Using linked data to interpret tables*

Varish Mulwad, Tim Finin, Zareen Syed, and Anupam Joshi

Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County, Baltimore, MD USA 21250
{varish1,finin,joshi}@cs.umbc.edu, zareensyed@gmail.com

Abstract. Vast amounts of information is available in structured forms like spreadsheets, database relations, and tables found in documents and on the Web. We describe an approach that uses linked data to interpret such tables and associate their components with nodes in a reference linked data collection. Our proposed framework assigns a class (i.e. type) to table columns, links table cells to entities, and inferred relations between columns to properties. The resulting interpretation can be used to annotate tables, confirm existing facts in the linked data collection, and propose new facts to be added. Our implemented prototype uses DBpedia as the linked data collection and Wikitology for background knowledge. We evaluated its performance using a collection of tables from Google Squared, Wikipedia and the Web.

Keywords: Semantic Web, linked data, human language technology, entity linking, information retrieval

1 Introduction

Resources like Wikipedia and the Semantic Web's linked open data collection [1] are now being integrated to provide experimental knowledge bases containing both general purpose knowledge as well as a host of specific facts about significant people, places, organizations, events and many other entities of interest. The results are finding immediate applications in many areas, including improving information retrieval, text mining, and information extraction. Still more structured data is being extracted from text found on the web through several new research programs [2, 3].

We describe a prototype system that automatically interprets and extracts information from table found on the web. The system interprets such tables using common linked data knowledge bases, in our case DBpedia, and Wikitology [4], a custom hybrid knowledge base for background knowledge. To develop an overall interpretation of the table, we assign a class to every table column and link every table cell to an entity from the LOD cloud. We also present preliminary work in identifying relations between table columns. This interpretation can be used for variety of tasks; in this paper we describe the task of annotation of web tables

* Research supported in part by a gift from Microsoft Research, a Fulbright fellowship, NSF award IIS-0326460 and the Human Language Technology Center of Excellence.

for the Semantic Web. We describe a template used to publish the annotations as N3. Our implemented prototype was evaluated using a collection of tables from Google Squared, Wikipedia and tables found on the Web.

2 Motivation and Related Work

While the availability of data on the Semantic Web has been progressing slowly, the Web continues to grow at a rapid pace. In July 2008, Google announced that they had indexed one trillion unique documents on the web[1]. And much of this data on the Web is stored in HTML tables. Caferella et al. [5] estimated that there are around 14.1 billion HTML tables, out of which 154 million contain high quality relational data.

As a part of the Linked Open Data initiative US, UK and various other governments have also shared publicly available government data in tabular form. This represents a huge source of knowledge currently unavailable on the Semantic Web. There is a need for systems that can automatically generate data in suitable formats for the Semantic Web from existing sources, be it unstructured (e.g., free text), semi-structured (e.g., text embedded in forms or Wikis) or structured (e.g., data in spreadsheets and databases).

Extracting and representing tabular data as RDF is not a new problem. Significant research has been done in the area of mapping relational databases to RDF; various manual and semi-automatic approaches have been proposed(see [6–10]). To standardize the mapping language, for mapping relational databases to RDF and OWL, the W3C has formed a working group RDB2RDF[2]. On June 8, 2010 the group published its first working draft, capturing use-cases and requirements to map Relational Databases to RDF [11].

The other research focus has been on mapping spreadsheets into RDF (see [12, 13]). While existing systems like RFD123 [12] are practical and useful for extracting knowledge from tables they suffer from several shortcomings. Such systems require human intervention, for e.g., requiring the users to choose classes and properties from appropriate ontologies to be used in the annotations. These systems do not offer any automated (or semi-automated) mechanisms for associating the columns headers with known classes or linking the entities in spreadsheets to known entities from the linked data cloud.

While these systems generate triples, since the columns and entities are not linked, the triples are not much of use to other applications that want to exploit this data. In certain cases the triplified data is as useless as raw data would have been on the Semantic Web. Just triplifying the data may not be that useful.

