SELF-SUPERVISED PRODUCT FEATURE EXTRACTION USING A KNOWLEDGE BASE AND VISUAL CLUES

Rémi Ferrez[1], Clément de Groc[1,2] and Javier Couto[1,3]
[1]*Syllabs, Paris, France*
[2]*Univ. Paris Sud & LIMSI-CNRS, Orsay, France*
[3]*MoDyCo, UMR 7114, CNRS-Université de Paris Ouest Nanterre La Défense, Nanterre, France*

Keywords: Web Mining, Information Extraction, Wrapper Induction.

Abstract: This paper presents a novel approach to extract product features from large e-commerce web sites. Starting from a small set of rendered product web pages (typically 5 to 10) and a sample of their corresponding features, the proposed method automatically produces labeled examples. Those examples are then used to induce extraction rules which are finally applied to extract new product features from unseen web pages. We have carried out an evaluation on 10 major French e-commerce web sites (roughly 1 000 web pages) and have reported promising results. Moreover, experiments have shown that our method can handle web site template changes without human intervention.

1 INTRODUCTION

Product feature extraction is a popular research area given the vast amount of data available on the Web and the potential economic implications. In this paper we focus on mining commercial product features from large e-commerce web sites, such as bestbuy.com or target.com. Given a product, we want to extract a set of related pairs (feature name, value). For example, for the "Apple MacBook Pro MD311LL/A" product, we would like to extract the information that the product color is silver, that its maximal display resolution is 1920x1200 pixels, its RAM size 4GB and so forth.

The massive extraction of product features can be useful to a variety of applications including: product or price comparison services, product recommendation, faceted search, or missing product features detection.

Our goal is to develop a method that allows mining product features in a self-supervised way (i.e. a semi-supervised method that makes use of a labeling heuristic), with a minimal amount of input. Moreover, the method should be as domain-independent as possible. We present in this paper a method that relies on a small set of web pages, few examples of product features, and visual clues. The input examples can be the output of a previous data processing, they may be given by a human, or they can be chosen from an existing Knowledge Database such as Icecat [1].

Using visual clues such as spatial position, instead of relying on HTML tags, brings robustness to the method and independence from specific HTML structure. Let's take tables as an example: various HTML tags can be used to present information in a tabular way. On the other hand, the <table> tag is sometimes used to visually organize web pages. Therefore, relying on the HTML <table> tag to identify tabular information is unsure. In addition to robustness, a good degree of domain-independence is achieved, as our method does not depend on text content, but only relies on visual clues. This is a major difference with similar work (see Section 2).

We have evaluated our system on 10 e-commerce web sites (1 000 web pages). Results show that the proposed approach offers very high performances. Further evaluations should be done to validate the method over e-commerce web sites which are less homogeneous from a structural point of view. However, as Gibson et al. pointed out (Gibson et al., 2005), about 40-50 % of the content of the web is built using templates. Thus, it seems to us that the results obtained are promising.

[1]http://icecat.us is an IT-centered multilingual commercial database created in collaboration with product manufacturers. Part of this database, Open Icecat is freely available but very incomplete.

Ferrez R., de Groc C. and Couto J..
SELF-SUPERVISED PRODUCT FEATURE EXTRACTION USING A KNOWLEDGE BASE AND VISUAL CLUES.
DOI: 10.5220/0003936706430652
In *Proceedings of the 8th International Conference on Web Information Systems and Technologies* (WEBIST-2012), pages 643-652
ISBN: 978-989-8565-08-2

The article is structured as follows: in Section 2, we survey existing methods regarding wrapper induction and product feature extraction. In Section 3, we describe the proposed approach. In Section 4, we evaluate our approach on a panel of 10 web sites (1 000 web pages). We conclude in Section 5.

2 RELATED WORK

The proposed method is close to two research fields in web mining: *Wrapper Induction* and *Product Feature Extraction*.

Wrapper Induction refers to the generation of extraction rules for HTML web pages. Introduced by Kushmerick (Kushmerick, 1997), wrapper induction methods rely on the regularity of web pages from the same web site, mostly due to the use of Content Management Systems (CMS).

While early work relied on human-labeled examples (Kushmerick, 1997), recent approaches, known as unsupervised wrapper induction, have been proposed in order to avoid this step. Those new approaches rely on two types of web pages: list-structured web pages displaying information about multiple products (Chang and Lui, 2001; Liu and Grossman, 2003; Wang and Lochovsky, 2002; Zhao et al., 2005) and product web pages (Arasu et al., 2003; Chang and Kuo, 2007; Crescenzi et al., 2001). However, unsupervised methods require a post-processing step, as attribute names are usually unknown (Crescenzi et al., 2001).

