'hommeaux & Seaborne 2007) end point[2].

[2]http://dbpedia.org/sparql

2.3 Freebase

Freebase is collaboratively built database of structured data contributed by users and also data collected from many sources such as Wikipedia and MusicBranz. As of 2008, Freebase contained around 125,000,000 tuples, more than 4000 types and more than 7000 properties. Freebase stores its data in a graph based tuple store and provides a public read/write access to it via a HTTP based graph query API using a ”Meta Query Language” (Flanagan 2007).

2.4 WordNet

WordNet is a lexical database of the English Language. Nouns, verbs, adjectives and adverbs are grouped to represent a unique lexical concept. These group of similar words are interlinked via “conceptual semantic” and “lexical” relations, which results in the creation of network of related words and concepts.

2.5 Yago

Yago is knowledge base that includes facts extracted automatically from Wikipedia and WordNet, unified using a combination of rule-based and heuristic methods. Yago contains more than 2 million entities and knows more than 20 million facts about them.

2.6 Related Work

Extracting, interpreting and understanding data from tables is a problem of interest to many areas such as Databases, Web Systems and the Semantic Web.

In the Database domains, understanding tables is of key interest in Data Integration (Ziegler & Dittrich 2004), (Pantel, Philpot, & Hovy 2005). In the area of Web and Web

Systems, tables have received attention from researchers who want to learn techniques of extracting and indexing tables (Cafarella *et al.* 2008) to improve the search experience of the user. Recent work on web tables (Lin *et al.* 2010) also focused on expanding web tables by discovering new relations using hyperlinks present in web tables.

Researchers in the area of the Semantic Web have had particular interests in tabular data because of its semi - structuredness which can be utilized to convert legacy tabular data into RDF (Lassila & Swick 1999). One direction of focus has been Database schema to Ontology mappings. Barrasa et.al (Barrasa, scar Corcho, & Gmez-prez 2004) define database schema to ontology mapping as a set of correspondences (i.e. mapping elements) that relate the vocabulary of a relational DB schema with that of an ontology. The goal of this work is to map or relate DB's tables, columns, primary and foreign keys, etc., with ontology's concepts, relations, attributes etc. Some of the other work in Database schema to Ontology Mappings includes (Hu & Qu 2007), (Papapanagiotou *et al.* 2006), and (Lawrence 2004).

The problem of mapping databases to RDF has received formal attention from the World Wide Web Consortium (W3C). The W3C has formed a working group to create a standard for exposing Relational Databases as RDF. On June 8, 2010 the group published it's first working draft, which captures use-cases and requirements to map Relational Databases to RDF (Auer *et al.* 2010). The eventual goal of the working group is create a specification language for mapping relational databases to RDF and OWL (OWL 2004).

The other direction of research in Semantic Web with respect to tabular data is converting data stored in spreadsheets to RDF. RDF123 (Han *et al.* 2008) is a system that translates spreadsheet data into RDF. RDF123 allows the user to control and define the mappings between spreadsheet data and RDF. Users create a graphical RDF123 template which specifies how each row in the spreadsheet should be mapped into RDF. Each spreadsheet cell can either map either to a RDF node or to a literal value (i.e. a string). Some of the

other systems that map spreadsheets to RDF include (Bizer & Seaborne 2004), (Langegger & Wob 2009), (Golbeck *et al.* 2002).

While systems like RDF123 are useful in mapping data into RDF, the problem with such systems is that it requires human intervention. Users need to define and specify the mappings in which they need to select the appropriate properties or specify RDF terms from an Ontology, which means additional task of searching for ontology as well. Users who are not well - versed with the Semantic Web may find difficulty is using such systems. Finally these systems do not generate data that is linked. The RDF data generated by such systems are just literal strings, instead of resources on the Semantic Web.

Our work focuses on not only generating data in a serialization of RDF, but it also on automatic procedures of linking the data so that it can interpreted as a resource, instead of just a literal string.

Chapter 3

TABLES TO LINKED DATA

The process of converting data stored in tables into linked data is not a trivial one. A table provides evidence in the form of table rows, table columns and table headers. These evidences need to be combined together to get a better interpretation of the table. The table header often describes the type of entities present in the column, a particular column in a table often contain the same type of entities and the table row stores all entities that may be related to one another. Suppose we had a string "*Michael Jordan*" in a table about basketball players. The table header for a column containing the string could be Name, all the values in the column could be names of various basketball players and the row of the table may include other information related to the player such as team, birth place, position etc. Thus using various sources of evidence we can disambiguate and identify the correct *Michael Jordan*, which is the *basketball player* in the given table and not *Michael Jordan*, the *Berkeley professor*.

We break down the process of converting tables into linked data to the following tasks -

- Associate class labels from appropriate ontology for every column
- Linking the table cell values to appropriate entities
- Identifying and discovering new relationships between table columns

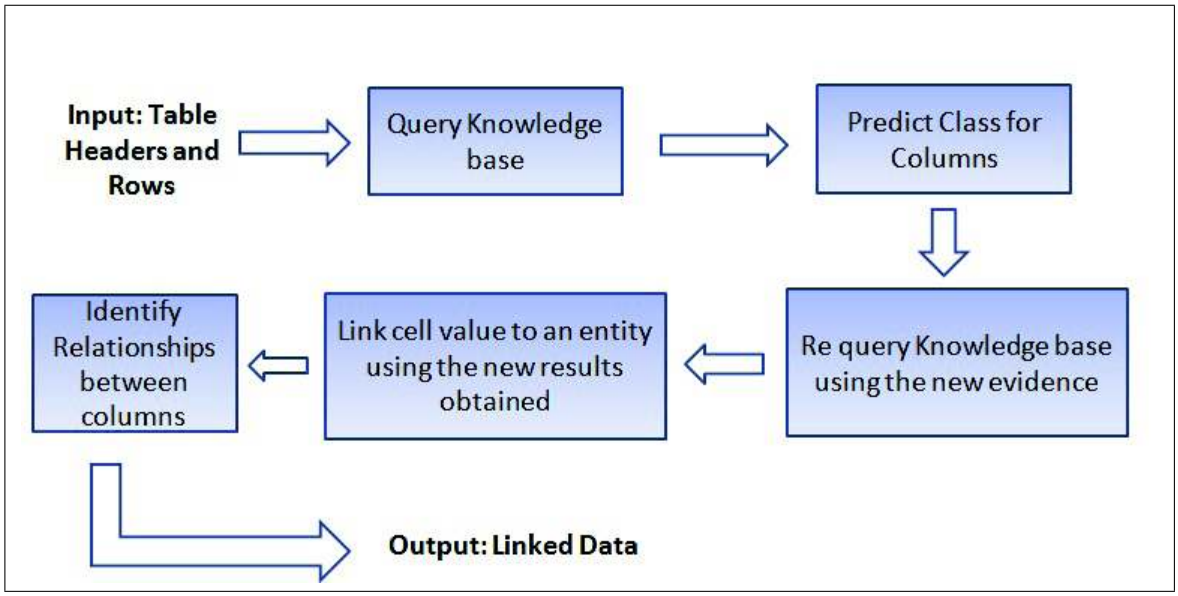

FIG. 3.1. An overview of the approach for converting tabular data to linked data

Figure 3.1 gives a high level overview of the approach adopted. Given an input table, we use table headers, rows and columns as a part of the query to the knowledge base (KB). The results obtained from the KB are processed to predict a class label for every column. Using the predicted class labels as additional evidence we re-query the KB. The results obtained from the KB are used in the process of linking table cells to entities. The linked entities are further used in the process of identifying relations between the various columns in the table. All this information can be published on the Semantic Web in some serialization of RDF. We use Wikitology as a KB since, Wikitology exploits the knowledge from Wikipedia and augments it with the knowledge from other structured KBs' such as DBpedia, Freebase, Wordnet and Yago.

In the following sections, we describe in detail how each task is accomplished. In the final section of this chapter (section 3.5), we describe certain cases which need to be handled in a slightly different way.

3.1 Associating Class Labels with Columns

Algorithm 1 "PredictClassLabel" An algorithm to pick the best class to be associated with a column

1: Let S be the set of k strings in a table column.
2: For each *s* in S, query KB to get a ranked list of top N possible Wikipedia instances along with their types or class labels and their predicted page ranks.
3: From the k $\times$ N instances, generate a set of class labels that can be associated with a column. Let C be this set.
4: Create a matrix V $[c_i, s_j]$ of class label - string pairings where $0 < i <$ size of (C), $0 < j <$ size of (S)
5: Assign a score to each V $[c_i, s_j]$ based on the highest ranking instance that matches c_i. The instance's rank R and its predicted Page Rank is used to assign a weighted score to V $[c_i, s_j]$ (we use w = 0.25):

$$\text{Score} = w \times (1 / R) + (1 \;\; w) \times (\text{Normalized PageRank})$$

6: If none of the instances for a string match the class label being evaluated assign the pair V $[c_i, s_j]$ a score of 0.
7: Choose the class label c_i which maximizes its score over the entire column (S) to be associated with the column.

The class label for every column is determined by the type of entities stored in the column. We develop an algorithm "PredictClassLabel" (see Algorithm 1) to associate a class label with every column. For a given vocabulary and a table column, the algorithm picks the best class to associate with the column. Let "S" be the set of "k" strings in a given column of a table (e.g. S = Baltimore, Boston, New York). For every string in S, we query the KB with using a custom query module described in Section 3.1.1. The KB returns a list of ordered top N instances (with the most relevant at the top, next relevant one in the second place and so on) that the string could be linked to, along with their types or class labels. The results obtained from the KB for the query over "S" would be similar to the ones shown in Figure 3.2.

Once we query for every string in the column, we would end up with k $\times$ N instances and their types from which we create a set of class labels "C" which contains all possible

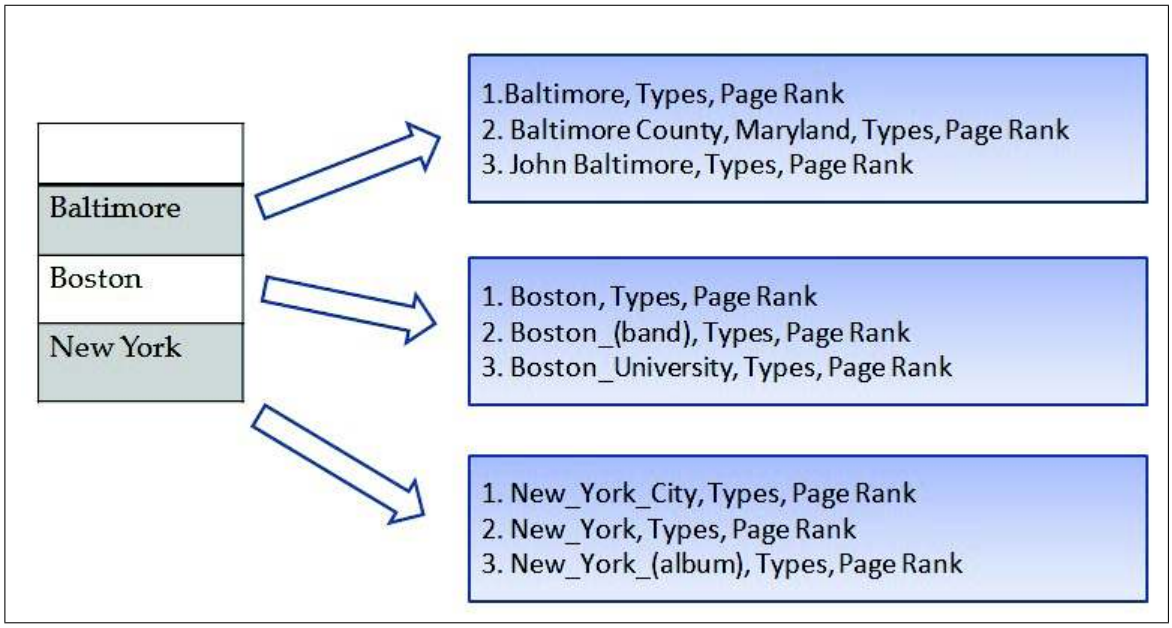

FIG. 3.2. An example of how the results returned by KB could look like

class labels that could be associated with a given column. Such a set "C" for a given column, would look something similar to

C = {dbpedia-owl:Area, dbpedia-owl:populatedPlace, dbpedia-owl:Person, dbpedia-owl:Organiztion, dbpedia-owl:Band, ... }

Each class label in C is paired with every string in S, which creates a matrix V $[c_i, s_j]$ (where $0 < i <$ size of (C) and $0 < j <$ size of (S)). Such a pairing would be similar to the one shown in Figure 3.3.

