Attribute Retrieval from Relational Web tables

Arlind Kopliku, Karen Pinel-Sauvagnat, Mohand Boughanem

IRIT, University of Toulouse, France
{Arlind.Kopliku, Karen.Sauvagnat, Mohand.Boughanem}@irit.fr

Abstract. In this paper, we propose an attribute retrieval approach which extracts and ranks attributes from HTML tables. Given an instance (e.g. Tower of Pisa), we want to retrieve from the Web its attributes (e.g. height, architect). Our approach uses HTML tables which are probably the largest source for attribute retrieval. Three recall oriented filters are applied over tables to check the following three properties: *(i)* is the table relational, *(ii)* has the table a header, and *(iii)* the conformity of its attributes and values. Candidate attributes are extracted from tables and ranked with a combination of relevance features. Our approach can be applied to all instances and is shown to have a high recall and a reasonable precision. Moreover, it outperforms state of the art techniques.

Keywords: information retrieval, attribute retrieval

1 Introduction

Most information retrieval systems answer user queries with a list of documents, but there are many information needs that can be answered with information within documents. For instance, the query "features of Mac Book" is not asking for all pages speaking of Mac Books rather than for their features.

Inspired by the work in [5], we distinguish three types of information needs that can be answered differently:
- *query by attribute* (GDP of Italy, adress of Hotel Bellagio)
- *query by instance* (Samsung Galaxy, Italy, Oscar Wilde)
- *query by class* (Macintosh notebooks, British writers)

When the query is specifically asking for an attribute, we can return its value right away. When the query is an instance, we can propose a summary of salient attributes (properties). When the query is a class of instances, the result can be a comparative table of the class instances with their attributes. Figure 1 shows what these results might look like. A direct application can be Google Squared[1], a commerical tool that produces similar results. We will call search based on attributes, instances and classes *relational search*.

Relational search is not the only application where attributes play a crucial role. They can also be used for query suggestion [3], faceted search [4], or

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

aggregated search [11]. For instance, given the query "Indonesia" and using attributes of "Indonesia" we can produce suggestions such as "Indonesia climate", "Indonesia capital" and "Indonesia hotels", while in a faceted search context we can navigate returned results on the query "modern architecture" by attributes such as "country", "architecture style" and so on. Attributes can also be used to summarize content about an instance/class.

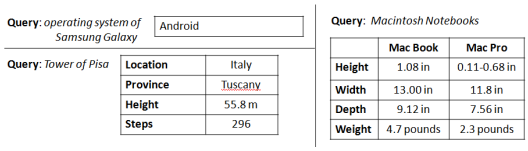


Fig. 1. Examples of relational search results

In this paper, we treat attribute retrieval which falls in the above defined framework. More precisely, the research question we address is identifying relevant attributes for a given instance. We use HTML tables to extract attributes from. There are some main reasons behind this choice. First, tables are widespread across the Web. Second, many tables contain relational data [7]. Third, tables have already been used sucessfully for attribute extraction [9, 8].

However, the task is not easy. Many of the HTML tables are used for layout design and navigation. In [7], authors estimate that only about 1% of the Web tables contain relational data. Furthermore, some relational tables have no headers (schema of attributes) and it is not easy to retrieve all relevant tables for a given instance.

We propose an attribute retrieval approach at Web scale which takes into account the above issues. First, we issue the instance as a query to a search engine. The retrieved documents are used to extract tables from. Successively, we apply 3 filters to candidate tables to check the following properties: *(i)* is the table relational, *(ii)* has the table a header, *(iii)* the conformity of its attributes and values. Extracted attributes are then ranked with relevance features.

Our approach integrates the work of Cafarella et al. [7]. Their work is at our knowledge the largest mining of Web tables for search purposes. They show we can filter out many tables that are not relational and that do not have a header. In addition to their work, we filter out many table columns (or rows) that do not contain attributes (name and values). We also integrate our previous work [12], which shows that we can retrieve attributes using a linear combination of relevance features such as table match, document relevance and external evidence from Web search, DBPedia and Wikipedia.