The use of prior knowledge to improve wrapper induction has been little studied compared to other approaches. Knowledge is provided to the system using different formalisms such as concepts (Rosenfeld and Feldman, 2007; Senellart et al., 2008) or facts/values (Wong and Lam, 2007; Zhao and Betz, 2007). Moreover, such methods usually aim at extracting a small number of specific features about a particular type of product (i.e. camera, computer, books).

On the other hand, *Product Feature Extraction* methods directly extract product features, without generating wrappers.

Wong et al.'s work (Wong et al., 2009) focuses on three structural contexts: two-column tables, relational tables and colon-delimited pairs. Once the structural context of their data has been heuristically identified, they apply a set of rules in order to handle the variable length of the data structures. Part of our method was inspired by this article, however the use of visual hypotheses instead of heuristics, allows us to handle more HTML structures displayed with the same appearance.

Wong et al. (Wong et al., 2008) propose a method that considers each page individually and can retrieve an unlimited number of features. The probabilistic graphical model used in their paper considers content and layout information. Therefore, relying on textual content implies that their model is domain-dependant.

Our work is closely related to that of Wu et al. (Wu et al., 2009). The main idea of their work is to first discover the part of the web page which contains all features, and then to extract them. The first step is performed using a classifier, and each NVP (Name Value Pair) discovered by this classifier receives a confidence score. The complete data structure is then located by taking the subtree with the best confidence score according to heuristic rules. A tree alignment is used to discover the remaining NVPs. This method can discover an unlimited number of features, but the initial classifier still needs to be trained on human-labeled examples. Moreover, as the previously discussed method (Wong et al., 2008), the classifier is trained for only one kind of product.

Our method inherits some ideas from these previous works, while investigating a different path based on visual information and an external knowledge base:

- A minimal knowledge base is provided to the system instead of human-labeled examples
- Visual clues avoid making assumptions about the HTML structure. As a result, features formatted with any kind of HTML structure but displayed as a table can be extracted
- The number of features extracted for one product is unlimited
- The extraction rules induced by our method can be applied to any type of product provided that the web site is built using templates

3 OUR METHOD

3.1 Overview

The different aspects of template-generated web pages used in the whole process include content redundancy (site invariant features), visual/rendering features and structural regularities. All these aspects lead to different steps applied to a set of web pages in two different approaches: page-level (local) and site-level (global) analyses (the site is represented by a sample of web pages, "site-level" is used instead of "page-set-level" for clarity). Page-level analyses refer

to algorithms that consider each page taken individually, whereas site-level analyses benefit from having multiple pages from the same site.

The whole process (summarized in figure 1) is iterative, and alternates steps at page- and site-level.

Taking as input a small set of web pages:

1. Product specifications are located using a combination of page- and site-level information (section 3.2)
   (a) content redundancy is evaluated using site-level information
   (b) an estimation of known feature coverage is computed per page
   (c) according to a) and b), every part of the pages is scored and ranked
   (d) features are located using a site-level vote
2. On each page, product feature names and values are automatically annotated (section 3.3)
   (a) a partial feature matching is performed to identify examples of feature names and values
   (b) more examples are inferred by relying on their layout
3. As a last step, extraction rules are induced using all annotated features (section 3.4)

3.2 Specification Block Detection

The first step of our method is to detect the block containing all the product features (which we call ”product specification” block) that we would like to extract.

Web pages generated from a particular template share common blocks of HTML. These parts are considered site-invariant. On the contrary, some elements depend on the product presented in the page, like feature table, description, prices, related products, ads, etc... Those site-variant features will give us a clue to identify the specification block. A distinction between the specification block and other variable parts of the pages is later achieved by crossing information with the external knowledge base.

After explaining how we generate candidate specification blocks (section 3.2.1, section 3.2.2), we describe a method for scoring and ranking each block (section 3.2.3) and a voting algorithm to select a final candidate (section 3.2.4).

3.2.1 Web Page Segmentation

Web pages can be cut into multiple parts of different sizes. These parts are called segments or blocks, and all correspond to a subtree in the DOM (Document Object Model) tree of the whole page. We studied segments instead of all displayed elements in the page (which is the trivial case of segmentation when every leaf in the DOM tree is a segment) in order to identify whole data structure blocks.