While Han et al. [14] focused on the problem of associating possible types with column headers, it did not focus on a complete interpretation of the table, nor integrating the table with the linked open data cloud. Limaye et al. [15] do the exact same sub-tasks as we do, however their goal is answering search queries

[1] http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html
[2] http://www.w3.org/2001/sw/rdb2rdf/

over web tables. The focus of this paper is an automatic framework for interpreting tables using existing linked data knowledge and using the interpretation generating linked annotated RDF from web tables for the Semantic Web.

3 Interpreting a table

Consider the table shown in Figure 1. The column headers suggest the type of information in the columns: Name and Team - might match classes in a target ontology such as DBpedia [16]; Position could match properties in the same or related ontologies. Examining the data values, which are initially just strings, provides additional information that can confirm some possibilities and disambiguate between possibilities for others. The strings in column one can be recognized as entity mentions that are instances of the *dbpedia-owl:Person* class and can be linked to known entities in the LOD. Additional analysis can automatically generate a narrower description, such as *dbpedia-owl:BasketballPlayer*.

Name	*Team*	*Position*
Michael Jordan	Chicago	Shooting guard
Allen Iverson	Philadelphia	Point guard
Yao Ming	Houston	Center
Tim Duncan	San Antonio	Power forward

Fig. 1. In this simple table about basketball players, the column header represents the type of data stored in columns; values in the columns represent instances of that type.

However just examining the string values in the column may not be enough. Consider the strings in column two. A initial examination of just the strings would suggest that they may be instances of the *dbpedia-owl:PopulatedPlace* class and that the strings should be linked to the respective cities. But in this case, this analysis would be wrong, since they are referring to NBA basketball teams.

Thus it is important to consider additional context evidence, provided by the column header and rest of the row values. In this example, given the evidence that values in column one are basketball players and values in column three are their playing position, we would be able to infer correctly that values in column two are basketball teams and not cities in the United States.

Identifying relations between columns will be important as well, since relations can help identify the columns which can be mapped as properties of some other column in the table. For example, the values in column three are values of the property *dbpedia-owl:position* which can be associated with the players in column one.

Producing an overall interpretation of a table is a complex task that requires developing an overall understanding of the intended meaning of the table as well as attention to the details of choosing the right URIs to represent both the schema as well as instances. We break down the process into following tasks:

- assign every column a class label from an appropriate ontology
- link table cell values to appropriate LD entities, if possible

– discover relationships between the table columns and link them to linked data properties

In this paper we focus on the first two tasks - associating type/class label with a column header and linking table cell to entity. We also present preliminary work on the discovering relations between columns. We also present how this interpretation can be used to annotate webtables. The details of our approach and its prototype implementation are described in Section 4 and the results of the evaluation are described in Section 5.

4 Approach

Our approach comprises four steps: associating ontology classes with columns, linking cell values to instances of those classes, discovering implicit relations between columns in the table, and generating annotation output. We discuss each step in turn.

4.1 Associating Classes with Columns

In a typical well formed table, each column contains data of a single syntactic type (e.g., strings) that represent entities or values of a common semantic type (e.g., people, yearly salary in US dollars). The column's header, if present, may name or describe the semantic type or perhaps a relation in which the column participates. Our initial goal is to predict a semantic class from among the possible classes in our linked data collection that best characterizes the column's values. Our approach is to map each cell value to a ranked list of classes and then to select the one which best characterizes the entire column. Algorithm 1 describes the steps involved in the process.

The algorithm first determines the type or class of each string in the column, by submitting a complex query to the Wikitology KB. The KB returns a ranked list of top N instances for each string in the column and their class. Using the classes of the instances returned by the KB, a set of possible class labels for a column is generated. Each class label in this set is assigned a score based on the weighted scoring technique described in algorithm 1. The class labels in the set are paired with strings in the column and each pair gets scored. The score is based on the highest ranked instance of the string matching the class being scored. The score is a weighted sum of the instance's Wikitology rank for the query and it's approximate page rank [17]. The class label that maximizes its score over the entire column is chosen as the class label to be associated with the column. We predict class labels from four vocabularies - DBpedia Ontology, Freebase [18], WordNet [19], and Yago [20].