The paper is structured as follows. The next section is about related work. In section 3, we describe our approach including filtering and ranking of attributes. Then we describe the experimental setup (section 4) and results (section 5) following with conclusions.

2 Related work

To acquire attributes from the Web, it is common to use HTML tags, decoration markup [21, 19, 10] and text [3, 16, 13]. Tags for tables, lists and emphasis have been shown to help for attribute extraction [21, 19]. Wong et al. [19] combine tags and textual features in a Conditional Random Fields model to learn attribute extraction rules, but they need a seed of relevant documents manually fed.

Another common technique to acquire attributes is through the use of lexico-syntactic rules. For example, Pasca et al. [1, 14] use rules such as "A of I" and "I's A" to acquire attributes from query logs. Authors represent the class as a set of instances and multiple class instances are used to improve extraction. In [2], authors use more precise lexico-syntactic rules such as "the A of I is", but recall of these rules is lower. In [15], Popescu et al. use lexico-syntactic rules to extract product attributes from reviews.

At last, tables are known to be a mine for relational data and attributes. Cafarella et al. [7, 6] show we can identify billions of relational tables in the Web. In [9], Chen et al. identify attributes using column (or row) similarities. Another common technique to extract attribute from tables is through wrapper induction [8, 10, 18]. Given a training set or a set of similar documents, wrapper induction learns extraction rules. Many wrappers extract at record level, but they do not distinguish between attribute name and attribute value. Furthermore, wrappers are precision oriented and they work well only for some sites.

To summarize, current attribute acquisition techniques can obtain a high precision. Although many of these techniques produce a considerable number of attributes, they cannot cover the needs that can be answered with the Web. Most of them are conceived to work offline and they cannot extract instance attributes whatever the instance.

Our work is inspired by work in [1] and [6, 7]. It differs from [1], in that we do not use lexico-syntactic rules but Web tables to identify attributes. It differs from the work in [6, 7], in that we do not use tables for relation search. We make use of learnings in [6, 7], but we introduce another filter at line level and we introduce relevance features. The combination of filters and relevance ranking allows us to enable attribute retrieval which was not investigated in [6, 7].

3 Attribute retrieval

We consider an information retrieval situation where the query is an instance and the results is a list of attributes. Concretely, **given an instance i, attribute retrieval should extract and rank attributes with respect to their relevance to i.**

We first collect a seed of potentially relevant tables. We issue the instance as a query to a search engine and we retrieve tables from the retrieved documents. We use these tables to extract attributes from. The problem is far away from being solved. Tables in the Web are quite heterogeneous and many of the retrieved tables are not relevant or they are partially relevant.

Before ranking attributes, we apply three filters on tables and attributes namely *relational filter*, *header filter*, and *attribute line filter*. The first two are the same as in [7]. They are recall oriented classifiers that can filter out many tables that are not relational and that do not have headers. Still, after applying these filters there remain many tables which are not relational. As well there are many tables which are almost relational such as table 2 in figure 2. Instead of filtering out this kind of tables, we introduce another filter at column (or row) level which we call *attribute line filter*. Each line is checked for its conformity for being an attribute line i.e. an attribute name followed with attribute values of similar format and length.

After applying the three filters, we rank the remaining attribute lines with respect to their relevance for the instance. The task is not easy. If we consider only tables that match the instance, we would lose many relevant tables(e.g. table 2 in figure 2 is relevant for "France", but does not match it). To increase recall, we use all tables from relevant documents. Extracted attributes are ranked with relevance features as described in section 3.3.

	Population	Median age
France	64,768,389	39.7 years
Italy	58,090,681	43.7 years

Table 1

Facts	
Capital	Paris
Demonym	French
Population	
Population	64,768,389
Median age	39.7 years

Table 2

1	Loud	Rihanna
2	The king of limbs	Radiohead
3	Femme fatale	Britney Spears

Table 3

Fig. 2. Examples of interesting tables

3.1 Relational tables and headers

We build the relational filter and the header filter using the same features as done by Cafarella et al. in [7]. Features on the left of table 1 are used to learn a rule-based classifier for the relational filter and features on the right are used to learn a rule-based classifier for the header filter. Learning is done with a training set of human-classified tables described later in the experimental setup.