Web page segmentation is another field of research and advanced methods are not necessary in our case. We simply want to preserve a relative coherence for each block, which can be achieved by using node CSS (Cascading Style Sheets) properties. One of them, called ”display”, gives a good hint of how content placed under the node is rendered by a web browser. This way, we can exclude every element rendered as ”inline” or as a table part (”table-row”, ”table-cell”, ”table-column-group”, etc...). More precisely, we only keep nodes with ”display” property as ”block” or ”table”. Taking these two values guarantees that we don’t restrain the method and we can potentially extract well structured data formatted with other HTML tags. Strictly speaking, this method is not a web page segmentation method, mostly because the segments obtained are nested. In fact, this is not a problem because our scoring algorithm will cope with this aspect.

3.2.2 Block Identification

A major issue when trying to evaluate any variable aspect of segments from different web pages, is how to identify these segments and how to locate them within each page. Two considerations should be taken:

1. Each identifier should locate a unique segment (a sub-tree of the whole DOM tree) of the web page, for every page in the set
2. The same segment in each web page of the set should share the same identifier regardless of HTML optional elements

An example of the second item is when we can clearly see that a table displayed in every page of the set is the same, but the strict path (from the root of the DOM tree to the table) is not the same in all web pages. We refer to ”strict path” as the concatenation of HTML tags from root to any node, with the position of each tag specified at every level. The position is computed as follows: the first occurrence of a tag under a node has the first position and for every sibling node with the same tag, we increment the position by 1. On the other hand, we call a ”lazy path” the concatenation of HTML tags from root to an element without positional information.

A softer path can be computed without any position information. However such path cannot cope with condition 1 and thus may identify multiple segments on the web page.

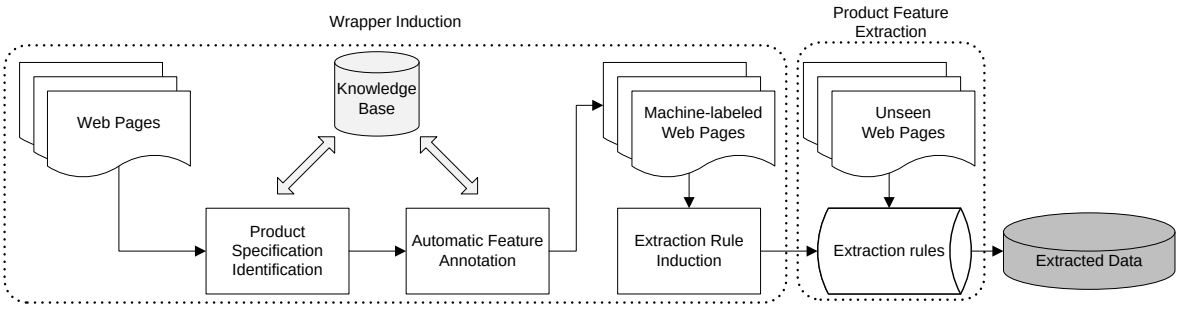

Figure 1: Complete product feature extraction framework.

These considerations led us to use a more flexible path, based on the XPath formalism.

At this point, we want each path to be robust against optional DOM nodes but strict enough to locate candidate blocks in all pages of the set. Hence, we start with a lazy path, and progressively add HTML attributes (”class”, ”id”) or position information so that each path locates a unique node in the page set. Usually pointing to CSS classes, HTML attributes such as class and id often refer to the visual or functional purpose of DOM nodes (”blue-link”, ”feature-name”, ”page-body”, ...). Using such information in our formalism generates more ”semantic” or interpretable paths.

3.2.3 Block Scoring and Ranking

In this section, we describe how the simultaneous use of content redundancy and of a knowledge base can help to distinguish which block contains the features regardless of how they are displayed.

We first analyze how text fragments are distributed within the set of pages, aiming at separating variable from invariable content. We later cross information between our knowledge base and pages in the set to isolate the variable part we want to extract.

Entropy-based Redundancy Analysis. There are different methods to evaluate content variability for the segments we have created. We used an entropy-based approach as proposed by Wong and Lam (Wong and Lam, 2007).