In the following sections we describe our knowledge base and our custom query module, used to the query the knowledge base.

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 the Wikitology 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 × N instances, generate a set of class labels that can be associated with a column. Let C be the set of all associated classes for a column.
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):

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

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.

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. 2. Description of the query to Wikitology for a table cell

Knowledge Base. We use DBpedia as our linked data knowledge base. We also use Wikitology, a hybrid KB of structured and unstructured information extracted from from Wikipedia augmented with structured information from DBpedia, Freebase, WordNet and Yago. The interface to Wikitology is via a specialized information retrieval index implemented using Lucene [21] that allows unstructured, structured as well as hybrid queries. Given the simple, yet powerful query mechanism augmented with the fact that the backbone of Wikitology is Wikipedia, one of the most comprehensive collaborative encyclopedia, we think Wikitology along with DBpedia are appropriate choices as KBs. The approach we have described so far and the approaches we describe further are KB independent, allowing the use of any appropriate and suitable linked data knowledge bases as and when needed.

Mapping table to Wikipedia. The table cell string that is being queried for along with its row data and column header are mapped to the various fields of the Wikitology index. The cell string is mapped to the title, redirects and the first sentence fields of the index. If there's a difference in spelling between the string in question and the title or a pseudo name is used, it may appear in the redirects. The column header is mapped to the first sentence, types and categories, since the column header of a table often describes the type of instances present in the column and the type is also likely to appear in the first sentence as well.

The string (with a Lucene query weight boost of 4.0) along with the row data is mapped to contents as well as the linked concepts fields. In a table the data present in a given row are likely to have some kind of relation amongst themselves, hence we map the row data to the linked concept fields which captures the linked concepts (article) to a given Wikipedia concept (article). The row data (excluding the string) is mapped to property values field, since the row data can be values of a property associated with the string in question. Figure 2 describes the query. All the fields are"*ored*" with each other. The query returns top N instances that the string in the query could be associated with, along with their types, page length and their approximate PageRank.

Augmenting types from DBpedia. For every instance returned by Wikitology, we also query DBpedia using its public SPARQL endpoint[3] to fetch the types associated with that instance on DBpedia. The types returned by Wikitology are augmented with the types returned by DBpedia to the get a complete and accurate set of types for a given instance.

4.2 Linking Table Cells to Entities

We have developed an algorithm "LinkTableCells" (see algorithm 2) to link table cell strings to entities from the Linked Open Data cloud. For every string in the table, the algorithm re-queries the KB using the predicted class labels for the column to which the string belongs to, as additional evidence. The predicted class labels are mapped to the typesRef field of the Wikitology index and "*anded*" in the query in Figure 2 , thus restricting the type of entities returned by the KB to the predicted types (class labels).

For each of the top N entities returned by the KB, a feature vector is generated. The feature vector consists of the entity's index score, entity's Wikipedia page length, entity's page rank, (all popularity measures) the Levenshtein distance [22] between the entity and the string in the query and the Dice score [23] between the entity and the string (similarity measures). The Levenshtein distance and the Dice score is calculated between the query string and all labels (all possible names) for the entity. To obtain all other possible names, we query DBpedia to get the values associated with the *rdfs:label* property of the entity. The best Levenshtein distance (i.e. the smallest) and the best Dice score (i.e. the largest) are selected as a part of the feature vector. We choose popularity

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

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

measures as a part of the feature vector because in cases where it is difficult to disambiguate between entities, the most popular entity is often the correct answer and choose similarity measures because the entity that will get linked to query string will be similar if not same in terms of string comparison.

For each query string, a set of feature vectors is generated from the top N instances returned as query results. A classifier built using SVM-rank [24] ranks the entities based on the feature vector set. A second classifier is trained to decide whether the evidence is strong enough to link to the top ranked entity or not. The classifier decides based on the feature vector of top ranked entity which now include two additional features - the SVM-rank score of the entity and the difference in scores between the top two ranked entities by SVM-rank. If the evidence is strong enough, the classifier suggests to link to the top ranked entity, else it suggests to link to "NIL". The above process is repeated for all the strings in the table.