Relational Filter	**Header Filter**
f_1: ♯ rows	f_1: ♯ rows
f_2: ♯ cols	f_2: ♯ cols
f_3: %rows w/mostly NULLS	f_8: %head row cells with lower-case
f_4: ♯ cols w/non-string data	f_9: % head row cells with punctuation
f_5: cell strlen avg μ	f_{10}: % head row cells with non string data
f_6: cell strlen stddev σ	f_{11}: % cols w/non-string data in *body*
f_7: $\frac{\sigma}{\mu}$	f_{12}: % cols w/$\lvert len(row\_1) - \mu\rvert > 2\sigma$
	f_{13} : % cols w/$\lvert \sigma < len(row\_1) - \mu\rvert \leq 2\sigma$
	f_{14} : % cols w/$\lvert \sigma > len(row\_1) - \mu\rvert$

Table 1. Features for the relational and header filter

Features include the dimensions of the table (f_1, f_2), the fraction of lines with

mostly nulls (empty cells) (f_3), the lines with non-string data (numbers, dates) (f_4), statistics on the character length of cells and their variation (f_5, f_6, f_7, f_{12}, f_{13}, f_{14}) and conformity of the header cells (lower-case, punctuation, non-string data) (f_8-f_{11}).

The main difference with the work of Cafarella et al. is that they consider that relational tables are all oriented vertically, i.e. the table header (if present) is on top of the table and the attribute lines are vertical (the columns). For example, tables 1 and 3 in figure 2 are oriented vertically and table 2 is oriented horizontally. We extend their approach to work for both horizontally and vertically oriented tables. This is not difficult. We consider the origin table t and another table $\bar{t}$ which is obtained from t considering its rows as columns and its columns as rows.

If t passes both the relational and header filter, table columns are considered as candidate attribute lines to extract attributes from. Similarly, if $\bar{t}$ passes both the relational and header filter, the table rows that are considered as candidate attribute lines. It can happen that both columns and rows are considered as candidate attribute lines. The latter are then passed to the attribute line filter.

3.2 Attribute line filter

The attribute (name and values) is extracted from a column of the table or a row of the table. Let a be the first cell of the line (row or column) and V be the rest of the cells of the line. A conform attribute line should contain an attribute name in the first cell and attribute values in the rest of the cells.

Typically, attribute names do not have much punctuation except of the colon and parenthesis. They rarely contain numbers. On the other hand, attribute values are usually in the same format (number, string, date) and their length is similar. Based on the above observations, we define the following features for the attribute line filter:

- presence of punctuation (except colon and brackets) in a
- presence of numbers in a; a is an English stop word
- length (char) of a; length (words) of a
- average and standard deviation of the length (char) of values: μ, σ
- $\sharp$values $v \in V$ with $|len(v) - \mu| > 2\sigma$
- $\sharp$values $v \in V$ with $|\sigma < len(v) - \mu| \leq 2\sigma$
- $\sharp$values $v \in V$ with $|\sigma > len(v) - \mu|$
- data conformity : $\frac{max_{T \in int,string,date}(\sharp values\_of\_type(T)}{|V|}$

These features are then used to learn a rule-based classifier from a training set of human classified attribute lines described later in the experimental setup. Once candidate attributes are filtered, we rank the remaining set as explained in the following section.

3.3 Relevance

It is not easy to tell whether an attribute is relevant for a given instance. There are many tables relevant to the instance where the instance is not even present in its cells. We propose combining different features to estimate a relevance score for attributes. There features include a score on the match of the instance on the table, document relevance and external evidence from DBPedia, Wikipedia and Web search. We use a simple linear combination of these features to rank attributes. Each of the relevance features is described below.

Table match: Attributes should be extracted from tables that are relevant for the instance. A necessary condition but not sufficient for a table to be relevant is to be present in a relevant document for the instance. In some tables, the instance appears explicitly. Such tables might be better candidates for extraction. The table match feature will be described below. It is intended to measure at which extent the table matches the instance (query).