In the following, we refer to each segment by the node in the DOM tree corresponding to the root of the sub-tree. $\mathbb{W}$ is the set of words of this block. We first define the probability to find word $w \in \mathbb{W}$ in the text content located under node N as:

$$P(w,N) = \frac{occ(w,N)}{\sum_{w_i \in \mathbb{W}} occ(w_i,N)} \quad (1)$$

where $occ(w,N)$ is the number of occurrences of word w in the text content located under node N.

We directly define an entropy measure for node N on page p as:

$$E_p(N) = -\sum_{w_i \in \mathbb{W}} P(w_i,N) log P(w_i,N) \quad (2)$$

Taking one of the pages as a reference, we compute the difference of entropies between this page and other pages from the set in order to evaluate the content variability for all segments.

Actually, the measure defined in equation 2 can be computed for a unique page, or for multiple pages. In this case, the text content under the node N is not taken on one page but on all pages. The set of words is directly computed as the union of all sets.

Wong and Lam took as reference one of the pages from the set. We believe that it is hard to find the most representative page of the set. Moreover, we won't be able to evaluate all paths since the reference page only contains a limited number of paths. However, a complete scoring can be achieved by taking each page as the reference page once.

Formally, we define a measure of word dispersion, the information $\mathfrak{I}$ for node N, computed by:

$$\mathfrak{I}(N) = \frac{1}{|\mathbb{P}|} \sum_{p \in \mathbb{P}} |E_p(N) - E_{\forall p' \in \mathbb{P}, p' \neq p}(N)| \quad (3)$$

where $\mathbb{P}$ is the set of pages.

For every node which contains invariant text content, $\mathfrak{I}$ will be null. On the contrary, when the text content varies a lot (the set of words located under node N is very large), $\mathfrak{I}$ will be high.

Because we are interested in segments which contain a lot of informative nodes (feature values are expected to be very different from one product to another), this measure gives a good hint for identifying potential specification block.

At this point, we could use a threshold to differentiate variable blocks from invariable ones. However, identifying the specification block by solely relying on the variability criterion $\mathfrak{I}$ proved difficult. For instance the specification block was often blended with other variable segments, like customer reviews.

Feature Matching and Final Score. The easiest way to differentiate feature-rich sections from other variant sections is to look at the coverage of a reference feature set.

The advantage of the knowledge base we used is that it does not require a human effort. Moreover, this knowledge base can be completed when new data are extracted. During our test we used a free product feature database, Icecat which provides a large multilingual product feature source.

The feature coverage FC can be computed using a standard bag-of-words model, defined as:

$$FC(N) = \frac{|\omega_N \cap \omega_f|}{|\omega_f|} \quad (4)$$

where ω_f is the set of words computed on feature values in the reference set, and ω_N is the set of words in the text content of node N.

Finally, we can combine equations (3) and (4) to compute a final Specification Block Score SBS:

$$SBS(N) = (1-\lambda)\Im(N) + \lambda FC(N) \quad (5)$$

where $\lambda \in [0;1]$ is automatically computed according to the feature coverage in the text of the whole page FC_p. The fewer FC_p is, the bigger λ is for every node. In fact, feature coverage should be a strong indication of where the specification block is located. A small value of FC_p indicates differences on presentation text for a lot of features, and FC should be more weighted than $\Im$. On the contrary, if this value is too high, this may indicate either lots of matching in other parts of the page or less differences in how presentation text is written. In this case, weights of FC and $\Im$ are balanced because FC value is less reliable.

3.2.4 Candidate Block Selection

At this stage, we have a ranked list of blocks for each page in the set. We now want to decide which block designates the specification one.

Instead of averaging values of the SBS score for each block over all web pages, we use a voting method, more robust to the fact that SBS scores are simultaneously very small and close.

For example, a typical case we try to overcome is when one page contains a product description written in plain text and composed of many product features. Using an average SBS value usually leads to a wrong final ranking.

Therefore, we have evaluated two preferential voting methods: Borda count and Nanson's method. The difference between those two is that Nanson eliminates choices that are below the average Borda count score at every ballot. Initial tests show Nanson's method yields better results and is robust enough to deal with our most ambiguous cases.

The final result of the product specification block detection is illustrated in figure 2. The specification block is colored in light grey.

Figure 2: First step - specification block identification.

3.3 Data Structure Inference

After locating the product specification block, we need to find how features are presented in order to annotate them. Recall that each feature we want to extract is composed of two elements. The first part of each feature is its name and the second part is its corresponding value.

