Towards a Framework for Attribute Retrieval

Arlind Kopliku
IRIT
Paul Sabatier University
Toulouse, France
Arlind.Kopliku@irit.fr

Mohand Boughanem
IRIT
Paul Sabatier University
Toulouse, France
Mohand.Boughanem@irit.fr

Karen Pinel-Sauvagnat
IRIT
Paul Sabatier University
Toulouse, France
Karen.Sauvagnat@irit.fr

ABSTRACT

In this paper, we propose an attribute retrieval approach which extracts and ranks attributes from HTML tables. We distinguish between class attribute retrieval and instance attribute retrieval. On one hand, given an instance (e.g. University of Strathclyde) we retrieve from the Web its attributes (e.g. principal, location, number of students). On the other hand, given a class (e.g. universities) represented by a set of instances, we retrieve common attributes of its instances. Furthermore, we show we can reinforce instance attribute retrieval if similar instances are available.

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 is shown to have a high recall and a reasonable precision. Moreover, it outperforms state of the art techniques.

General Terms

Measurement, Experimentation, Algorithms

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 one or more extracts of information coming from one or more documents. For instance, the query "features of Mac Book" is not asking for all pages speaking of Mac Books rather than for a list of features.

Inspired by the work in [6], we distinguish three types of information needs that can be answered differently:

- *query by attribute* ("GDP of UK", "address of Hotel Bellagio")
- *query by instance* ("Samsung Galaxy S", "Scotland", "Oscar Wilde")
- *query by class* ("Toshiba 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 commercial 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 [4], faceted search [5] and aggregated search [15]. 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 instances/classes.

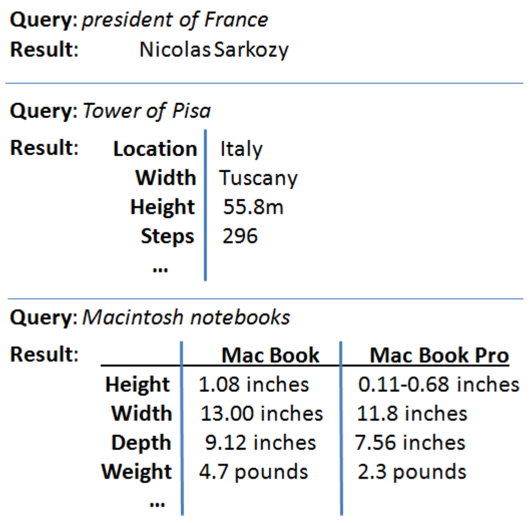
In this paper, we deal with attribute retrieval which falls in the above defined framework. More precisely, we deal with three attribute retrieval problems. First, we retrieve relevant attributes for one given instance (e.g. "University of Strathclyde"). Second, we retrieve attributes for a given class (e.g. "universities") represented as a set of instances. Third, we retrieve attributes for one instance when some other similar instances are given (from the same class).

Attributes are extracted from HTML tables in the Web. There are some main reasons behind this choice. First, tables are widespread across the Web. Second, many tables contain relational data (i.e. data in a relational form) [8]. Third, tables have already been used sucessfully for attribute extraction [10, 9].

However, the task is not easy. Many of the HTML tables are used for layout design and navigation. In [8], authors estimate that only about 1% of the Web tables contain relational data. Furthermore, some relational tables have no

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.
Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...$10.00.



Query: *president of France*
Result: Nicolas Sarkozy

Query: *Tower of Pisa*
Result:

Location	Italy
Width	Tuscany
Height	55.8m
Steps	296
...	

Query: *Macintosh notebooks*
Result:

	Mac Book	Mac Book Pro
Height	1.08 inches	0.11-0.68 inches
Width	13.00 inches	11.8 inches
Depth	9.12 inches	7.56 inches
Weight	4.7 pounds	2.3 pounds
...		

Figure 1: Examples of relational search results

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. For every given instance, 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. Then, depending on the considered problem, we rank candidate attributes with a combination of relevance features.