Let a be an attribute extracted from a table T for an instance i. The match of an instance within a table cell $T_{x,y}$ is measured with the cosine distance among the terms of the instance and the terms of the table cell. Let i and c be the vector representation of the instance and the table cell content. We have $i = (w_{1,i}, w_{2,i}, ..., w_{n,i})$ and $c = (w_{1,c}, w_{2,c}, ..., w_{n,c})$. Each dimension corresponds to a separate term. If a term occurs in the instance, its value is 1 in i. If it occurs in the table cell content, its value is 1 in c. The match is computed with the cosine similarity among the vectors.
The table match score is computed as the maximal match within table cells:

$$match(i, T) = \max_{T_{x,y} \in T} cos(i, T_{x,y})$$

However an attribute name is unlikely to be present in the same line (row or column) with the instance name, while the relevant attribute value is likely to appear in the same line with the instance. The latter can be also observed in table in figure 2. We will define the shadow area of a cell O as the set of cells in the same row and same column with O. But, there are some exceptions to this rule. We call headline cells, the ones that have are spanned in one line cell ($colspan > tablewidth$)[2] such as "Facts" and "Population" in table 2, figure 2. Headline cells usually act as titles that introduce parts of the table. We consider that the headline cells are not part of the shadow of another cell (see figure 3). We define the match score of an attribute as the difference between the table match score and the shadow match score.

$$match(a, i, T) = match(i, T) - match(i, shadow(a))$$

where

$$match(i, shadow(a)) = \max_{T_{x,y} \in shadow(a)} cos(i, T_{x,y})$$

[2] The colspan is an HTML attribute that defines the number of columns a cell should span.

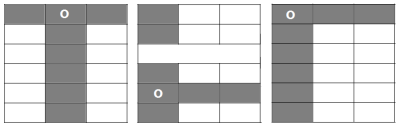


Fig. 3. The shadow for a cell **O**, 3 cases

Document relevance (*drel*) : If a document is relevant for the instance, the tables within the document are likely to be relevant for the instance. We should though take into account the document relevance. More precisely, let $\sharp results$ be the number of retrieved results for an instance i and $rank$ be the rank of a document d within this list. We compute $drel(d, i) = \frac{\sharp results - rank}{\sharp results}$.

Search hits: Search hits is a feature that has already been used for attribute extraction [21, 15]. It corresponds to the number of search results returned by a Web search engine to a query "attribute of instance" within double quotes (successive ordering of terms is obligatory, e.g. "capital of France"). As done in literature, we use the logarithm (base 10) of the search hits count. To normalize, we used the above observation. Few attributes score higher than 6 i.e $log_{10}(search\_hits\_count(\text{a of i})) > 6$. All the attributes that score higher than 6 were given a score of 1, the other scores were normalized by 6. Doing so, we have all scores in the interval $[0, 1]$.

DBPedia feature: DBPedia represents a large ontology of information which partly relies on information extracted from Wikipedia. Given an instance i and an attribute a, the DBPedia feature is equal to 1 if a is found as an attribute of i in DBPedia.

Wikipedia feature: Although information in DBPedia is much more uniform, there exist many attributes in Wikipedia *infobox* tables which are not present in DBPedia. *Infobox* tables are available for many Wikipedia pages. They contain lists of attributes for the page. Given an instance i and a candidate attribute a, we set the Wikipedia feature to 1 if a can be found in the infobox of a page for i in Wikipedia.

4 Experimental setup

Data set: To build our dataset, we first chose a set of classes and then 30 instances per class. To choose instances and classes, we use sampling to avoid biases. 5 participants had to write down 10 classes each. Classes could be broad (e.g. Countries) or specific (French speaking countries). We sampled 20 classes out of the 50. This is a reasonable amount as in state of the art approaches [16, 21, 3, 20, 13], the number of assessed classes varies from 2-25 classes.
Similarly for each class, we asked the 5 participants to write down 10 instances. Sampling and removing duplicates we obtained 10 instances per class. We do not apply any specific criteria for selecting instances.

This is the list of classes: *rock bands, laptops, american universities, hotels, software, british army generals, chancellors of German, American films, IR pa-*

pers, SLR cameras, 2000 novels, Nirvana songs, Nissan vehicles, programmable calculators, countries, drugs, companies, cities, painters, mobile phones. The entire dataset is in http://www.irit.fr/∼Arlind.Kopliku/FORCEdataset.txt .