The second SVM classifier was trained using Weka. In cases where linking to the top ranked entity returned by SVM-rank based classifier would be incorrect for example, if the entity is not present in the KB, the second classifier is useful to determine to link the query string to the entity or to predict a link to "NIL".This step is useful in discovery of new entities in a given table.

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

"Name"@en is rdfs:label of dbpedia-owl:BasketballPlayer .
"Team"@en is rdfs:label of yago:NationalBasketballAssociationTeams .

"Michael Jordan"@en is rdfs:label of dbpedia:Michael_Jordan .
dbpedia:Michael_Jordan a dbpedia-owl:BasketballPlayer .

"Chicago Bulls"@en is rdfs:label of dbpedia:Chicago_Bulls .
dbpedia:Chicago_Bulls a yago:NationalBasketballAssociationTeams .
```

Fig. 3. A example of N3 representation of a table as linked RDF

4.3 Relation identification

We have developed a preliminary approach for identifying relations between table columns. The algorithm generates a set of candidate relations from the relations that exist between the concepts associated with the strings in each row of the two columns. To identify relation between the pair of strings, we query DBpedia using its public SPARQL endpoint.

Each candidate relation is scored as follows - each pair of strings in the two columns vote for the candidate relation with a score of 1, if the candidate relation appears in the set of relations between the the pair of strings. The sum of score of each of the candidate relation is normalized by the number of rows in the table. The relation with the highest score is selected to represent relation between the two columns. Our work on relations identification is pretty preliminary and we have still have to develop approach to identify columns that can be mapped as properties of some other column.

4.4 Annotating the webtable

In the previous section (sections 4.1, 4.2 , 4.3) we described approaches that help us develop an overall interpretation of the table. We now describe how this can be used for annotating a webtable. We have developed a template for annotating and representing tables as linked RDF. We choose N3 because it is compact as well as human readable. Figure 3 shows an example of a N3 representation of a webtable. To associate the column header with its predicted class label, the *rdfs:label* property from RDF Schema is used. The *rdfs:label* property is also used to associate the table cell string with its associated entity from DBpedia. To associate the table string with its type (i.e. class label of the column header), the *rdf:type* property is used.

# of Tables	15
# of Rows	199
# of Columns	56 (52)
# of Entities	639 (611)

(a)

Category	# of Columns (%)	# of Entities (%)
Place	40	45
Person	25	20
Organization	12	10
Other types	23	25

(b)

Fig. 4. (a) provides a summary of the data set. (b) presents the distribution of columns and entities across the four categories.

Mean Average Precision	columns
$m = 1$	11.53%
$0 < m < 1$	69.23%
$m = 0$	19.24%

(a)

Recall	columns
$r = 1$	46.15%
$0 < r < 1$	34.61%
$r = 0$	19.24%

(b)

Fig. 5. The percentage of columns with various MAP (see a) and recall (see b) scores.

5 Evaluation

Our implemented prototype was evaluated against 15 tables obtained from Google Squared, Wikipedia and from a collection of tables extracted from the Web[4]. We consider simple regular tables with column headers and tables where the number of cells is equal to the product of number rows and columns. We do not consider tables which have been used for formatting. The task of assigning class label to column header was evaluated against 52 columns; linking the table cell to entity against 611 entities. The distribution of the columns and entities across the four categories - *Persons*, *Places*, *Organizations* and *Other (movies, songs, nationality etc.)* is as shown in Figure 4(b)

5.1 Class label prediction for columns