Our approach integrates the work of Cafarella et al. [8]. 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 [16, 17], which shows that we can rank attributes using a linear combination of relevance features such as table match, document relevance and external evidence from Web search, DBPedia and Wikipedia. In our previous work, we have focused on instance attribute retrieval, while in this paper we deal with instance and class attribute retrieval with an extended experimental setup. To the best of our knowledge, this is the first work that put together attribute retrieval at instance and class level through the use Web tables. We also introduce a technique to reinforce attribute retrieval and we show some initial work on attribute value retrieval (which is not the main goal of this paper). The combination of these techniques contributes to build a framework for attribute retrieval at Web-scale.

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

Attributes can find different uses. They can be used for the summarization (representation) of content [29, 18], to assist search such as for query suggestion [4] and faceted search [5], or for question answering [13]. Attribute acquisition methods can be domain-independent or domain-dependent. Among domain-dependent approaches, we can mention approaches that focus on products. In this domain, attributes have been used to improve product search and recommendation [18, 22], but also to enable data mining [27].

Attribute retrieval provides another granularity in Web search. This can interest communities that propose a more focused access to information or communities that envision aggregating pieces of information such as aggregated search [19, 15].

To acquire attributes from the Web, it is common to use decoration markup [30, 27, 11] and text [4, 24, 20]. HTML tags (for tables, lists and emphasis) have also been shown to help for attribute acquisition [30, 27]. Wong et al. [27] 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, 21] 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 [22], 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. [8, 7] show we can identify billions of relational tables in the Web. In [10], Chen et al. identify attributes using column (or row) similarities. Another common technique to extract attribute from tables is through wrapper induction [9, 11, 26]. 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 the work in [1] and [7, 8]. 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 [7, 8], in that we do not use tables for relation search. We make use of learnings in [7, 8], 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 [7, 8].

We retrieve attributes for instances and classes. A clear distinction among class attributes and instance attributes can be found in [1]. They represent the class as a set of instances and they extract class attributes using class instances. We use the same setup to retrieve attributes for classes.

Class instances can also be acquired automatically. Here, we can mention the work done in named entity recognition [12, 23]. Class instances can be also acquired from hierarchies such as the ones that derive from quality sources such as Wikipedia or DBPedia. Hearst [14] shows that we can extract class instances using the pattern "C such as LI" (e.g. cities such as London, Paris and Rome). However, class acquisition methods need to be improved to have high recall and reasonable precision at Web scale.We prefer separating the class acquisition issues from class attribute retrieval. In this paper, we thus consider that class instances are given.

3. ATTRIBUTE RETRIEVAL

We consider three different attribute retrieval problems detailed below.

Problem 1: 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.**

Retrieving attributes for whatever instance can be quite complex. There is no knowledge base with a list of attributes for the instance, but the Web with its size can help us retrieve many of them. We need methods that can work for most instances and are domain-independent. In this paper, we aim retrieving attributes at Web scale using HTML tables which are known to be a huge source for relational data.

Due to the complexity of the problem, we deal in this paper with attribute names only. To illustrate, given the instance "France", we want to retrieve its attributes such as "capital", "president", "area", "population", "GDP", "GDP per capita", etc. We do not focus on attribute values (e.g. "Paris" is the value of the attribute "capital" for the instance "France"). Although attribute names and values are usually met in the same line in tables (row or column), selecting the correct attribute value is not easy. We will discuss later shortly some of our experiments regarding attribute values, leaving for a future publication the entire study on attribute value retrieval.

We extract attributes from the lines (rows or columns) of HTML tables from the Web. To do so, we use the following procedure. Initially, we issue the instance i as a query to a search engine and we retrieve tables from the retrieved documents. These tables are potentially relevant for the instance and are used to extract attributes from. However, the problem is far away from being solved. Tables in the Web are quite heterogeneous and many of the retrieved tables are partially or not 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 [8]. 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 with respect to their relevance (depending on the retrieval problem). 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 all retrieved documents. Extracted attributes are ranked with a relevance score $\phi(a, i)$ which is applied over an instance i and an attribute a. $\phi(a, i)$ is a simple linear combination of relevance features which was shown to work well to rank attributes in previous work [16]. It will be described in details in section 3.3.