General setup: For each instance of the dataset (200 instances), we retrieved top 50 search results using the Yahoo! BOSS API. These pages are used as a seed to extract tables and attributes. For each table, we apply the three filters. The remaining attribute lines are used as candidate attributes, which are ranked with the relevance features.

Learning filters: The three filters correspond to rule-based classifiers. They were trained using the previously mentioned features using the WEKA package [17]. From the extracted tables, we randomly selected a sample of 3000 tables (with more than 1 row and more than 1 column) which were judged by 3 assessors. For each table assessors had to tell, if the table is relational. If "yes" they had to tell if the table is oriented vertically or horizontally and whether the table has a header.

Similarly, we choose a sample of 3000 random attribute lines from our dataset of tables. They are as well assessed from our assessors. For each attribute line, the assessor has to tell if it is a conform attribute line i.e. it contains an attribute name and attribute values.

Similarly to [7], we cross-validated the trained classifiers by splitting the human-judged dataset into five parts. Each time four parts are used training and the fifth for testing. We trained five different classifiers, rotating the testing set each time. Performance numbers are averaged from the resulting five tests.

Ranking: Candidate attributes are ranked based on the relevance features. To measure the performance of ranking, the 30 top ranked attributes for each instance were assessed. 5 human participants shared this task assessing each a disjoint set of attributes.

Assessments were binary. An attribute is assigned 1 if it is *relevant* for the instance it was extracted for. It were assigned 0 otherwise. During evaluation, they could access the source page of the attribute or other sources (Web search, dictionary) to make their decision. Attributes were shown in alphabetical order to avoid any bias.

5 Evaluation results

This section is about experimental results. First, we present the performance of the three filters. Then we analyze the performance of attribute retrieval.

5.1 Filtering

First, we analyzed all retrieved tables for all instances in our dataset. We found that only 16.9% of the tables had more than one row and more than one column. Within the 3000 tables that were assessed only 23% were considered relational. We can thus estimate a concentration of 3.9% relational tables in the retrieved Web pages. Now, we will analyze the effect of the filters. It is important to point

out that our goal is not to compare with the results of Cafarella et al. [7], but to integrate their work and show its effect on attribute retrieval.

Relational filter: As we mentioned earlier, we learn separately a classifier for relational tables that are oriented horizontally and another classifier for relational tables that are oriented vertically. Results are shown in table 2 aside with the results obtained by Cafarella et al. [7].

Table 2. Precision and recall for the relational filter

Horizontal			Vertical			Cafarella		
	prec.	recall		prec.	recall		prec.	recall
yes	0.50	0.82	yes	0.38	0.81	yes	0.41	0.81
no	0.98	0.94	no	0.98	0.95	no	0.98	0.87

We tuned classification for high recall over positives. The performance of our classifiers is similar to Cafarella et al. The classifier of relational tables oriented horizontally retains 82% of the true positives and it filters out 94% of the true negatives. Similarly, the classifier of relational tables oriented vertically retains 81% of the true positives and it filters out 95% of the true negatives. After this filtering step, we will have about 45% of relational tables within our corpus out of an initial estimated concentration of about 3.9%.

Table 3. Precision and recall for the header filter in relational tables

Horizontal			Vertical			Cafarella		
	prec.	recall		prec.	recall		prec.	recall
yes	0.96	0.95	yes	0.76	0.89	yes	0.79	0.84
no	0.63	0.70	no	0.87	0.73	no	0.65	0.57

Header filter: In table 3, we show results for the header filter. Results are better than those of Cafarella et al. In particular, we found that most of the horizontally oriented relational tables have headers. They usually have two to three columns and are easier to classify.

The header classifier for relational tables oriented horizontally retains 95% of the true positives with a precision of 96%. Although, the header classifier of relational tables oriented vertically is less performant, it retains 89% of the true positives with a precision of 76%. After this filtering step, about 87.5% of the relational tables will have headers.