Each class label - string pair is assigned a score based highest ranked instance whose class label matches the class label being scored. The score is a weighted sum of the instances rank and its predicted page rank as described in Algorithm 1. The page rank used while scoring is normalized, by dividing it by 7 (since the highest page rank to any Wikipedia instance is 7). If there is no instance associated with a given string, whose class label matches the class label being scored, that class label - string pair is assigned a score

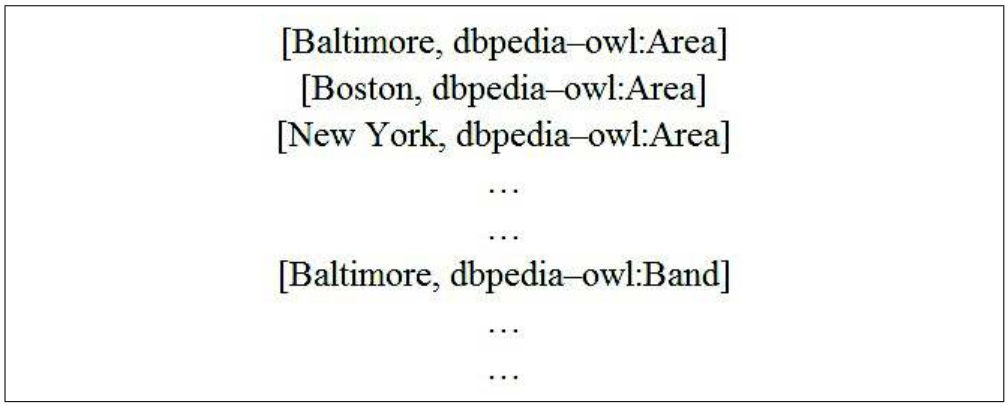

FIG. 3.3. An example of class label and string pairings

of zero. An example of how the scoring would work is shown in Figure 3.4.

Once the entire matrix of class label - string pairings are assigned a score, the class label that maximizes its score over the entire column is chosen to be associated with the column header. The final sum score is also normalized between zero and one by dividing it by number of rows in a given table. A class label is considered as candidate class label for a column only if it has a score higher than a cutoff score. We do so since if a label has less score, it is less likely to be a correct label that can represent all the strings in the column. We use a cutoff score of 0.3. The score is chosen on a heuristic that the class label should get 30 % of the total score. For every column, we associate a class label from four vocabularies - DBpedia Ontology, Freebase, Wordnet and Yago.

3.1.1 Custom Query Module for Wikitology

For every table cell / string in the column, we query Wikitology to determine which instances of the KB the table cell can link to. The interface to Wikitology is via a specialized IR index, which allows structured, unstructured as well as a combination of structured and unstructured queries. Some of the fields in the Wikitology index include title (Wikipedia

Processing - [Baltimore, dbpedia-owl:Area] pair

Results from the query to KB for the String Baltimore:
(R = 1) Baltimore, {dbpedia-owl:Place, dbpedia-owl:Area}, PR = 6
(R = 2) Baltimore County, {dbpedia-owl:Place, dbpedia-owl:Area}, PR = 4
(R = 3) John Baltimore, {yago:AmericanConductors,yago:LivingPeople} PR = 5

Score = (0.25 x 1 / 1) + (0.75 x 6 / 7) = 0.892

Since the 1st ranked instance has a class label that matches the class label being evaluated, R is set to 1 and Page Rank is set to 6

Processing - [Baltimore, dbpediaowl:Band] pair

Results from the query to KB for the String Baltimore:

Results from the query to KB for the String Baltimore:
(R = 1) Baltimore, {dbpedia-owl:Place, dbpedia-owl:Area}, PR = 6
(R = 2) Baltimore County, {dbpedia-owl:Place, dbpedia-owl:Area}, PR = 4
(R = 3) John Baltimore, {yago:AmericanConductors,yago:LivingPeople} PR = 5

Score= 0

Since the class does not match any of the entities for Baltimore, the pair gets a score of 0

FIG. 3.4. An example of how score is assigned to each class label - string pair

concept title), first sentence (first sentence of the Wikipedia page for a concept), contents (contents of the Wikipedia concept page), page rank (approximate page rank for Wikipedia concept (Syed *et al.* 2010)), redirects (redirects associated with the Wikipedia concept), types (Freebase entity types, DBpedia, WordNet and Yago types for the Wikipedia concept), linked concepts (includes all the associated concepts for a given Wikipedia concept), categories (which category the Wikipedia concept belongs to) and property values (contains DBpedia infobox properties and values for a given Wikipedia concept).

We develop a custom query module in which the column header, the table cell string and the row data is mapped to the various fields of the Wikitology index. The table cell string is mapped to title field since the Wikipedia title often includes the name of the concept. The table cell string is also mapped to the redirects field of the index. The table cell along with the column header is also mapped to first sentence field, since the first sentence of a Wikipedia page article often mentions the name of the concept as well as its type. For example the first sentence of the Wikipedia page of Maryland is "The **state** of **Maryland** is an American state ..." which includes both the concept name "Maryland" and its type "state". The column header is also mapped to the types and categories field of the index. The table cell with a boost of 4.0 and the row data is mapped to the contents field and the linked concepts field of the index. The table row values excluding the table cell that is being queried are mapped to the propertiesValues field of the index. All the fields in query have a "or" clause association amongst themselves. Figure 3.5 describes the query.

To get types associated with a Wikipedia concept, we also query DBpedia using its public SPARQL endpoint. Since we query different KB's we also need to handle disparity of the data that may be present in these KB's. For example the same concept maybe be referred by two different names in different KB's - Baltimore is referred to as "Baltimore,_Maryland" in Wikitology whereas DBpedia refers to it as "Baltimore". To overcome such disparity, we use the "redirects" feature of Wikipedia. Every Wikipedia concept

Input: Table Cell Value (String)
Table Row Data (RowData)
Table Column Header (ColumnHeader)

Output: Top "N" matching instances from KB (TopN)

Query = wikiTitle: String (or)
redirects: String (or)
firstSentence: String, ColumnHeader (or)
types: ColumnHeader (or)
categories: ColumnHeader (or)
contents: (String) ^ 4.0, RowData (or)
linkedConcepts: (String) ^ 4.0, RowData (or)
propertiesValues: RowData

FIG. 3.5. Description of the query to Wikitology for a table cell

has redirects page associated with it, which captures all other titles the Wikipedia concept maybe referred to by. For example some of the redirects of Annapolis Maryland are Annapolis,_Maryland, Anapolis,_Maryland, Annapolis, Annapolis,_MD.

Thus to get all types associated with a concept, we also query all the redirects of a concept, to avoid missing out on any data due to disparity between KB's. The SPARQL queries to get redirects from DBpedia and types associated with a concept from DBpedia are described in Figure 3.6. For a given Wikipedia concept, first all the redirects are obtained from DBpedia using the SPARQL query for redirects. For each redirect, we get the types associated with it, using the SPARQL query for types. Finally the types returned by Wikitology and DBpedia are merged to create a set of types associated with the Wikipedia instance.

SPARQL Query to get redirects –

```
PREFIX dbpprop: <http://dbpedia.org/property/>
SELECT * WHERE {< url > dbpprop:redirect ?redirects }
```

where url = Dbpedia url for the concept, dbprop:redirect = Property of a concept that maps redirects to a concept