Problem 2: We consider that the query is a set of instances I that represents a class. For example the class "'countries"' can be represented by the set: "'France"', "'Italy"', "'UK"', etc. The goal is to retrieve common relevant attributes for the class. The relevant attributes should not be specific to one instance only, but to all the class.

For every instance $i \in I$, we repeat the same procedure as in problem 1 to retrieve candidate attributes. Then, we rank the set of all candidate attributes for all instances. The relevance score for the class (set of instances I) is :

$$\phi(a, I) = \frac{\sum_{i \in I} \phi(a, i)}{|I|} \quad (1)$$

In other terms, a relevant attribute for the class is likely to be present in many instances of the class. Though, we average the relevance score across all instances.

Problem 3: We reconsider retrieving attributes for an instance i, but we consider that a set of instances I_+ of the same class are given. The hypothesis is that having more instances can improve attribute retrieval due to the simple fact that similar instances share common attributes. However, even if most relevant attributes are shared among instances of the same class, many exceptions can occur. For example, "queen" is relevant for the instance "UK", but it is irrelevant for the instance "USA". To overcome this problem, when it comes to ranking, we privilege attributes retrieved for the instance i using only the instances in I_+ to reinforce ranking.

In practical terms, for every given instance (i, or $i_+ \in I_+$), we repeat the same procedure as in problem 1 to retrieve candidate attributes. The relevance score for the instance i is then computed as follows:

$$\phi'(a, i) = \begin{cases} 0 & \text{if } \phi(a, i) = 0 \\ \phi(a, i) + \phi(a, I_+) & \text{otherwise} \end{cases} \quad (2)$$

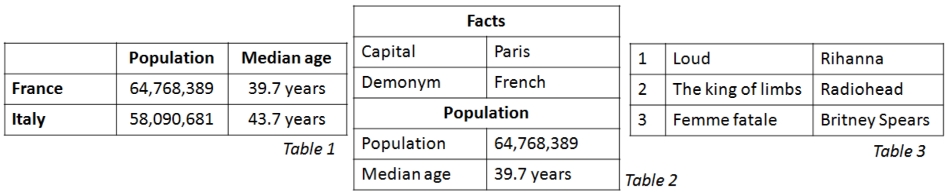
In the next sections, we will explain how our filters (*relational*, *header* and *attribute line*) work and which are relevance features used to compute $\phi(a, i)$.

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 [8]. 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.

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 (lowercase, 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 ver-



	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

Figure 2: Examples of interesting tables

	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

tically, 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 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

Let a be the first cell of the line (row or column) and V be the rest of the cells of the line. We consider that a conform attribute line should contain an attribute name in the first cell and attribute values in the rest of the cells. Attribute values can correspond to one instance or multiple instances.

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 colons, 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
- ♯values $v \in V$ with $\lvert len(v) - \mu\rvert > 2\sigma$
- ♯values $v \in V$ with $\lvert\sigma < len(v) - \mu\rvert \leq 2\sigma$
- ♯values $v \in V$ with $\lvert\sigma > len(v) - \mu\rvert$
- 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 $\phi(a, i)$ for a candidate attribute a and an instance i. These features include a score on the match of the instance on the table, document relevance and external evidence from DBPedia, Wikipedia and Web search. $\phi(a, i)$ is a simple linear combination of these features[2]. Relevance features are described below.

Table match: **Tables from which attributes are extracted should be 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. In fact, we analyzed samples of Web tables and we observed a higher concentration of relevant attributes within tables that match the instance. The table match feature will measure at which extent the table matches the instance.

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

[2]This combination has been shown to be the most effective way to combine features in previous experiments.

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:

$$match(i, T) = \max_{T_{x,y} \in T} cos(i, T_{x,y}) \quad (3)$$

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 (1) 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. There are some exceptions to this rule. We call headline cells, the ones that have are spanned ($colspan > 1$)[3] that cover the entire table width such as the ones in table 2 in 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).