Table 4. Precision and recall for the attribute line filter

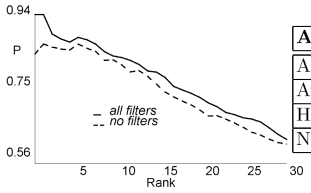
	prec.	recall
yes	0.56	0.95
no	0.69	0.55

Attribute line filter: As well as the other filters, the attribute line filter is tuned for recall over positives. Results are shown in table 4. This filter retains 95% of the correct attribute lines, while it filters out about 55% of the incorrect

attribute lines. It clearly helps in filtering out useless attribute lines at the cost of 5% of correct attribute lines.

5.2 Attribute retrieval

Precision at rank: Attributes are ranked with a linear combination of the relevance features. Results are shown in figure 4. They are promising, especially when all three filters are applied. At rank 10, we have a precision of about 83%. At rank 20 we have a precision of about 72%. This means that if we apply this ranking for query suggestion, we will have that about 8.3 correct suggestions within top 10 suggestions and about 14.4 correct suggestions among the top 20.



Approach	p@1	p@10	p@20	p@30
All filters	0.94	0.83	0.72	0.61
Attribute line filter	0.90	0.82	0.70	0.61
Header and relat. filter	0.89	0.83	0.70	0.59
No filters	0.84	0.81	0.68	0.59

Fig. 4. Impact of filters

In figure 4, we can compare results for the ranking when no filter is applied and when all 3 filters are applied. Clearly, filters improve results. The table in figure 4 allows us to analyze the impact of each filter. We can see that the attribute line filter has a significant impact in the ranking as well as the relational and header filter.

Comparison with existing approaches: We first compared our approach to the one based on Lexico-syntactic rules [3, 21, 16, 14] Lexico-syntactic rules are common for attribute extraction . We tested the lexico-syntactic extraction rules in our retrieval framework with our dataset. Concretely, we use the patterns "A of I" and "I's A" as done in [14, 16]. We collect candidate attributes for 10 instances of all classes. The top 50 search results for all instances are used as extraction seed for both techniques. The attributes extracted by the lexico-syntactic extraction method are ranked with the same scoring (excluding table match score which does not apply to lexico-syntactic rules).
Results are shown in table 5. Our method performs significantly better with 61% of relevant attributes at rank 30 against 33% for the lexico-syntactic rules.

Approach	p@1	p@10	p@20	p@30
Our approach	0.94	0.83	0.72	0.61
Lexico-syntactic rules	0.46	0.48	0.43	0.33

Table 5. Comparison with lexico-syntactic scores

We also compared both approach in term of recall, by considering only the 4 classes used to estimate recall. Lexico-syntactic rules have a lower recall, too.

For each class, lexico-syntactic rules identify 55 candidate attributes on average. Among these there are about 24 relevant attributes per class.

We can conclude that lexico-syntactic rules work well for certain applications such as queries, but they do not work well for long documents, especially in terms of precision.

Estimated recall: To estimate in a reasonable time the recall of our method, we randomly selected 4 classes namely "SLR cameras", "countries", "companies", "Nissan vehicles". For all instances of the class, our assessors evaluated all candidate attributes (the ones that are not filtered out).

We found an average of 918 distinct candidate attributes per class (SLR cameras 689, countries 1253, companies 804, vehicles 925). Among them, there are on average 256 relevant attributes per class (cameras 303, countries 213, companies 160, vehicles 347). This is a considerable amount of relevant attributes and it shows the potential of our method.

Comparison with DBPedia: We compared the recall of our approach with the one obtained using DBPedia, for the 4 classes mentioned above. To do so, we collected all Wikipedia pages related of the instances of each class. We then extracted attributes which are either present in DBPedia or in infoboxes from Wikipedia. We found an average of 25 distinct relevant attributes per class (cameras 8, countries 38, companies 37, vehicles 18). We can conclude that our approach has a high recall even if compared with quality and large sources such as DBPedia and Wikipedia.

6 Conclusions

We propose an attribute retrieval approach that can extract and rank attributes from HTML tables sparse in the Web. Our method is flexible and recall-oriented. Given an instance as a query, we retrieve attributes from top ranked search results. Then, we combine filtering and ranking to obtain a list of attributes ranked by relevance.