We used human judgments to evaluate the correctness of the class labels predicted by our approach. We evaluated the class label predicted from the DBpedia ontology, since it would have been fairly easy for our evaluators to browse the DBpedia ontology. In the first evaluation of the algorithm for assigning class labels to columns, we compared the ranked list of possible class labels generated by the system against the list of possible class labels ranked by the evaluators. As shown in Figure 5 for 80.76% of the columns the Mean Average Precision (MAP) [25] between the system and evaluators list is greater than 0 which indicates that there was at least one relevant label in the top three of the system ranked list. For 75% of the columns, the recall of the algorithm was greater than or equal to 0.6. A high recall value shows that there is a high match between the labels in the top three of the system as compared to the top three list of the evaluators[5].

[4] Set of tables available online at www.cs.umbc.edu/∼varish1/t2ld-tables/

[5] While the top three from the system and evaluator's list were compared; the total length of the list varied between 5 and 11 which depended upon the set of possible class labels for every column

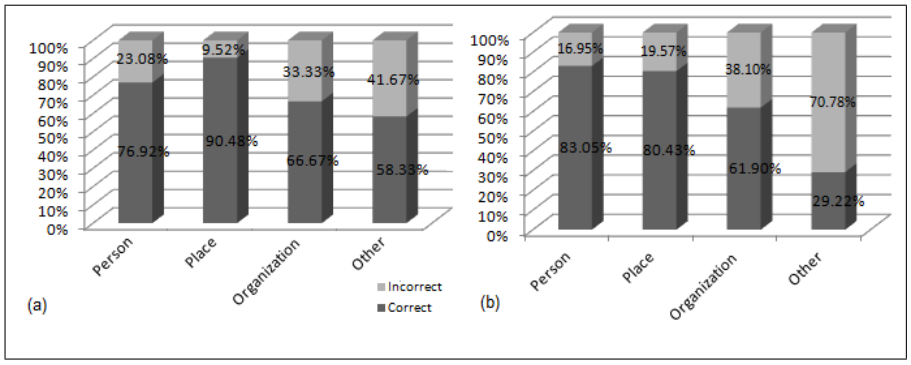

Fig. 6. Category wise accuracy for "column correctness" is shown in (a) and for entity linking in (b)

In the final evaluation, we tried to assess whether our predicted class labels were reasonable based on the judgment of human evaluators. Even though a more accurate class label may exist for a given column, the evaluators needed to determine whether the predicted class was reasonable. For example, for a column of cities, a human might judge *dbpedia-owl:City* as the most appropriate class, consider *dbpedia-owl:PopulatedPlace* and *dbpedia-owl:Place* as acceptable, and consider other classes as unacceptable (e.g., *dbpedia-owl:AdministrativeRegion*, *owl:Thing*, etc). 76.92% of the class labels predicted were considered correct by the evaluators. The accuracy in each of the four categories is shown in Figure 6. We enjoyed moderate success in assigning class labels for *Organizations* and *Other* types of data probably because of sparseness of data in the KB about these types of entities.

5.2 Linking table cells to entities

For the evaluation of linking table cells to entities, we manually hand-labeled the 611 table cells to their appropriate Wikipedia / DBpedia pages. The system generated links were compared against the expected links. 66.12% of the table cell strings were correctly linked. A look at the breakdown of accuracy based on the categories (Figure 6) 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% probably because of sparseness of data in the KB about these types of entities.

Our dataset had 24 entities which were unknown to the KB and in all the 24 cases, the system was able to predict correctly that the table cell should be linked to "NIL". Comparing the entity linking results against our previous work where we used a heuristic based method for linking table cells to entity [17], 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[6] of the current data set.

5.3 Relation identification

We did a preliminary evaluation for identification of relation between columns. We asked human evaluators to identify pairs of columns in a table between which a relation may exist and compared that against the pairs of columns identified by the system. For five tables, used in this evaluation, in 25% of the cases, the system was able to identify the correct pairs of columns.

6 Conclusion

We presented an automated framework for interpreting data in a table using existing Linked Data KBs. Using the interpretation of the table we generate linked RDF from webtables. Evaluations show that we have been fairly successful in generating correct interpretation of webtables. Our current work is focused on improving relationship discovery and generating new facts and knowledge from tables that contain entities not present in the LOD knowledge bases. To deal with web scale analytics, we plan to focus on adapting our algorithms for parallelization using Hadoop or Azure type frameworks. We are also exploring ways to apply this work to create an automated (or semi-automated / human in the loop) framework for interpreting and representing public government datasets as linked data.