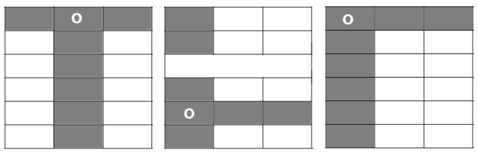


Figure 3: The shadow for a cell O, 3 cases

We define the match score of an attribute as the difference between the table match score and the shadow match score.

$$match(a, i, T) = match(i, T) - match(i, shadow(a)) \quad (4)$$

where

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

Document relevance : 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:

$$drel(d, i) = \frac{\sharp results - rank}{\sharp results} \quad (5)$$

Search hits: Search hits is a feature that has already been used for attribute extraction [30, 22]. 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 following observation. Few attributes score higher than 6 i.e $log_{10}(search\_hits\_count(a\ of\ i)) > 6$. 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 $[0, 1]$.

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

DBPedia feature: DBPedia represents a large ontology of information which partly relies on information extracted from Wikipedia [3]. Given an instance i and an attribute a, the DBPedia feature $DBPedia(a, i)$ 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 $Wikipedia(a, i)$ to 1 if a can be found in the infobox of a page for i in Wikipedia.

Other features: It is difficult to find features which help rank attributes that are domain-independent and apply at large-scale. We excluded from the relevance features the *frequency feature*. This feature is meant to measure how often an attribute appears within tables of relevant documents. Intuitively, we can think that a relevant attribute will repeat more frequently than non-attributes or irrelevant attributes. We observed that candidate attributes that repeat the most are the ones that are used in ads or forms. Thus, candidate attributes such as "login", "search", "previous", etc. were the ones that were favored by this feature. To tackle this issue, we developed a stop-word list for attributes, but even with this list, previous experiments did not show the interest of the frequency feature. This may be due to the fact that our list is still incomplete and need to be built using the whole web. Another alternative is to divide frequency by the frequency of the candidate attribute in the entire collection of documents being considered (analogous to *tf-idf*). In other terms, an attribute has to be more frequent in relevant documents than in a random sample of documents. Once again, preliminary experiments did not demonstrate the validity of the feature. We thus leave for future work a correct integration of this feature in the evaluation of $\phi(a, i)$.

Combination of features At last, $\phi(a, i)$ is evaluated according to the following formula:

$$\begin{aligned}\phi(a, i) &= match(a, i, T) + drel(d, i) \\ &+ log_{10}(search\_hits\_count(a\ of\ i)) \\ &+ DBPedia(a, i) + Wikipedia(a, i) \quad (6)\end{aligned}$$

with T the table from which a was extracted and d the document containing T.

4. EXPERIMENTAL SETUP

Data set: To evaluate our approaches, we built a dataset of classes and instances. First we chose a set of classes and then 10 instances per class. To choose instances and classes, we used 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 [24, 30, 4, 28, 20], the number of assessed classes varies from 2-25 classes.
Similarly for each selected class, we asked the 5 participants to write down 10 instances. Sampling and removing duplicates we obtained 10 instances per class.

This is the list of classes: *rock bands, laptops, american*

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

Table 2: Precision and recall for the relational filter

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

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

universities, hotels, software, british army generals, chancellors of German, American films, IR papers, SLR cameras, 2000 novels, Nirvana songs, Nissan vehicles, programmable calculators, countries, drugs, companies, cities, painters, mobile phones. The entire dataset can be found 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.

For the problem 1, candidate attributes are ranked using $\phi(a, i)$. For the second problem, attributes are ranked using $\phi(a, I)$ For the third problem attributes are ranked using $\phi(a, i)$ reinforced with $\phi(a, I_+)$ as described before.

Learning filters: The three filters correspond to rule-based classifiers. They were trained using the previously mentioned features using the WEKA package [25]. 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 [8], 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 and depending on the problem being treated. 5 human participants evaluated this task assessing each a disjoint set of attributes. To measure the performance of ranking we had to measure top attributes retrieved for the three problems. To assess the performance over the first and third problem, assessors judged the 30 top ranked attributes for each instance being the target. To assess the performance for the problem 2, top 30 attributes were assessed for all classes of our dataset.

Assessments were binary. An attribute is assigned 1 if it is *relevant* for the instance/class it was retrieved for[4]. It were assigned 0 otherwise. During evaluation, assessors 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.

[4]We recall that only attribute names were assessed.

5. EVALUATION RESULTS

This section is about experimental results. First, we present the performance of the three filters. We then 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 within the entire set of 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. [8], 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. [8].