SPARQL Query to get types –

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {<url> rdf:type ?labels }
```

Where rdf:type = property that maps types to concepts

FIG. 3.6. SPARQL Query to get redirects and types for a given instance

3.2 Linking the Table Cells

Once the class labels for every column are predicted, we proceed to link the table cells to instances / entities from the Wikipedia / DBpedia. We develop an algorithm "LinkTable-Cells" (see Algorithm 2) for linking table cells to entities. The algorithm works as follows. Let "S" be the set of k strings in a given table (e.g. S = Baltimore, Boston, New York, etc.). For every string s in S, we re-query the KB with using a custom query module described in Section 3.2.1. The KB returns a list of ordered top N instances that the string could be linked to, along with their approximate page rank. The results obtained from the KB for the query would be similar to the ones shown in Figure 3.2.

Let I be the set of Instances that can be linked / associated with a table cell string *s*. For every instance *i* in I, we calculate the Levenshtein distance (Levenshtein 1966) between the table cell string *s* and *i*. Since the same string can be referred by several names (for e.g. the basketball team Los Angeles Lakers is referred to by names such as Lakers, L.A.Lakers,

Algorithm 2 "LinkTableCells" - An algorithm to link table cell to entities

Let S be the set of strings in a table.
for all s in S **do**
  Query the KB and get top N instances that the string can be linked to. Let I be this set of instances for the string.
  for all i in I **do**
    Get all the other names associated with i. Let this set be O
    Calculate the Levenshtein distance between s and all $o \in$ O
    Choose the best (smallest) Levenshtein distance between s and any $o \in$ O
    Similarly calculate the Dice score between s and all $o \in$ O
    Choose the best (largest) Dice score
    Create a feature vector for i. The vector includes the following features: $i's$ Wikitology index score, $i's$ page rank, $i's$ page length, best Levenshtein distance and best Dice score
  end for
  Input feature vectors of all $i \in$ I to a SVM Rank Classifier. The Classifier outputs a ranked list of instances in I
  Select the instance which is top ranked. Let it be top_i
  To feature vector of top_i, append two new features - the SVM Rank score for the top_i and the difference of scores between the top two instances ranked by SVM Rank
  Input this vector to another classifier which produces a label "yes" or "no" for the given vector
  If the classifier labels the feature vector a "yes", link the string s to instance top_i else Link it to NIL.
end for

Los Angeles Lakers etc.), we get possible other names that i can be referred with. We describe the process of getting other names for *i* in Section 3.2.2. Let O be the set of other names for *i*. We calculate the Levenshtein distance between the table cell string *s* and all *o* $\in$ O. Smaller the Levenshtein distance means, the strings maybe more similar, zero being an exact match. We choose the best (i.e. smallest) Levenshtein distance between table cell string and *o*. In a similar manner, we also calculate the Dice Score (Salton & Mcgill 1986) between the table cell string *s* and all *o* $\in$ O. The score we get back is a number between 0 and 1. If the Dice Score is 1 or tending to 1, means that the two strings are similar. We choose the best (i.e. largest) Dice Score.

Along with the Levenshtein distance, Dice Score, we get the instance *i*'s Wikitology Index score, *i*'s page length and its approximate page rank. All these values become a part of a feature vector f (i). For all $i \in$ I, we generate an f (i) to get a set of feature vectors F. We input F to a classifier that returns a ranked list of items in F. Since for every string *s* we have a set of Instances I from which we have to choose the best *i*, we think it is appropriate to build a classifier that can learn to rank instances within a given set. We describe how the classifier is trained in section 3.2.3.

We select the instance *i* which gets the top rank assigned by the classifier.To the feature vector of the selected *i*, f (i) we append two more features - the score assigned to f (i) by the previous classifier and the difference between the scores of the top two instances in the ranked list generated the previous classifier. This new feature vector f'(i) is passed to another SVM classifier which produces a label "yes" or "no". "Yes" indicates that the table string *s* should be linked to *i* and "no" indicates that *s* should be not linked to *i* and instead *s* should be linked to "NIL". "NIL" indicates that the KB has no knowledge or information about the entity *s*. The purpose of the second classifier is to determine whether the table cell string s should be linked to the top ranked instance *i* or not. We describe how we train this classifier in section 3.2.4.

Input: Table Cell Value (String)
Table Row Data (RowData)
Table Column Header (ColumnHeader)
Predicted Class Labels for Columns (ClassLabels)

Output: Top "N" matching instances from KB (TopN)

Query = wikiTitle: String (or)
redirects: String (or)
firstSentence: String, ColumnHeader (or)
types: ColumnHeader (or)
categories: ColumnHeader (or)
contents: (String) ^ 4.0, RowData (or)
linkedConcepts: (String) ^ 4.0, RowData (or)
propertiesValues: RowData (**and**)
typesRefValues: ClassLabels

FIG. 3.7. Description of the re-query to Wikitology for a table cell

3.2.1 Re-querying Wikitology

For every string *s* in S, we re-query our KB - Wikitology to determine the top N possible instances that *s* can be linked with. An important addition to the re-query module as compared to the initial query (section 3.1.1) is the use of class labels. We use the class labels predicted for every column as additional evidence in the re-query. The modified query is described in Figure 3.7. Besides the fields that were present in the previous query (Figure 3.5), we include the following additional field in the query:

- typesRef - typesRef is an exact match field for the field "types" from the Wikitology index. The class labels of a column are mapped to the typesRef field. By including the typesRef, we restrict the types of instances returned by the KB to types described in the class label

SPARQL Query to get other names –

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * WHERE { < url> rdfs:label ?labels }
```

FIG. 3.8. SPARQL Query to get other names for a given instance

While all the other fields in query have a "*or*" clause amongst themselves, the typesRef is "*anded*" with the rest of the fields in the query, which means the instance returned by the KB should have one of types or class labels from the predicted class labels for the column which the table string *s* belongs to.

3.2.2 Commonly Used Names for Entities

The same string (entity) can be referred by several names. For e.g. the basketball team Los Angeles Lakers is referred to by names such as Lakers, L.A.Lakers, Los Angeles Lakers etc. Thus to get the most accurate values for the Levenshtein distance and the Dice score between a table cell string *s* and instance *i*, we gather all possible names, the instance is referred to by. We obtain this information from DBpedia. Every instance in DBpedia has a "rdfs:label" property associated with it, which maps the instance to other names, the instance can be referred to by.

We map the Wikitology instance to the DBpedia instance. To avoid the problem of disparity of data between the two KB's, we first get all the redirects associated with the instance. Then for every redirect for the instance, we query the public SPARQL end point of DBpedia to get all the names the instance can be referred by. The SPARQL query for redirects is described in Figure 3.6 and the SPARQL query for getting others names (via rdfs:label) is described in Figure 3.8.

3.2.3 Learning to Rank

For every string *s* in S, we get a set of Top N instances (I) to which *s* can be linked to. Since we have to pick the best and the most appropriate one to link, we decide to use a machine learning algorithm, which can learn to learn to rank instances within a given set. We use SVM^{rank} [2] algorithm of Joachims (Joachims 2006) for this purpose. SVM^{rank} takes a feature set of *n* instances as input and returns a ranked list of instances back as output.

We build the feature vector based on measures that can be broadly categorized into two categories - similarity measures and popularity measures. For an instance *i* from the KB to be linked to the table cell string *s*, it must be similar to *s* and it must be a popular entity within the KB. We include the similarity measures as a part of the feature vector, since the table cell string and the instance in the KB are likely to have same or similar names. And in situations where there are many good candidate entities to which a string can be linked to, its' often the case that the more popular or well-known candidate entity is often the correct answer. Hence we include the popularity measures in the feature vector as well.

The similarity measures included in the Feature vector are - the Levenshtein distance and the Dice Score. The popularity measures included in the Feature vector are - the instance's Wikitology index score, its approximate page rank and its Wikipedia article page length. Instead of using the actual page length, we use a normalized page length. We normalize the page length, by taking a log to the base 10 of the page length.

[2] http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html

3.2.4 "*To Link or Not to link, that is the question*"

We train a second classifier to decide whether to link the table cell string to the top ranked entity returned by SVM^{rank} or not. There can be cases where all the Top N instances returned by the KB maybe the incorrect ones to link. Such a case is very likely when the table cell string we are querying, may be a new entity and the KB has no information or knowledge about the it.

We use a binary SVM classifier SMO provided in Weka (Hall *et al.* 2009) which takes a feature vector as input and produces a label "yes" or "no" as output. The label "yes" indicates that the table cell string should be linked to the top ranked instance and the label "no" indicates that it should be linked to "NIL". "NIL" is similar to choosing an "None of the above" option.

Two new features are added to the existing feature vector - the score assigned to top ranked instance by SVM^{rank} and the difference in score between the top two instances in the list generated by SVM^{rank}. We choose these features since if the top ranked instance is a correct choice then it gets a much higher score as compared to other instances in the list.

3.3 Identifying Relationships between columns

We develop an algorithm "IdentifyColumnRelations" (see Algrothm 3). Given two columns whose strings are linked to Wikipedia / DBpedia entities, it identifies the best possible relation between those two columns. The algorithm works as follows. Let C_i and C_j be set of *k* strings each (whose values are linked) in any two given columns I and J. Let every k^{th} string from C_i and C_j be $s_{k,i}$, $s_{k,j}$ respectively. For every k^{th} string in both the columns we query DBpedia to identify relations between the two strings $s_{k,i}$ and $s_{k,j}$. We describe the query process in section 3.3.1. Let the set of relation between every such $s_{k,i}$

Algorithm 3 "IdentifyColumnRelations" - An algorithm to identify relations between two columns

1: Let C_i and C_j be sets of *k* strings each (which are linked to entities from DBpedia) in columns I and J
2: For every k^{th} string ($s_{k,i}$, $s_{k,j}$) in C_i and C_j, query DBpedia to get set of relations between $s_{k,i}$, $s_{k,j}$. Let that set be R_k
3: Generate a set of candidate relations CR between C_i and C_j from the relation sets R_k between each of the k^{th} strings in C_i and C_j
4: **for all** r in CR **do**
5: **for** $k = 1$ to $sizeOf(C_i)$ **do**
6: **if** $r \in R_k$ **then**
7: score = score + 1
8: **end if**
9: **end for**
10: Normalize the score: score = score / number of rows in the table
11: **end for**
12: Choose the relation r from CR that gets the highest score

and $s_{k,j}$ be R_k.

Once we identify relations between all the pair of strings between the two columns, we generate a set of candidate relations CR from the set of relations R_k between each $s_{k,i}$ and $s_{k,j}$. Each candidate relation in CR is assigned a score. For every relation *r* in CR, we check if *r* appears in R_k (where $0 <$ k $<$ number of rows i.e. number elements in columns C_i *or* C_j). If *r* appears R_k, then score of *r* is incremented by 1. Finally the sum score of *r* is normalized to a number between zero and one, by dividing the sum score by number of rows. Once all the relations in CR are assigned a score, the relation with the best (highest score) is chosen as the relation between the columns I and J.

Once the relationships are identified between two columns, the next step is figure out which relations would best describe the table as a whole. There might be many relations that may exist between various columns in a table. For example if we had a table with four columns, column one might be related with columns 2, 3 and 4; column 2 with columns 1 and 4 and column 3 with columns 1 and 4 as well. Not all relations might be relevant and

useful in interpreting and describing the table overall, thus we need to figure out more the relevant relations.

We address this issue via a heuristic based approach. Every table has a primary column or a significant column, which can uniquely identify each row of the table. For example if we go to the table in Figure 1.1, column one “City” would a primary column / significant column of the table. We choose all relations between the primary column and any other in the table column with which the primary column is related.

Our definition of a “primary column” is as follows - The column that has relation with most other columns in the table is chosen as the primary column. If we go back to the table in Figure 1.1, the column “City” is related with most other columns in table. City is related with State, Mayor and Population. Thus the Column City would be chosen as the primary column. In the example mentioned above of table with four columns, column 1 would be chosen as the primary column.

Another heuristic that can be applied to choose a primary column, would be choosing the left most column as the primary column, since generally in most tables, the left - most column is the more significant or important column around which the table is structured. We recognize that our current work and approach of selecting the best possible relations to describe and interpret the table overall is pretty preliminary. We discuss some of the issues and other possible approaches in the Future Work chapter.

3.3.1 Querying DBpedia to identify relations

To identify relations between two strings we query the DBpedia. Since the strings are linked at this stage, we can query the public SPARQL endpoint of DBpedia using their URIs. The data in DBpedia is stored in form of the following triple - “Subject : Predicate / (Property) : Object”. When a SPARQL query is made over DBpedia data, it also needs to follow the same organization. The Subject in a SPARQL query needs to be URI and

the object can be either a literal string or a URI. The predicate can be any of the properties present between any two instances of DBpedia (e.g. rdfs:label, rdf:type etc). The predicate is our case is the relation we want to identify between the two instances.

When we query DBpedia for relations between two instances, we need to consider the following things -

1. Properties and relations have a direction. We can obtain one relation when string in column I is the subject and another one when the string in Column J is the subject. For example the relation between George Bush (as subject) and Barack Obama (as object) would be "successor" whereas the relation between Barack Obama (as subject) and George Bush (as object) would be "predecessor".

2. There is disparity within DBpedia itself on how data is represented. Given two instances and a relation between them, the object in the relation is sometimes is represented as URI and in some other cases is represented as a literal string. For example the relation largest city between Baltimore and Maryland is represented as <http://dbpedia.org/Maryland> dbpprop:largestCity <http://dbpedia.org/Baltimore>, whereas the relation team between Kobe Bryant and Los Angeles Lakers is represented as <http://dbpedia.org/Kobe_Bryant> dbpprop:team "Los Angeles Lakers". Even though a DBpedia page for Los Angeles Lakers exists, it is not linked.

Taking these factors into account, we use the following approach when we query DBpedia. As mentioned before, since there is disparity between the data stored in different KBs (viz. Wikitology and DBpedia), we first get the set of redirects for both the strings ($s_{k,i}$ and $s_{k,j}$). The SPARQL query for redirects is the same as described in Figure 3.6. For each of the redirects for $s_{k,i}$ and $s_{k,j}$, we query DBpedia to obtain the relations between both the strings as described in Figure 3.9.