References

1. Bizer, C.: The emerging web of linked data. IEEE Intelligent Systems **24** (2009) 87–92
2. Etzioni, O., Banko, M., Cafarella, M.: Machine reading. In: Proceedings of the National Conference on Artificial Intelligence. Volume 21., Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2006) 1517
3. McNamee, P., Dang, H.: Overview of the TAC 2009 knowledge base population track. In: Proceedings of the 2009 Text Analysis Conference. (2009)
4. Finin, T., Syed, Z.: Creating and Exploiting a Web of Semantic Data. In: Proc. 2nd International Conference on Agents and Artificial Intelligence, Springer (2010)
5. Cafarella, M.J., Halevy, A.Y., Wang, Z.D., Wu, E., Zhang, Y.: Webtables: exploring the power of tables on the web. PVLDB **1** (2008) 538–549
6. Barrasa, J., Corcho, O., Gómez-pérez, A.: R2o, an extensible and semantically based database-to-ontology mapping language. In: Proc. 2nd Workshop on Semantic Web and Databases(SWDB2004). Volume 3372. (2004) 1069–1070

[6] The subset had 171 entities to link

7. Hu, W., Qu, Y.: 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., Cudré-Mauroux, P., eds.: ISWC/ASWC. Volume 4825 of Lecture Notes in Computer Science., Springer (2007) 225–238
8. Papapanagiotou, P., Katsiouli, P., Tsetsos, V., Anagnostopoulos, C., Hadjiefthymiades, S.: Ronto: Relational to ontology schema matching. In: AIS SIGSEMIS BULLETIN. (2006)
9. Lawrence, E.D.: Composing mappings between schemas using a reference ontology. In: Proceedings of International Conference on Ontologies, Databases and Application of Semantics (ODBASE), Springer (2004) 783–800
10. Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr, T., Auer, S., Sequeda, J., Ezzat, A.: A survey of current approaches for mapping of relational databases to rdf. Technical report, W3C (2009)
11. Auer, S., Feigenbaum, L., Miranker, D., Fogarolli, A., Sequeda, J.: Use cases and requirements for mapping relational databases to RDF, W3C working draft. Technical report (2010)
12. Han, L., Finin, T., Parr, C., Sachs, J., Joshi, A.: RDF123: from Spreadsheets to RDF. In: Seventh International Semantic Web Conference, Springer (2008)
13. Langegger, A., Wob, W.: Xlwrap - querying and integrating arbitrary spreadsheets with sparql. In: 8th International Semantic Web Conference (ISWC2009). (2009)
14. Han, L., Finin, T., Yesha, Y.: Finding Semantic Web Ontology Terms from Words. In: Proceedings of the Eigth International Semantic Web Conference, Springer (2009) (poster paper).
15. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. In: Proc. of the 36th Int'l Conference on Very Large Databases (VLDB). (2010)
16. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia - a crystallization point for the web of data. Journal of Web Semantics **7** (2009) 154–165
17. Syed, Z., Finin, T., Mulwad, V., Joshi, A.: Exploiting a Web of Semantic Data for Interpreting Tables. In: Proc. Second Web Science Conference. (2010)
18. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: 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, New York, NY, USA, ACM (2008) 1247–1250
19. Miller, G.A.: Wordnet: a lexical database for english. Commun. ACM **38** (1995) 39–41
20. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: 16th Int. World Wide Web Conf., New York, ACM Press (2007)
21. Hatcher, E., Gospodnetic, O.: Lucene in Action (In Action series). Manning Publications (2004)
22. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8 (1966)
23. Salton, G., Mcgill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York, NY, USA (1986)
24. Joachims, T.: Training linear svms in linear time. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. KDD '06, New York, NY, USA, ACM (2006) 217–226
25. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. 1 edn. Cambridge University Press (2008)