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%.

Header filter: In table 3, we show results for the header filter. Results are better than those obtained by Cafarella et al. This can be explained with the fact that we use top search resuts which are presumably of better quality than random Web pages. 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.

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.

	prec.	recall
yes	0.56	0.95
no	0.69	0.55

Table 4: Precision and recall for the attribute line filter in relational tables

In the next section, we will discuss on the effectiveness of the filters for the attribute retrieval task.

5.2 Instance attribute retrieval

For the first problem considered (instance attribute retrieval), attributes are ranked with $\phi(a, i)$. In figure 4, we show precision at rank averaged over all instances. We can say that results are promising. 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.

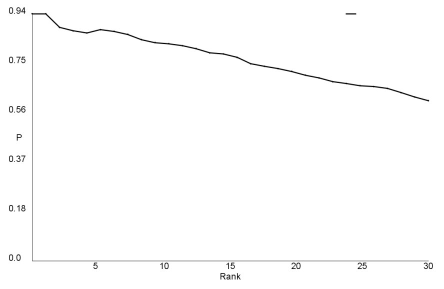


Figure 4: Precision at rank for instance attribute retrieval

Table 5 allows us to analyze the impact of each filter for attribute retrieval. Symbol * after the results of the first row indicates statistical significance using a paired t-test at p< 0.05 (all filters against no filters). We can see that the attribute line filter has a significant impact in the ranking as well as the relational and header filter. Combining all three filters provides the best performance. This is probably because the filters remove useless tables and table lines, making it easier to rank attributes.

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

Table 5: The impact of filters

Attribute values: The above results concern only attribute names. We will introduce here some of our work on attribute values. Given a set of candidate attribute lines, we can rank attribute values in different ways. An easy way to rank is the following. First, we rank attribute lines with $\phi(a, i)$. Then, if the attribute line has two rows (or columns), we use the second cell as attribute value. If the instance name appears ortogonally with some cell (except the first) in the attribute line, we select this cell as the attribute value for the attribute. Otherwise, we consider that the attribute value is the union of all cells (except the first). We call this attribute as multi-value. This simple method was shown to acquire correctly attribute values for 66% of relevant attributes in our dataset. These promising results will be completed by a complete analysis of attribute values retrieval in future work.

5.3 Class attribute retrieval

Figure 5 shows precision at rank for class attribute retrieval (problem 2) when we represent the class as a set of 10 instances. We retrieve attributes for each of the 10 instances of the class and we rank with the function $\phi(a, I)$ defined in section 3 for the second problem. As expected, results are better than for instance attribute retrieval. We have precision of 0.95 at rank 10, a precision of 0.84 at rank 20 and a precision of 0.77 at rank 30. For the same problem, if we use 5 instances per class, we obtain precision 0.89 at rank 10, precision 0.76 at rank 20 and precision 0.66 at rank 30. In general, we noticed that class attribute retrieval performance improves with the increase of available instances.