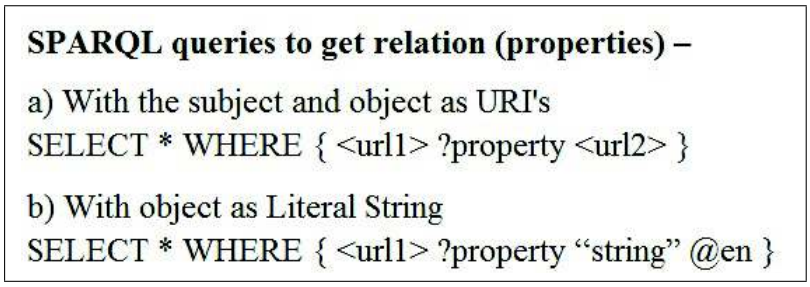

FIG. 3.9. SPARQL Query to get relations / properties between two instances

We make two types of queries. The first query is when the string $s_{k,j}$ is referred with its URI in the "object" part of the query. In the second query, the string $s_{k,j}$ is referred as a literal string in the "object" part of the query. As mentioned before, a string may have many other names that it can be referred with, hence we get all other names that the string $s_{k,j}$ can be referred to by and repeat the second query by using all possible other names as literal string in "object" part of the query.

3.4 Representing Tables as linked data

Based on the data generated in the previous steps, we develop a template which can be used to generate a linked data representation of table. The template is described in Figure 3.10. We generate the output in Notation 3 or N3 (Berners-Lee & Connolly 2008), a serialization of RDF. We decided to use the N3 representation since it is much more compact and human readable as compared to RDF.

Every column header is associated with a class label from an appropriate ontology. We use the "rdfs:label" property from the RDF Schema (Brickley & Guha 2004) to associate the column header string with the class label. For example R rdfs:label L states that L is a human readable label (or common name) for the resource R.

A template in N3 format:
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
"*ColumnHeader1*" is rdfs:label of *PredictedClassLabel1* .
"*ColumnHeader2*" is rdfs:label of *PredictedClassLabel2* .
"*TableCellString*" is rdfs:label of *CellValueURL* .
CellValueURL a *PredictedClassLabel* .
property rdfs:domain *PredictedClassLabel1* .
property rdfs:range *PredictedClassLabel2* .

Where:
ColumnHeader - is a column header from the table
TableCellString - is a string representing a table cell
PredictedClassLabel - is the class label associated with the column
CellValueURL - is the DBpedia url, the table cell string is linked to
property - is the relation discovered between the two columns

FIG. 3.10. A template for representing tables as linked data in N3

Every table cell is also linked to an entity from DBpedia. We also use the "rdfs:label" property to associate a table cell string with the url. The type of table cell string is described by the predicted class label for the column of the string. We use the "rdf:type" property to capture this information in the output. N3 notation allows us to make use of "a" instead of rdf:type. Thus the statements :Bob a :Person and :Bob rdf:type :Person are equivalent. We make use of "a" for rdf:type property in our template. To describe the relation identified between two columns, we use the the "rdfs:domain" and "rdfs:range" property. P rdfs:domain C states that the subject of a triple having predicate as P, can be an instance of class C. P rdfs:range C states that the object of a triple having predicate as P, can be an instance of class C.

The template in Figure 3.10 uses variables ClassLabelPredicted, TableCellString, CellValueURL and property. Replacing these variables with actual strings from a table and the

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dbpedia: <http://dbpedia.org/resource/> .
@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .
@prefix dbpprop: <http://dbpedia.org/property/> .

“City”@en is rdfs:label of dbpedia-owl:City .
“State”@en is rdfs:label of dbpedia-owl:AdminstrativeRegion .

“Baltimore”@en is rdfs:label of dbpedia:Baltimore .
dbpedia:Baltimore a dbpedia-owl:City .
“MD”@en is rdfs:label of dbpedia:Maryland .
dbpedia:Maryland a dbpedia-owl:AdministrativeRegion .