We combine a total three filters. Two of them are already known in literature. They are applied to HTML tables to filter out non relational tables and tables without header. In addition, we propose a third filter which is specific to attributes. All three filters are shown to have a high recall over povitives and they filter out a huge amount of useless data. The remaining data is candidate attributes which are ranked with relevance features. Our ranking algorithm combines document match, table match and other external evidence. The experimental setup shows that we can rank attributes with a reasonable precision and a high recall. Our approach outperforms lexico-syntactic rules and it provides a much larger quantity of attributes than quality sources such as DBPedia and Wikipedia (for our dataset).

For future work, we will investigate additional relevance features. We will focus more on attribute values and try combining different attribute acquisition techniques in a large-scale domain independent attribute retrieval framework.

References

1. E. Alfonseca, M. Pasca, and E. Robledo-Arnuncio. Acquisition of instance attributes via labeled and related instances. In *SIGIR '10*, pages 58–65, New York, NY, USA, 2010. ACM.
2. A. Almuhareb and M. Poesio. Attribute-based and value-based clustering: An evaluation. In *EMNLP*. ACL, 2004.
3. K. Bellare, P. P. Talukdar, G. Kumaran, O. Pereira, M. Liberman, A. Mccallum, and M. Dredze. Lightlysupervised attribute extraction for web search. In *Proceedings of Machine Learning for Web Search Workshop, NIPS 2007*, 2007.
4. O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev. Beyond basic faceted search. In *WSDM '08*, pages 33–44, New York, NY, USA, 2008. ACM.
5. M. J. Cafarella, M. Banko, and O. Etzioni. Relational Web Search. Technical report, University of Washington, 2006.
6. M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549, 2008.
7. M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the Relational Web. In *WebDB*, 2008.
8. C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of web information extraction systems. *IEEE Trans. on Knowl. and Data Eng.*, 18:1411–1428, October 2006.
9. H.-H. Chen, S.-C. Tsai, and J.-H. Tsai. Mining tables from large scale html texts. In *COLING '00*, pages 166–172, USA, 2000.
10. V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB '01*, pages 109–118, USA, 2001.
11. A. Kopliku. Aggregated search: From information nuggets to aggregated documents. In *CORIA RJCRI 09, Toulon, France*, 2009.
12. A. Kopliku, K. Pinel-Sauvagnat, and M. Boughanem. Retrieving attributes using web tables. In *Joint Conference on Digital Libraries 2011, Ottawa, Canada*, 2011.
13. M. Pasca and B. V. Durme. What you seek is what you get: Extraction of class attributes from query logs. In *IJCAI*, pages 2832–2837, 2007.
14. M. Pasca and B. V. Durme. Weakly-supervised acquisition of open-domain classes and class attributes from web documents. In *ACL*, pages 19–27, 2008.
15. A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *HLT '05*, pages 339–346, Stroudsburg, USA, 2005. ACL.
16. K. Tokunaga and K. Torisawa. Automatic discovery of attribute words from web documents. In *IJCNLP 05', Jeju Island, Korea*, pages 106–118, 2005.
17. I. H. Witten and E. Frank. Data mining: practical machine learning tools and techniques with java implementations. *SIGMOD Rec.*, 31:76–77, March 2002.
18. T.-L. Wong and W. Lam. A probabilistic approach for adapting information extraction wrappers and discovering new attributes. In *ICDM '04*, pages 257–264, Washington, DC, USA, 2004. IEEE Computer Society.
19. T.-L. Wong and W. Lam. An unsupervised method for joint information extraction and feature mining across different web sites. *Data Knowl. Eng.*, 68:107–125, January 2009.
20. F. Wu, R. Hoffmann, and D. S. Weld. Information extraction from wikipedia: moving down the long tail. In *KDD '08*, pages 731–739, New York, USA, 2008.
21. N. Yoshinaga and K. Torisawa. Open-domain attribute-value acquisition from semi-structured texts. In *Proceedings of the workshop on Ontolex*, pages 55–66, 2007.