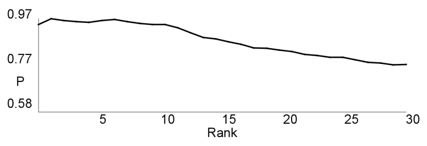


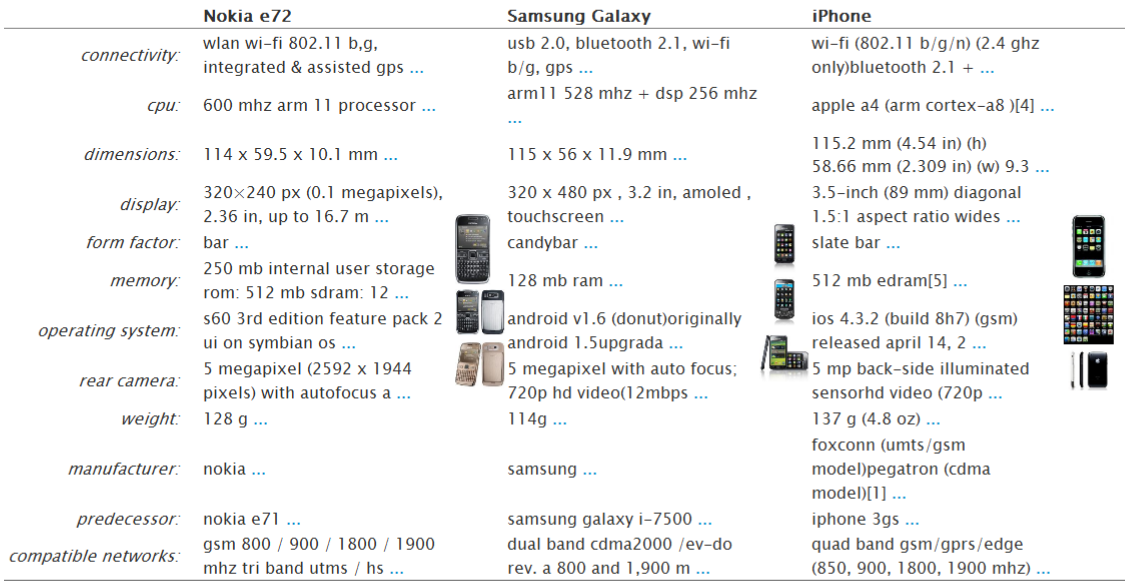
Figure 5: Precision at rank for class attribute retrieval

In figure 6, we show how results look like in one Web application we built. Here, we rank attributes names and values with $\phi(a, I)$ and then we select attribute values as described in section 5.2. Then we add some photos using image search, that can be seen as some decoration in our application. The example in figure 6 concerns a query composed of three instances from the class "mobile phones".

We also analyzed results class by class. Figure 7 shows precision at rank 30 by class. We can see that attribute retrieval can vary across classes. For some classes in our dataset, performance is lower such as for "drugs" and "programmable calculators", while for others precision remains high such as for "british army generals", "chancelors", "countries". The quality of attribute retrieval by class depends on the quality of attribute retrieval for every instance of the class. Results are relatively heterogeneous. Some instances are ambiguous. For some others, returned search results are slightly relevant. For some instances or class of instances, there is more tabular data than for other within search results. Furthermore, for some classes there exist more relevant attributes than for others.

5.4 Reinforced attribute retrieval

To reinforce instance attribute retrieval (problem 3), for each target instance we use the 9 other instances of the



	Nokia e72	Samsung Galaxy	iPhone
connectivity:	wlan wi-fi 802.11 b,g, integrated & assisted gps ...	usb 2.0, bluetooth 2.1, wi-fi b/g, gps ...	wi-fi (802.11 b/g/n) (2.4 ghz only)bluetooth 2.1 + ...
cpu:	600 mhz arm 11 processor ...	arm11 528 mhz + dsp 256 mhz ...	apple a4 (arm cortex-a8)[4] ...
dimensions:	114 x 59.5 x 10.1 mm ...	115 x 56 x 11.9 mm ...	115.2 mm (4.54 in) (h) 58.66 mm (2.309 in) (w) 9.3 ...
display:	320×240 px (0.1 megapixels), 2.36 in, up to 16.7 m ...	320 x 480 px , 3.2 in, amoled , touchscreen ...	3.5-inch (89 mm) diagonal 1.5:1 aspect ratio wides ...
form factor:	bar ...	candybar ...	slate bar ...
memory:	250 mb internal user storage rom: 512 mb sdram: 12 ...	128 mb ram ...	512 mb edram[5] ...
operating system:	s60 3rd edition feature pack 2 ui on symbian os ...	android v1.6 (donut)originally android 1.5upgrada ...	ios 4.3.2 (build 8h7) (gsm) released april 14, 2 ...
rear camera:	5 megapixel (2592 x 1944 pixels) with autofocus a ...	5 megapixel with auto focus; 720p hd video(12mbps ...	5 mp back-side illuminated sensorhd video (720p ...
weight:	128 g ...	114g ...	137 g (4.8 oz) ...
manufacturer:	nokia ...	samsung ...	foxconn (umts/gsm model)pegatron (cdma model)[1] ...
predecessor:	nokia e71 ...	samsung galaxy i-7500 ...	iphone 3gs ...
compatible networks:	gsm 800 / 900 / 1800 / 1900 mhz tri band utms / hs ...	dual band cdma2000 /ev-do rev. a 800 and 1,900 m ...	quad band gsm/gprs/edge (850, 900, 1800, 1900 mhz) ...