dbpprop:LargestCity rdfs:domain dbpedia-owl:AdminstrativeRegion .
dbpprop:LargestCity rdfs:range dbpedia-owl:City .
```

FIG. 3.11. A example of how the template could be used for representing tables as linked data in N3

predicted class labels, urls will generate a linked data representation of the table. Figure 3.11 describes an example with actual values.

Algorithm 4 can be used to generate a linked data representation of the table as N3. Given a table, along with its predicted class labels for a column, table cells linked and relation between columns discovered, Algorithm 4 iteratively uses the template in Figure 3.10 to generate linked data representation for a given table.

3.5 Special Cases

3.5.1 Column containing Numbers

The approaches discussed so far work well when the values in the given column are strings. But there are columns in tables that contains numbers - population (see Figure 1.1),

Algorithm 4 "GenerateLinkedData" - An algorithm to represent tabular data in N3

1: Let C be the set of Column Headers of the table and PCL be the set of Predicted Class labels for column headers.
2: Let S be the set of Strings from a table which can be linked and U be the set of the urls to which the strings are linked to .
3: Let getColumn be a method which returns a column header, given a string.
4: Let R be a method that returns a relation between the columns in the table. R (C1, C2) will return the relation between columns C1 and C2.
5: Insert the appropriate prefixes in the N3 file.
6: **for all** c in C **do**
7: Insert the following triple in the file:"c"@en is rdfs:label of PCL(c) .
8: **end for**
9: **for all** s in S **do**
10: Insert the following triple in the file:
"s"@en is rdfs:label of U(s) .
U(s) a PLC(getColumn(s)) .
11: **end for**
12: **for all** c_i,c_j in R **do**
13: Insert the following triples in the file:
R(c_i,c_j) rdfs:domain PLC(c_i) .
R(c_i,c_j) rdfs:range PLC(c_j) .
14: **end for**

telephone numbers, social security numbers etc. We need to develop different mechanism to handle columns that contain numbers.

Numbers are not entities that can be linked to entities of a KB. In fact, numbers are generally values of some property associated with entities in the KB. For example if we go back to the table in Figure 1.1, the values in the column "Population" are values of the property "population" and the property population is associated with the strings in column one "City".

We propose the following approach to handle columns that contains numbers. Once it is identified that a column contains numbers, we try to detect what kind of numbers the column stores (for example telephone numbers, SSN etc.). Now consider the first row of the table. For every string in row, we query DBpedia to check if any property is returned that has the number (or its near approximate) as its value for that property.

Once we identify the string whose property has a value from the column that contains numbers, we check if all other strings in the same column also have the same the property for the respective numbers in the column that contain numbers, the values in the column of numbers can be linked to the strings as values of the identified property.

3.5.2 Column containing Acronyms

Another case which we need to handle differently is acronyms. Columns can consist of acronyms of states, stock tickers, organizations, measurement units etc. Although these are strings and can be queried for, a better approach would be detecting that the column contains acronyms and expanding the acronyms during the query process.

Acronyms are generally small and are of three to five characters in length. We can check the length of all the strings in a given column. If the length of all the strings is five or less, we can consider that column for further acronym processing. Once we determine that the column may contain acronyms, we can query the strings of the columns against well-

known acronyms databases to determine what they stand for. Replacing acronyms with their expanded forms will enhance the class label prediction, entity linking and relation identification process.

Chapter 4

EVALUATION

We developed a prototype to test the validity of the approach described in Chapter 3. We describe our data set in section 4.1, our evaluation for class label prediction for columns in section 4.2, our evaluation for linking table cells in section 4.3 and evaluation for column relations in section 4.4.

4.1 Data Set

Our test data consists of fifteen tables obtained from Google Squared, Wikipedia and data set of tables shared by Google researchers. The table topic and their columns are described in Figure 4.1 and a summary of the data set is presented in Figure 4.2. Overall the fifteen tables have 199 rows, 56 columns and 639 entities. However four tables included one column each which had numbers in them. In our current evaluation we excluded columns which contain numbers, leaving 52 columns for assigning class labels to and 611 entities to be linked.

We also classified the columns and the entities in the data set into four categories - Person, Place, Organization and Others. The distribution of entities across these four categories is shown in Figure 4.3 and the distribution of columns across these four categories is shown in Figure 4.4. 45 % of the entities are Places, 20 % are Person, 10 % are Orga-

Table Topic	Columns	Source
US States	State, Capital City, Largest City,Governor	Google Squared
US Presidents	President, Party, Vice President,Preceded by, Succeeded by	Google Squared
Basketball players	Name, College, Team, Nationality,Place of Birth	Google Squared
US Cities	City, State, Mayor, Population	Google Squared
Rivers in Europe	River, Length, Basin Countries,Mouth, Origin	Google Squared
Fortune 500 Companies	Item Name, Industry, Ceo,Headquarters, Type	Google Squared
James Bond Movies	Item Name, Director, Country, Cinematographer, Preceded By	Google Squared
American Films of 2009	Title, Director, Genre, Notes	Wikipedia
American Music Awards of 2009	Artist, Song	Wikipedia
Foreign Born United States Politicians	Name, Country of Birth, Positions Held	Wikipedia
Hospitals in Kenya	Name, Locale, Opened	Wikipedia
National Capitals	City, Country	Wikipedia
Technology Companies	Company, Location, Founded, Industry	Wikipedia
Country Capital	Country, Capital	Google Dataset
Country Currency	Country, Currency, Currency Code	Google Dataset

FIG. 4.1. Table topics, their columns and source

Number of Tables	15
Total Number of Rows	199
Total Number of Columns	56 (52)
Total Number of Entities to be Linked	639 (611)

FIG. 4.2. Summary of the data set used in Evaluation

nizations and 25 % are others (like Films, Songs, Nationality etc.). 40 % of the columns contained Places, 25 % contained Persons, 12 % contained Organizations and 23 % contained other types of data.

4.2 Evaluation for Class label prediction for Columns

The evaluation for class label prediction for columns was bit tricky since labeled data did not exist. We did not have tables with column headers associated with class labels from appropriate ontologies. Hence we decided to use human judgment to test how we fare in this task. We evaluate our class labels predicted from the DBpedia ontology only, since it would be fairly easy for our human judges to browse the DBpedia ontology if they wished to, when they are evaluating. All the three human judges were graduate students in Computer Science who have completed a graduate level course on Artificial Intelligence; however they had no background in the Semantic Web and linked data.

For each of the 52 columns, our system generates a set of possible classes, ranks each class in the set and picks the top one. In the first evaluation, we compare the ranked list that we generate against a "gold standard" ranked list.

To generate a "gold standard" ranked list, we asked three human judges to rank each of the class labels in set generated for every column. We asked them to rank the most relevant as 1, the next relevant one as 2 and so on. The complete evaluation guideline given

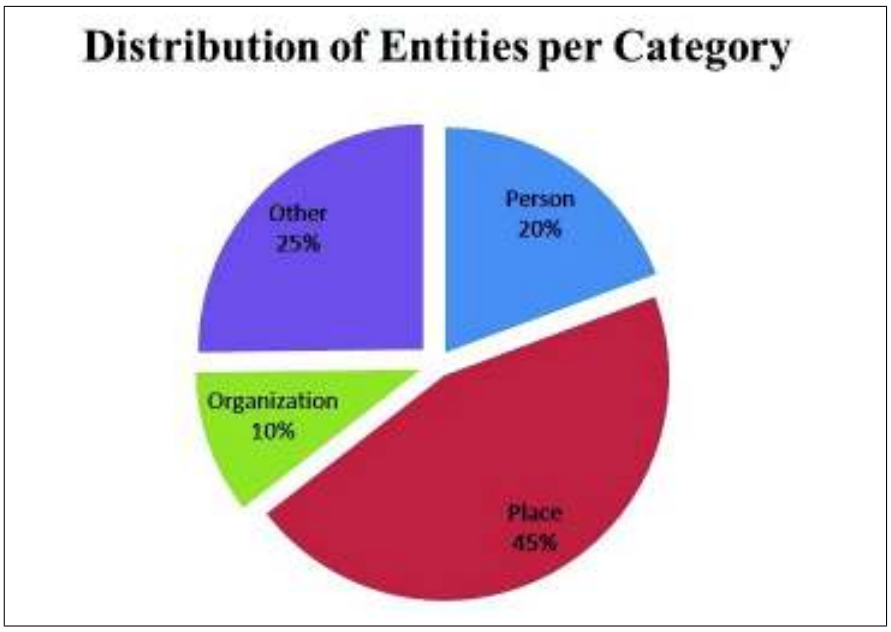

FIG. 4.3. The percentage of entities in each of the four categories

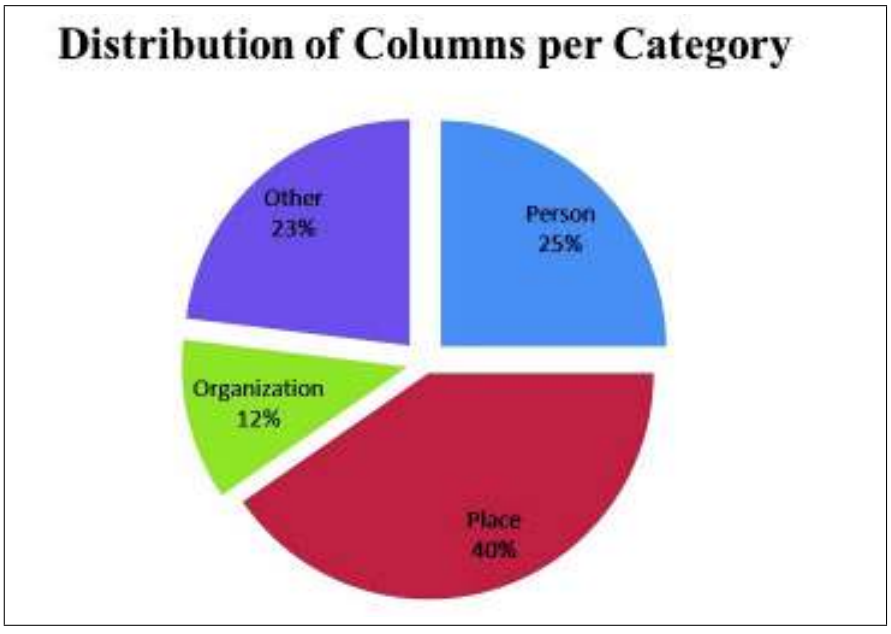

FIG. 4.4. The percentage of columns in each of the four categories

to the human judges is specified in Appendix A.1. Since there were three judges, each class label in a set got three ranks (or scores); we calculated the average rank for each class label and sorted the set of class labels based on average rank. The class label with the smallest average rank was ranked 1; the second one was ranked 2 and so on.

We use Mean Average Precision (Manning, Raghavan, & Schütze 2008) to compare the ranked list generated by the system against the gold standard. Given a query q, the Mean Average Precision is defined as follows -

$$\mathrm{AveragePrecision(q)} = \frac{\sum_{n=1}^{N} \mathrm{P(n)} \times \mathrm{R(n)}}{\text{total number of relevant labels for q}}$$

Where:

R(n) - is the relevance at n. If the class label ranked "n" in the system generated set is a relevant one then R(n) is 1,else it is 0.

P(n) - is the precision at n. It measures the relevance of the top n results.

$$\mathrm{P(n)} = \frac{\text{Number of relevant class labels of rank n or less}}{\mathrm{N}}$$

N - is the number of labels retrieved. For our evaluation we consider the top 3 labels retrieved.

We define ***relevance*** as follows - For each column, we consider the top three ranked labels by the users as the relevant class labels for that column. Thus we restrict the total number of relevant class labels per column to 3. For a given column any label which is not in the top 3 ranked labels is considered as not relevant.

Mean average precision is useful in comparing two ranked sets. It's commonly used in the Information Retrieval domain to compare two sets of ranked results. Mean average precision captures the ordering of the documents returned by a query, giving importance to higher ranked documents. If the ordering of documents in both the ranked sets is same or similar, then Mean Average Precision will be 1 or will tend towards 1. It will be on the lower side of 1 (tending to zero), if the ordering of the elements does not match.

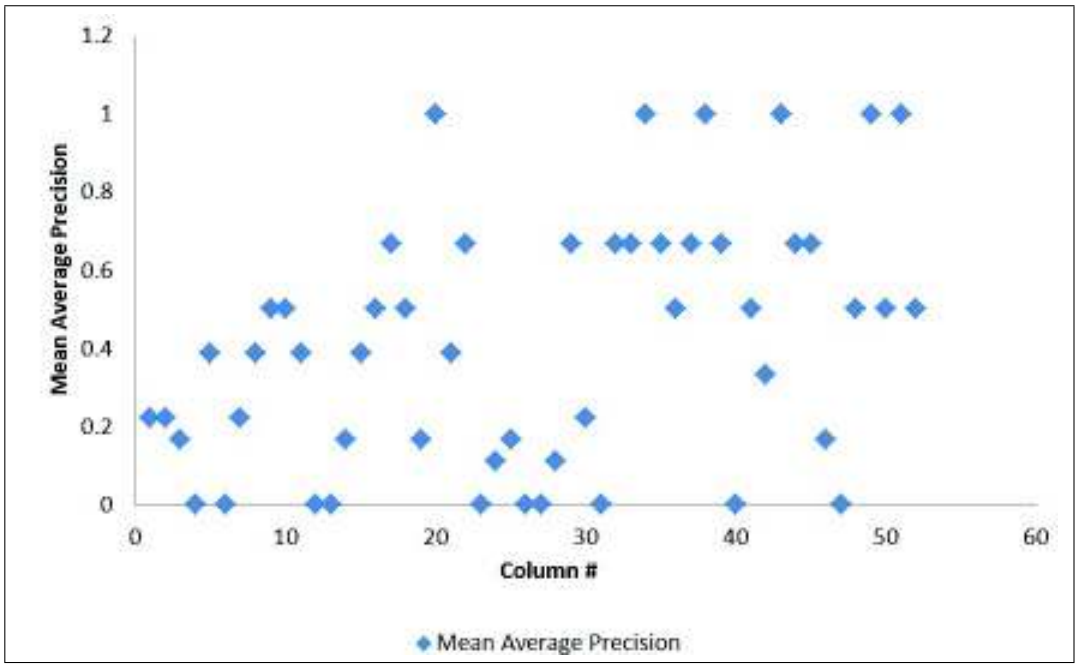

FIG. 4.5. Distribution of Mean Average Precision for table columns

The distribution of mean average precision between the system generated ranked set of labels and the gold standard ranked list is given in Figure 4.5. Our algorithm fares moderately as compared to human judgment; for **80.75 %** of the columns, the MAP between the system generated ranked set of labels and the gold standard ranked list is greater than 0. A MAP greater than 0 indicates that there is at least one relevant label in the relevant ranked position in the system generated list.

We also calculated recall between systems' generated ranked set of labels and gold standard ranked list of labels. This measure helps us to check whether our algorithm is producing relevant set of labels for every column or not. We define Recall (R) as -

$$\mathrm{R} = \frac{\text{Number of relevant labels retrieved}}{\text{Total Number of relevant labels}}$$

The definition of relevance remained the same; for each column, we consider the top

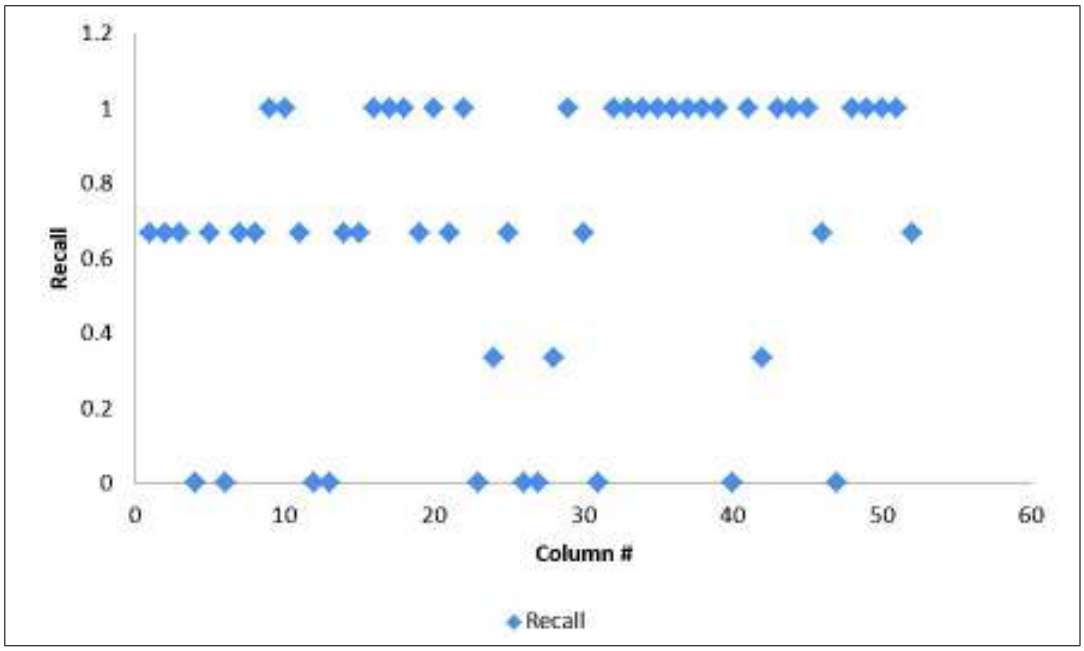

FIG. 4.6. Distribution of Recall values for table columns

three ranked labels by the users in the gold standard as the relevant class labels for that column, thus restricting the total number of relevant class labels per column to 3.

The recall between the system generated ranked set of labels and the gold standard ranked list is given in Figure 4.6. Our algorithm does fairly well predicting relevant class labels in the top 3 ranks. For **75 %** of the columns, Recall between the system generated ranked set of labels and the gold standard ranked list is **0.6** or greater.

We also wanted to compare how frequently system produced top three ranked label matches against the top three of the gold standard. Figure 4.7 presents this comparison. 15.38 % of the times the system's top 1 and 2 matched with the gold standard's top 1 and 2; 23.08 % of the times, system's rank 3 label matched with the gold standard's ranked 3 label.

In our final evaluation for class label prediction, we went back to our three human

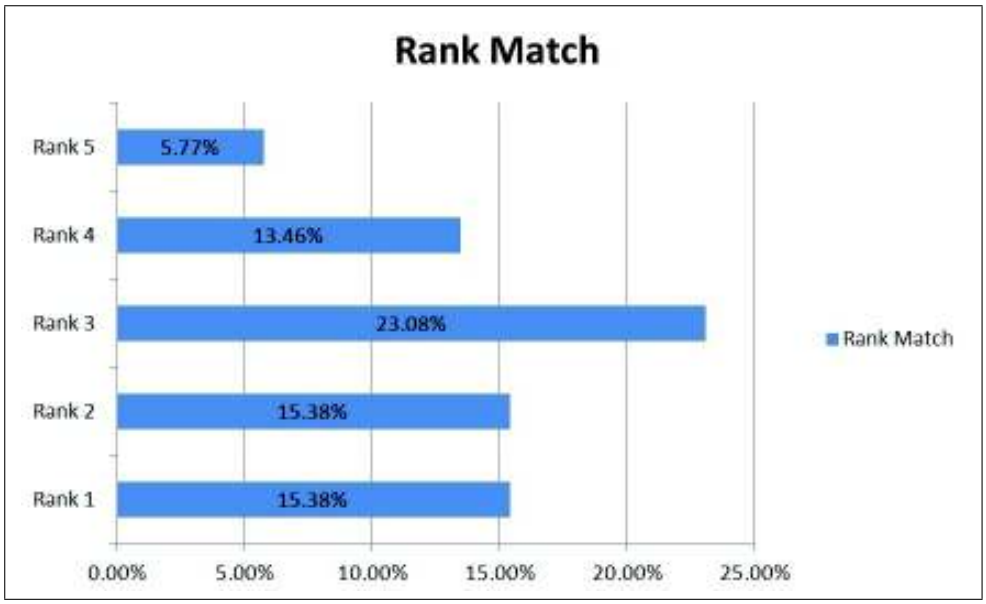

FIG. 4.7. A comparision of how many times the top 3 ranked labels match with the top 3 ranked labels from the gold standard list

judges. In this evaluation the human judges were asked to evaluate our predicted labels for a given column. The human judges were presented a "yes / no" question for every column. For a given column, the judges were asked whether the predicted label was a correct one or not. The evaluation guideline (see Appendix A.2) specified that even though a more accurate label may exist for a given column, the judges needed to decide whether the predicted label is fair and a correct one for a column. The prediction for a class label was considered to be correct if two out of the three judges agreed that it was a correct and fair prediction.

The results of the "correctness evaluation" are presented in Figure 4.8. On an overall basis, the judges considered 40 out of the 52 class labels predicted to be correct giving us an accuracy of **76.92 %**. We also checked the accuracy within the four categories in which the columns were distributed. We obtained the highest accuracy in identifying columns that

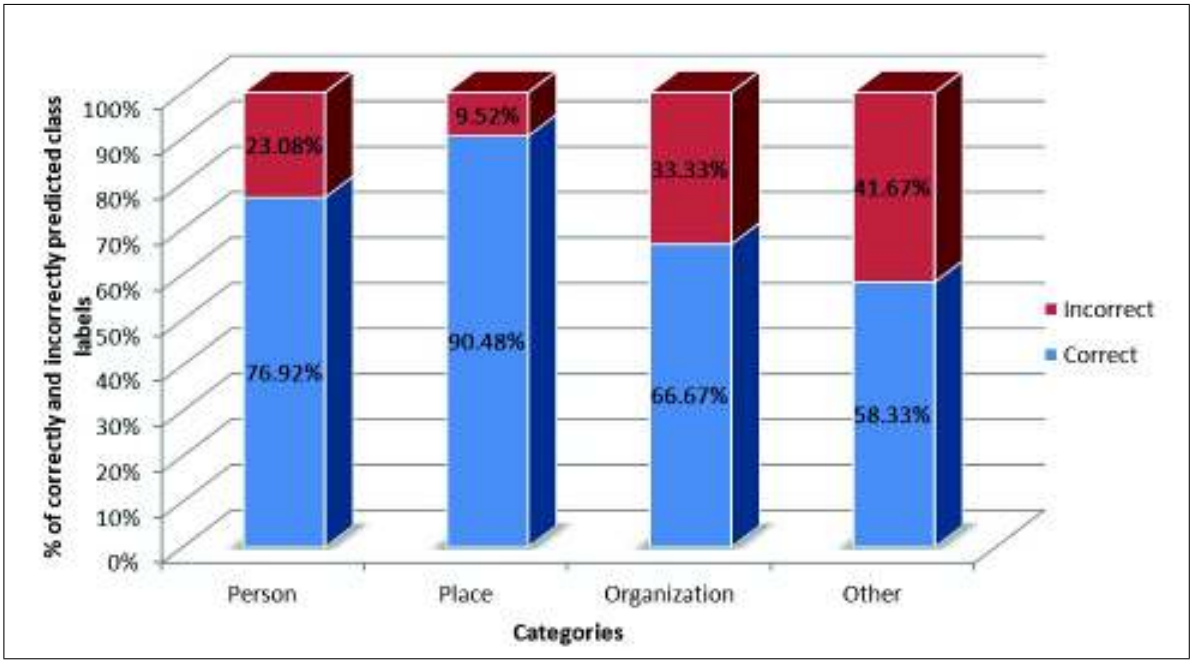

FIG. 4.8. A category-wise breakdown for class label correctness

contained Places (**90.48 %**), followed by Persons (**76.92 %**). However we had moderate success in identifying columns which contained Organizations (**66.67 %**) and other types of data (**58.33 %**) (movies, songs etc.).

To conclude the evaluation for class label predictions, our evaluation measures (recall and the "correctness evaluation") show that our approach is indeed producing relevant and correct labels, but the other measures (Mean Average Precision and Rank Comparison) show that we are enjoyed moderate success in ranking the labels within the predicted set well and predicting the most accurate label as class label for a column.

4.3 Evaluation for linking table cells

For the evaluation of linking table cells to entities, we manually hand labeled the 611 table cells to their appropriate Wikipedia / DBpedia pages. We compared the system generated link against the expected link for each table cell.

Our results for linking table cell to entities are presented in Figure 4.9. Overall we were able to correctly link **66.12 %** of the table cell strings to the appropriate entity from Wikipedia / DBpedia. A look at the breakdown of accuracy based on the categories (see Figure 4.9 shows that we had the highest accuracy in linking Persons (**83.05 %**) followed by linking Places (**80.43 %**). We have moderate success in linking Organization (**61.90 %**), but we fare poorly in linking other types of data like movies, nationality, songs, types of business and industry etc. with an accuracy of just **29.22 %**.

In cases where the KB had no knowledge about an entity, our algorithm is supposed to link such an entity to "NIL". Our dataset had 24 entities and in all the 24 cases, we were able to predict correctly that the table cell should be linked to "NIL". As a preliminary evaluation, we can say that our approach works well, in cases when KB has no knowledge about the instance.

Comparing the entity linking results against our previous work where we used a heuristic based method for linking table cells to entity (Syed *et al.* 2010), we have improved our accuracy in linking places by a good margin (18.79 %) and accuracy for linking persons and organizations slightly decreased (by 7.71 % and 4.77 % respectively). However this would not be that fair a comparison since the initial results from the previous method are for a small subset[1] of the current data set.

In the following two subsections, we discuss the performance of our two classifiers - the first one which learns to rank entities within a set (SVM^{rank} classifier) and the second

[1]The subset had 171 entities to link

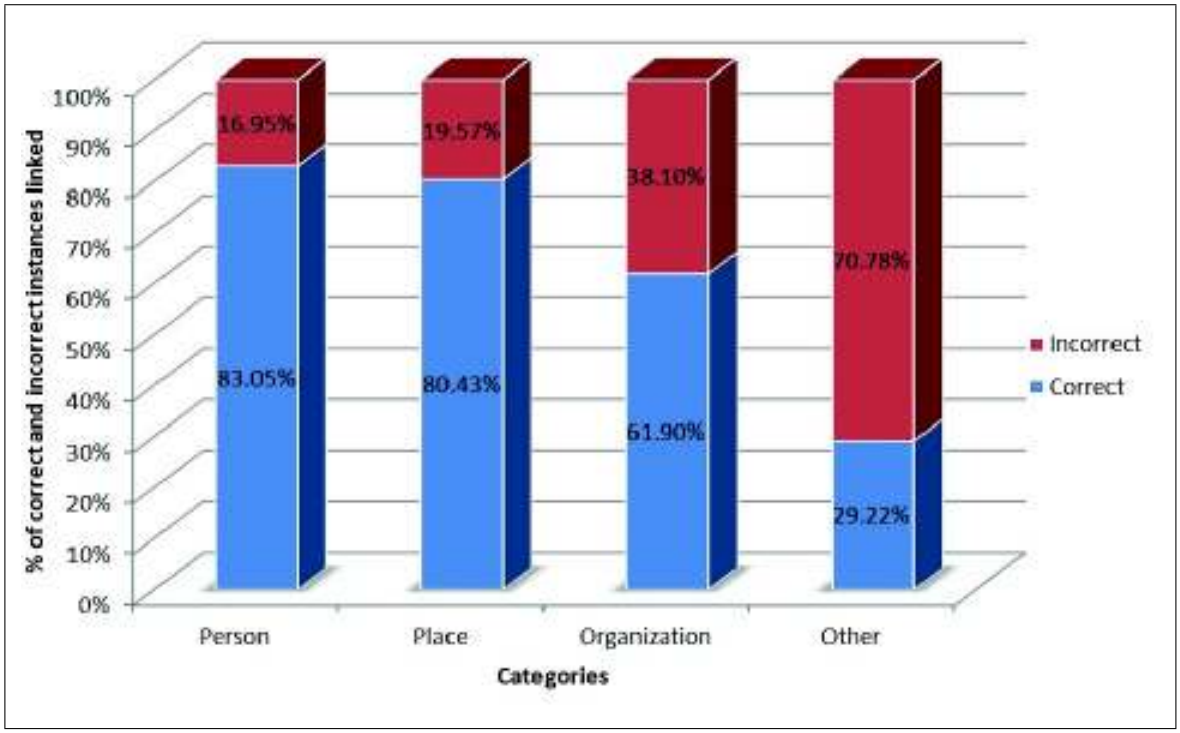

FIG. 4.9. A category-wise breakdown accuracy for linking table cells

Correctly predicted top ranked instance	215
Incorrect predicted top ranked instance	7
Total number of instances	222

FIG. 4.10. The classifier accuracy for test data

one which predicts whether to link to the top one or not (Binary SVM yes/no classifier).

4.3.1 Evaluation for the SVM rank classifier

We trained the SVM^{rank} classifier using a training data that consisted of 171 queries each having 10 results returned by the KB (which makes it 1710 feature vectors). For every query, the instance to which the table cell is supposed to be linked was given the highest rank and all other instances were given a same lower rank. If the query results did not include the correct instance, then all the instances were given a same lower rank. For such a query there would be no top-ranked instance. Within each query the classifier learnt to order the instances within the set, assigning the highest rank to the correct instance and lower ranks to the other incorrect instances.

We generated a test - data on similar lines. The test data was an unseen set of data, which the classifier hadn't seen when it learnt. The test data consisted of 222 queries each having 10 results returned by the KB. The results of the classifier on the test data are given in Figure 4.10, the classifier obtained an accuracy of **96.84** % in correctly identifying the top ranked instance.

The final test data (for the 611 entities) had 611 queries again with each having 10 results. The accuracy result of the classifier for the 611 entities is shown in Figure 4.11. The classifier obtained an accuracy of **88.87 %** in correctly identifying the top ranked instance.

Correctly predicted top ranked instance	543
Incorrect predicted top ranked instance	68
Total number of instances	611

FIG. 4.11. The classifier accuracy for the 611 entities

4.3.2 Evaluation for the SVM binary (yes/no) classifier

The training data for the Binary SVM (yes/no) classifier consisted of 222 feature vectors. The 222 feature vectors were of the instances which were classified as top ranked by SVM^{rank} classifier. For each vector, we assigned a class label of yes if the top ranked instance was the correct one for linking, else we assigned a class label of no if the top ranked instance was incorrect one for linking. The distribution of positive (class yes) and negative (class no) is as follows - the training data had 146 positive examples and 76 negative examples. The classifier learnt to assign a class label yes to if the linking is correct and a class label of no, if the classifier thought the linking was incorrect.

The test data was a set of unseen data, not seen by the classifier during the training phase. It consisted of 171 queries, which were assigned a class label of yes or no on similar lines described above. The test data had 119 positive examples and 52 negative examples. The results of the classifier on the test data are given in Figure 4.12. The classifier obtained an accuracy of **84.79 %** in correctly identifying whether to link to the top ranked instance returned by SVMrank or not. The Precision, Recall and F-Measure values for the test data are presented in Figure 4.13.

The final test data (for the 611 entities) had 611 queries. The accuracy result of the classifier for the 611 entities is shown in 4.14. The classifier obtained an accuracy of **88.54 %** in correctly identifying whether to link to the top ranked instance returned by SVMrank or not. The Precision, Recall and F-Measure values for the final test data are presented in