Figure 6: Attribute retrieval results for 3 mobile phone instances

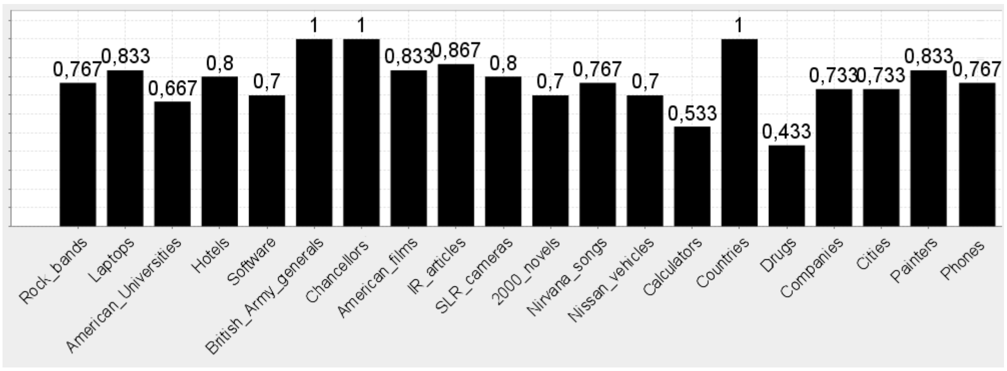


Figure 7: Precision at rank 30 by class

same class. Attributes are ranked combining both $\phi(a, i)$ and $\phi(a, I_+)$ as described earlier. The intuition is that the other instances can provide additional evidence of relevance for attributes.

In figure 8, we can see the impact of reinforcement on instance attribute retrieval. The lower curve corresponds to instance attribute retrieval without reinforcement, while the upper curve corresponds to the reinforced attribute retrieval. We can see that reinforcement improves significantly results[6].

We can conclude that having multiple instances of the same class helps attribute retrieval for instance queries. We can explain this phenomena due to the fact that similar instances have similar attributes. Promoting common attributes has a positive impact on retrieval.

5.5 Estimating recall

To estimate in a reasonable time the recall of our method, we used 4 classes randomly selected from our dataset namely "SLR cameras", "countries", "companies", "Nissan vehicles".

[6]The significance of the improvement was validated using a paired t-test with p<0.05 for P@10, P@20, P@30.

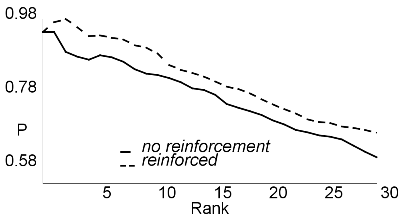


Figure 8: The impact of reinforcement

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. We can say that tables are a good source for attributes and that our filters keep a high concentration of relevant attributes.

It is difficult to estimate how many relevant attributes there are for a given class or instance as it is difficult to write down all of them. We used Wikipedia and DBPedia to estimate their coverage in terms of attributes and then compare it to our method. We used 10 instances for each of the 4 classes mentioned above which have Wikipedia pages and are present in DBPedia. 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). Although Wikipedia and DBPedia are quality and huge sources of information, they have a much lower recall than our method.

5.6 Comparison with state of the art

We also compared our approach to the ones based on lexico-syntactic rules [4, 30, 24, 21] 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 [21, 24]. We collect candidate attributes for 10 instances of all classes.

To compare we used the class attribute retrieval approach (problem 2). We use top 50 search results for all instances of our dataset 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 6. Our method performs significantly better with 61% of of relevant attributes at rank 30 against 33% for the lexico-syntactic rules. We recall that symbol * after the results indicates statistical improvement using a paired t-test ($p<0.05$).

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 6: 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 (against 256 for our approach).

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.

6. CONCLUSIONS

In this paper, we propose 3 approaches for attribute retrieval using HTML tables from the Web. Our approaches combine 3 filters and they rank attributes using relevance features. Results are promising both in terms of precision and recall.