Correctly Classified Instances	145
Incorrectly Classified Instances	26
Total number of instances	171

FIG. 4.12. Accuracy for test data of the SVM binary Classifier

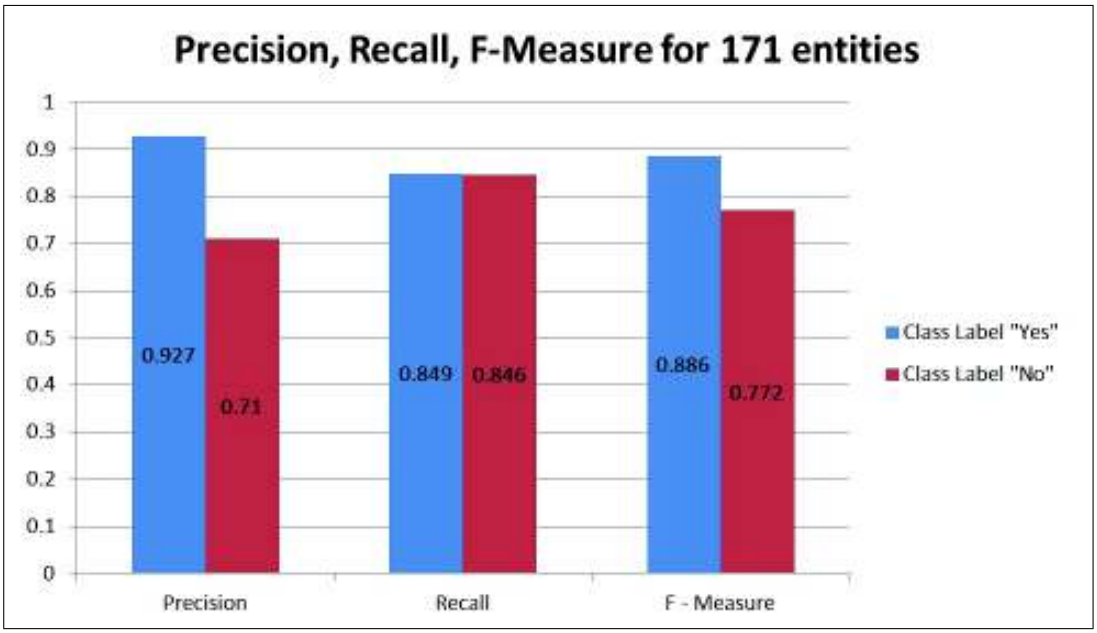

FIG. 4.13. Precision, Recall and F-Measure for the test data of the SVM binary Classifier

Correctly Classified Instances	541
Incorrectly Classified Instances	70
Total number of instances	611

FIG. 4.14. The accuracy for SVM binary Classifier for the 611 entities

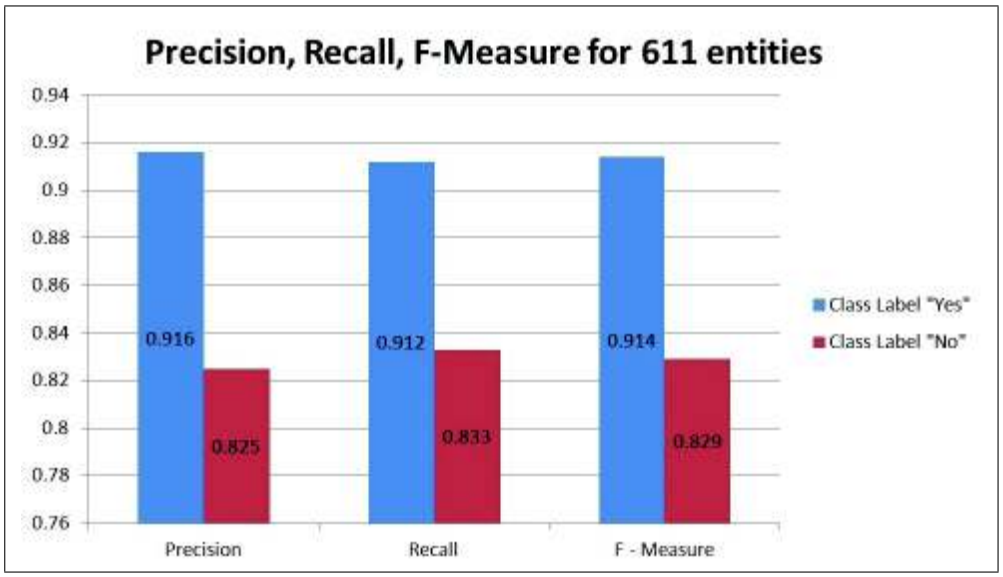

FIG. 4.15. Precision, Recall and F-Measure of the SVM binary Classifier for all 611 entities

Figure 4.15.

The values of Precision, Recall and F-Measure for both the class labels yes and no, in both the test sets shows that the classifier does a pretty good job of identifying whether to link to the top ranked instance or not.

4.4 Evaluation for Column relations

Evaluating relation identified between columns was again tricky since we did not have labeled data. A solution to this problem would be getting humans to look at tables and

identify all possible relations between. Humans could annotate the tables to indicate which columns in the table are related.

We were not sure of the feasibility of the idea, thus we decided to do a pilot experiment. We selected five random tables from our data set and asked three evaluators to identify if relations existed between pairs of columns in the table. The three evaluators were fairly successful in identifying columns between which a relation exists, but there were cases where the evaluators themselves were not sure whether a relation can exist between given two columns.

Since the tables used in this evaluation were fairly simple and less ambiguous, it was possible for the evaluators to identify the columns between which a relation exists. However this approach needs to tested with more complex and ambiguous tables, to see how the evaluators fare in identifying relations.

For the given five tables, the evaluators were able to identify 20 different relations between various columns. Of all the columns and relations identified by the user, we choose all such relations that two out of the three evaluators were able to identify. Our approach of relation identification did not fare well against the human judgment. We were able to identify only 5 out of the 20 relations the humans identified, giving us an accuracy of just **25 %**.

Our approach did not do well since the basis of Algorithm "IdentifyColumnRelations" is the table cell strings being linked accurately. The sparseness of data in the KB and the error in the linking of table cells are possible reasons for the low accuracy. We discuss an alternate and possible approach for relation identification and discovery in the next chapter.

Chapter 5

DISCUSSION AND FUTURE WORK

Evaluations show that our proposed approach can be used to convert and represent data stored in tables as Linked Data. However there are many issues and problems that still need to be handled and addressed. We present some of these issues here.

5.1 Machine Learning based Algorithm for Class Label Prediction

Our evaluation shows that the algorithm "PredictClassLabel" (Algorithm 1, Chapter 3, Section 3.1) works well in predicting a correct label, but is does not work well in ranking and picking the best or more accurate class label for a column. The algorithm is based on a couple of heuristics.

The first heuristic used is in the formula for score, which is used to assign a score for every class label - string pairing. The score is weighted sum of the instance's Wikitology rank and its predicted page rank. In the process of weighting we give a higher weight to Google page rank (w = 0.75) based on the heuristic that more popular the instance / entity is, more likely it is a correct answer as compared to a less popular one.

The second heuristic is used in the selection of class labels as candidate labels for a column header. If a particular class label has a sum score (over the entire column) less than a particular score, we do not select it as a candidate label. We use a cutoff score of 0.3. This

is based on the heuristic that if a class label has a low score it means that not many strings contributed or voted towards the class label, and hence it is less likely to be a correct label that can represent all the strings in the column.

We would like to explore the idea of replacing this heuristic based scoring by some machine learning techniques. We can build a feature set for all possible class labels for a column and train a classifier. The feature set for every class label can include the following features - the normalized sum of instances' Wikitology rank that voted for the class label, normalized sum of the approximate page ranks of the same instances, a measure of semantic similarity between the class label and the table column header, and a measure of how deep the class label is in the hierarchy of classes.

Since there can be more than one correct label for each column (For e.g. in Figure 1.1. for Column 1 "City" the correct labels can be any one of dbpedia-owl:City, dbpedia-owl:Settlement, dbpedia-owl:PopulatedPlace and dbpedia-owl:Place), we could use an algorithm that can learn to rank instances within a given set.

5.2 Discovery of Relations between columns

The basis of the algorithm "IdentifyColumnRelations" (Algorithm 3, Chapter 3, Section 3.3) is on table cell strings being linked to entities from Linked Open Data cloud. However this approach will not work well if our linking of table cell strings is inaccurate or if we are not able to link the strings to entities (which can happen when table contains entities that are not present in the KB). Another shortcoming of this approach is that it identifies existing relations between strings in the two columns, to identify relations between columns. Thus we need to figure out an alternative approach for identification and discovery of new relations between columns.

We are considering exploring the following approach. For any two classes on DB-

pedia, we discover what all relations exist between all the instances of those two classes. We can further calculate the probability of occurrence of every such relation given those two classes. We can discover all such relations between all the classes on DBpedia and pre-compute and store the probability of each relation given two classes. We can then use this knowledge in our process of discovery of relation between columns.

For e.g. once we identify the class label of two columns as dbpedia-owl:Politician and dbpedia-owl:Party, we can look up our KB which will have relations with pre-computed probabilities given the two classes dbpedia-owl:Politician and dbpedia-owl:Party.

5.3 Representing Tables as Linked Data

We have proposed an initial representation of tables as Linked Data. Our proposed representation will work well for known entities and entities present in a given KB. However we will discover a lot of entities in tables that are unknown and we have no knowledge about. We won't be able to link such entities with existing entities from the KB, yet in the process of converting the entire table to Linked Data, we can discover facts and knowledge about an unknown entity.

For e.g. suppose we had a place like Arbutus in a column of Populated Places. Using the class label predicted for the column, we discover that the Arbutus is a type of Populated Place. Based on the relations discovered between the column in which Arbutus is and other columns in the table, we can discover that Arbutus is located in Administrative region Baltimore County and the population of Arbutus is the number from the column Population.

How we handle and represent the new information is an area that we will explore in the coming days. We identify that how we publish known as well as unknown tabular data as Linked Data on the Semantic Web, how we make published information useful and what mechanisms do we provide to allow applications and agents to exploit this data, remain

important questions and issues that need to be addressed.

5.4 Evaluation Mechanisms

Evaluating the correctness of our work has been indeed a tricky task. A major issue was availability of labeled data. Although we were able to extract tables from Wikipedia in which the cell values were linked to their appropriate Wikipedia pages, we did not have any data in which column headers were linked to class labels or data in which relations between columns were identified. Thus we had to scale down the number of tables we evaluated our approach on.

For our evaluation of class label prediction for columns and relation between columns, we used judgment of three human judges. In the future we would have more human judges for evaluation of our work. We would explore the use of Amazon Mechanical Turk[1]to manage a large scale evaluation.

We also have a large data set of 2500+ tables discovered on the web by Google[2]. We plan to annotate (i.e. link the table cells) these tables, so that we can run our experiments on this large data set of naturally occurring tables found on the web.

Our analysis of Wikipedia shows that there are about 120,000+ tables in various Wikipedia articles. The advantage of tables found on Wikipedia is that their table cells are already linked. However not all of these tables hold high quality relational data. We plan to explore mechanism to extract tables from Wikipedia which contain useful relational information, which once converted to Linked Data can be exploited applications.

The data set for our current evaluations included very few entities that were not there in the KB. In the coming future, we also plan to test our approach with tables with entities

[1]https://www.mturk.com/mturk/welcome

[2]We would like to thank Dr.Alon Halevy,head of the Structured Data Group of Google Research for sharing this dataset with us

that are not present in the KB.

Chapter 6

CONCLUSION

In this thesis we presented an automated framework for extracting, interpreting and generating Linked Data representation of data stored in tables. In this framework, we presented an approach for associating and predicting a class label for every column header in the table. We also presented an approach for linking the table cell strings to appropriate entities from the Linked Open Data cloud. We also presented mechanism for identifying and discovering relations between the columns of the tables. We also proposed a Linked Data representation for tables in N3 serialization.

Evaluations show that we have been fairly successful in representing tables as Linked Data accurately. Many issues still remain to be addressed as well. Working on the issues discussed in the previous chapter, we can definitely improve our success rate of converting and representing tables as Linked Data. One of the problems with the Semantic Web has always been availability of useful data. With our automated framework, we are unlocking large amounts of tabular data currently inaccessible and useless for the Semantic Web and making it more meaningful and useful on the Semantic Web for agents and applications to exploit. We believe our work will contribute in materializing the web of data vision.

Appendix A

EVALUATION GUIDELINES

A.1 Evaluation Guidelines for ranking class labels

The following guidelines were presented to the human judges, when they were asked to rank the class labels for every column.

Evaluation Guidelines

- You are given a table at the top of every evaluation sheet.
- For every column in the table, you are given a list of descriptions or labels that can be used to describe the values in the column.
- You have to rank the descriptions or labels that will be the best match for all the values in the column in the context of the entire table.
- The label that is the best match should get the highest rank (1), the second best match getting the next rank (2), the third best getting a rank of 3 and so on.
- You can choose the ranks from the dropdown box provided besides each label.
- The labels in the options may have a hierarchy associated amongst themselves.
- For example in the DBpedia ontology (from where the labels are) City is a subclass of PopulatedPlace which in turn is a subclass of Place. In such a case, you may want

to assign a Rank of 1 to city (since its the finest description), followed by Rank 2 to PopulatedPlace and Rank 3 to Place.

If you wish to, you can browse the hierarchy of the class labels at -

- http://www4.wiwiss.fu-berlin.de/dbpedia/dev/ontology.htm
- http://mappings.dbpedia.org/server/ontology/classes

An example :

City	State	Country
Baltimore	MD	US
Boston	MA	US
New York	NY	US
Los Angeles	CA	US

Suppose you had to choose a label for the column “City” like the one shown above. Given the options, a possible correct way of ranking would be as follows -

http://dbpedia.org/ontology/Place ... [3]*
http://dbpedia.org/ontology/PopulatedPlace ... [2]*
http://dpedbia.org/ontology/City ... [1]*
None of the above ... [4]*

* - The numbers in the square bracket indicates the rank for the labels for column “City”.

A.2 Evaluation Guidelines for correctness of class labels

The following guidelines were presented to the human judges, when they were asked to evaluate the correctness of the class label predicted for every column.

Evaluation Guidelines

- You are given a table at the top of every evaluation sheet.
- For every column in the table, you are given a class label that describes the values in the column.
- You have to decide whether the class label is a correct class label for that column or not.
- If the class label is correct, you should mark "Yes" or if the class label is incorrect you should mark "No"
- Even though a better and a more accurate class label may exist, in this evaluation, you need to decide whether the given class label is fair and correct one or not.

An example :

City	State	Country
Baltimore	MD	US
Boston	MA	US
New York	NY	US
Los Angeles	CA	US

Q. Is the label http://dbpedia.org/onotlogy/PopulatedPlace a correct label for column "City"?

A. Answer to this question would be “yes”. Even though a more accurate label like http://dbpedia.org/onotlogy/City exist since its a fair and correct label.

Q. Is the label http://dbpedia.org/onotlogy/Highways a correct label for column “State”?

A. Answer to this question would be “no”. Since the label is inaccurate for the column

REFERENCES

[1] Auer, S.; Feigenbaum, L.; Miranker, D.; Fogarolli, A.; and Sequeda, J. 2010. Use cases and requirements for mapping relational databases to rdf. W3c working draft, W3C.

[2] Barrasa, J.; scar Corcho; and Gmez-prez, A. 2004. R2o, an extensible and semantically based database-to-ontology mapping language. In *in In Proceedings of the 2nd Workshop on Semantic Web and Databases(SWDB2004*, 1069–1070. Springer.

[3] Berners-Lee, T., and Connolly, D. 2008. Notation3 (n3): A readable rdf syntax. W3c team submission, W3C.

[4] Berners-Lee, T. 1989. Information management: A proposal.

[5] Bizer, C., and Seaborne, A. 2004. D2rq - treating non-rdf databases as virtual rdf graphs. In *ISWC2004 (posters)*.

[6] 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. *J. Web Sem.* 7(3):154–165.

[7] Bizer, C. 2009. The emerging web of linked data. *IEEE Intelligent Systems* 24(5):87–92.

[8] Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, 1247–1250. New York, NY, USA: ACM.

[9] Brickley, D., and Guha, R. 2004. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, World Wide Web Consortium.

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

[11] Finin, T., and Syed, Z. 2010. Creating and exploiting a web of semantic data. In *Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 1 - Artificial Intelligence, Valencia, Spain, January 22-24, 2010*, 7–18. INSTICC Press.

[12] Flanagan, D. 2007. Developing metaweb-enabled web applications.

[13] Golbeck, J.; Grove, M.; Parsia, B.; Kalyanpur, A.; and Hendler, J. A. 2002. New tools for the semantic web. In Gómez-Pérez, A., and Benjamins, V. R., eds., *EKAW*, volume 2473 of *Lecture Notes in Computer Science*, 392–400. Springer.

[14] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The weka data mining software: an update. *SIGKDD Explorations* 11(1):10–18.

[15] Han, L.; Finin, T.; Parr, C.; Sachs, J.; and Joshi, A. 2008. RDF123: from Spreadsheets to RDF. In *Seventh International Semantic Web Conference*. Springer.

[16] Hatcher, E., and Gospodnetic, O. 2004. *Lucene in Action (In Action series)*. Manning Publications.

[17] Hu, W., and Qu, Y. 2007. Discovering simple mappings between relational database schemas and ontologies. In Aberer, K.; Choi, K.-S.; Noy, N. F.; Allemang, D.; Lee, K.-I.; Nixon, L. J. B.; Golbeck, J.; Mika, P.; Maynard, D.; Mizoguchi, R.; Schreiber, G.; and Cudré-Mauroux, P., eds., *ISWC/ASWC*, volume 4825 of *Lecture Notes in Computer Science*, 225–238. Springer.

[18] Hu, M.; Lim, E.-P.; Sun, A.; Lauw, H. W.; and Vuong, B.-Q. 2007. Measuring article quality in wikipedia: models and evaluation. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, 243–252. New York, NY, USA: ACM.

[19] Joachims, T. 2006. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, 217–226. New York, NY, USA: ACM.

[20] Langegger, A., and Wob, W. 2009. Xlwrap - querying and integrating arbitrary spreadsheets with sparql. In Bernstein, A.; Karger, D. R.; Heath, T.; Feigenbaum, L.; Maynard, D.; Motta, E.; and Thirunarayan, K., eds., *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, 359–374. Springer.

[21] Lassila, O., and Swick, R. 1999. Resource description framework (rdf): Model and syntax specification. recommendation. Technical report, W3C.

[22] Lawrence, E. D. R. 2004. Composing mappings between schemas using a reference ontology. In *In Proceedings of International Conference on Ontologies, Databases and Application of SEmantics (ODBASE*, 783–800. Springer.

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

[24] Lin, C. X.; Zhao, B.; Weninger, T.; Han, J.; and Liu, B. 2010. Entity relation discovery from web tables and links. In Rappa, M.; Jones, P.; Freire, J.; and Chakrabarti, S., eds., *WWW*, 1145–1146. ACM.

[25] Manning, C. D.; Raghavan, P.; and Schütze, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

[26] Miller, G. A. 1995. Wordnet: a lexical database for english. *Commun. ACM* 38:39–41.

[27] 2004. Owl web ontology language overview. W3c recommendation, World Wide Web Consortium.

[28] Pantel, P.; Philpot, A.; and Hovy, E. 2005. Aligning database columns using mutual information. In *Proceedings of the 2005 national conference on Digital government research*, dg.o '05, 205–210. Digital Government Society of North America.

[29] Papapanagiotou, P.; Katsiouli, P.; Tsetsos, V.; Anagnostopoulos, C.; and Hadjiefthymiades, S. 2006. Ronto: Relational to ontology schema matching. In *AIS SIGSEMIS BULLETIN*.

[30] Prud'hommeaux, E., and Seaborne, A. 2007. SPARQL query language for RDF (working draft). Technical report, W3C.

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

[32] Suchanek, F. M.; Kasneci, G.; and Weikum, G. 2007. Yago: A Core of Semantic Knowledge. In *16th international World Wide Web conference (WWW 2007)*. New York, NY, USA: ACM Press.

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

[34] Vanoirbeek, C. 1992. Formatting structured tables (invited paper). In *Proceedings of Electronic Publishing, '92, International Conference on Electronic Publishing,Document Manipulation, and Typography, Swiss Federal Institute of Technology,*

Lausanne, Switzerland, April 7-10, 1992, 291–309. New York: Cambridge University Press.

[35] Ziegler, P., and Dittrich, K. R. 2004. Three decades of data intecration: all problems solved? In *Building the Information Society*, volume 156 of *IFIP International Federation for Information Processing*, 3–12. Springer Boston.