We combine three filters. Two of them are already known in literature. They apply to HTML tables to filter out non relational tables and tables without header. In our experimental setup they prove their effectiveness. In addition, we propose a third filter which is specific to attributes. Similarly to the other two, this filter is shown to have a high recall over positives and to filter out a reasonable amount of useless data.

We treat 3 different problems for attribute retrieval. First, we retrieve attributes for a given instance. Then, we retrieve attributes at class level. Retrieval is shown to have a high recall and a reasonable precision for both problems. Then we propose a reinforced approach for instance attribute retrieval. Using similar instances of the same class to reinforce attribute retrieval is shown to improve significantly results.

Our approaches have different advantages which were shown in this paper. First, they can be applied to whatever instances. They work at Web scale and they have a high recall. Furthermore, they outperform state of the art techniques for the same purpose.

We envision for future work the completition of the attribute retrieval framework with other recall-oriented or precision-oriented techniques. We believe that relations between classes, instances and attributes can be further explored for a better attribute retrieval. We will also explore more deeply the issue of attribute value retrieval.

7. 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] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, pages 722–735, 2007.

[4] 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.

[5] 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.

[6] M. J. Cafarella, M. Banko, and O. Etzioni. Relational Web Search. Technical report, University of Washington, 2006.

[7] 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.

[8] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the Relational Web. In *WebDB*, 2008.

[9] 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.

[10] H.-H. Chen, S.-C. Tsai, and J.-H. Tsai. Mining tables from large scale html texts. In *Proceedings of the 18th conference on Computational linguistics - Volume 1*, COLING '00, pages 166–172, Stroudsburg, PA, USA, 2000. ACL.

[11] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner:

Towards automatic data extraction from large web sites. In *VLDB '01*, pages 109–118, San Francisco, USA, 2001. Morgan Kaufmann.

[12] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the Web: an experimental study. *Artif. Intell.*, 165(1):91–134, 2005.

[13] M. Fleischman, E. Hovy, and A. Echihabi. Offline strategies for online question answering: answering questions before they are asked. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 1–7, 2003.

[14] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

[15] A. Kopliku. Aggregated search: From information nuggets to aggregated documents. In *CORIA 09 RJCRI Rencontre Jeunes Chercheurs en Recherche d'Information, Toulon, France*, 2009.

[16] A. Kopliku, K. Pinel-Sauvagnat, and M. Boughanem. Attribute retrieval from relational web tables. In *Symposium on String Processing and Information Retrieval, SPIRE*, page to appear, 2011.

[17] A. Kopliku, K. Pinel-Sauvagnat, and M. Boughanem. Retrieving attributes using web tables. In *Joint Conference on Digital Libraries JCDL*, pages 397–398, 2011.

[18] Z. Nie, Y. Ma, S. Shi, J.-R. Wen, and W.-Y. Ma. Web object retrieval. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 81–90, New York, NY, USA, 2007. ACM.

[19] C. Paris, S. Wan, and P. Thomas. Focused and aggregated search: a perspective from natural language generation. *Information Retrieval Journal*, 44(3), 2010.

[20] 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.

[21] M. Pasca and B. V. Durme. Weakly-supervised acquisition of open-domain classes and class attributes from web documents and query logs. In *ACL*, pages 19–27, 2008.

[22] A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *HLT '05*, pages 339–346, Stroudsburg, USA, 2005. ACL.

[23] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 697–706, New York, NY, USA, 2007. ACM.

[24] K. Tokunaga and K. Torisawa. Automatic discovery of attribute words from web documents. In *IJCNLP 05', Jeju Island, Korea*, pages 106–118, 2005.

[25] I. H. Witten and E. Frank. Data mining: practical machine learning tools and techniques with java implementations. *SIGMOD Rec.*, 31:76–77, March 2002.

[26] 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.

[27] 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.

[28] 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. ACM.

[29] M. Yoshida and K. Torisawa. A method to integrate tables of the world wide web. In *In Proceedings of the International Workshop on Web Document Analysis (WDA 2001*, pages 31–34, 2001.

[30] 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.