For each page of our set, we first use the knowledge base to identify both elements for each feature in the data structure. We obtain a partial matching due to the fact that our knowledge base is incomplete. Moreover, due to language variability, several feature names and values will mismatch or not match at all. Consider for instance matching a camera's sensor resolution ("Canon EOS 600D"). Our database contains a "Megapixel" feature name and a corresponding value of "18 MP". However, depending on the web site, this same value may be written as "18 MP", "18 Mpx", "18 million px", "18 million pixels", "18 mega pixels" or even "18 000 000 pixels". Matching such values from our knowledge base without using normalization rules is a difficult task. In this work, we rely on a simple edit distance to match our knowledge base entries to web page elements which means we will have to handle a lot of mismatches.

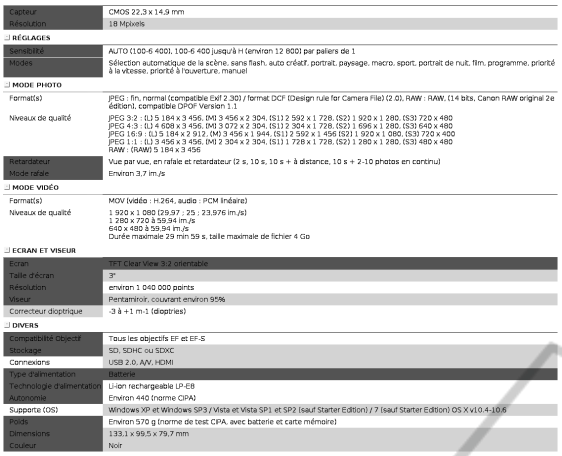

Figure 3: Second step - partial feature matching.

To cope with silence and errors, we use visual clues and hypotheses about how these features are displayed. We finally obtain a valid and large set of machine-labeled examples.

3.3.1 Partial Feature Matching

If we consider a product web page and a reference set of features for this product, we can assume to find known features in the web page, even if there is a lot of variation about how feature names and values are written.

We use a simple string edit distance to match each text fragment (corresponding to a leaf in the DOM tree) with reference feature names and values. Each text fragment is assigned to the feature name or value that minimizes the distance. An empirically fixed threshold is used to avoid matching unrelated text fragments and reference features.

This partial feature matching is illustrated in figure 3. feature names and values are respectively colored in dark grey and light grey.

3.3.2 Data Structure Generalization

The current machine-labeled examples (both feature names and values) are incomplete and noisy, for multiple reasons:

- Some values in the web page are considered as feature names in the reference feature set
- Some text fragments are neither a feature name nor a value
- Some text fragments have been mismatched

Thus, we need to clean these examples in order to:

1. Remove as much noise as possible
2. Maximize the number of examples without adding extra web pages

We can achieve these goals by making hypotheses about how features are displayed. We use visual-based hypotheses (after rendering the page) instead of tree-based ones because it gives us a complete independence towards the underlying HTML structure.

We distinguish between two kinds of practices used when presenting data in a table-like structure that justify the use of visual-based hypothesis.

First, we find every method used for displaying each feature:

- Different formatting tags (<b>, <i>, <big>, ...) for cells of the same table
- Some of the values or feature names are links
- Images are used to clarify some features (typically for features that take few values and are key features for selling the product, for example sensor resolution of a camera)

Web developers can employ other formatting methods (non-table tags combined with CSS properties) to display features as a table. Moreover, W3C recommendations are not always followed when using proper table tags. All those facts lead us to various situations:

- Each table row contains another table structure, giving a nested table tree
- Labeling cells (namely our feature names) should be encoded using the ”TH” tag, but are more often seen with the ”TD” tag
- The entire table is formatted using nested ”DIV” tags or HTML definition lists (”DL”/”DT” tags)

Having the DOM tree after rendering instead of the usual DOM tree based only on the HTML file, gives direct access to geometric and HTML specific attributes. Every node of the HTML tree is rendered as a box and geometric attributes can be retrieved as absolute positions and sizes. This process is quite expensive in time, because we need to fetch images and run scripts on every page. However, we conceived our framework to limit the rendering process to input pages only, thus making the resulting cost acceptable.

The two hypotheses that we made are:

1. Features should be displayed in a table-like structure
2. Based on the first hypothesis, feature names and values should be aligned vertically or horizontally

We applied theses hypotheses on name and value rendered box center coordinates. Experiments show that they are robust enough to tackle real life issues.

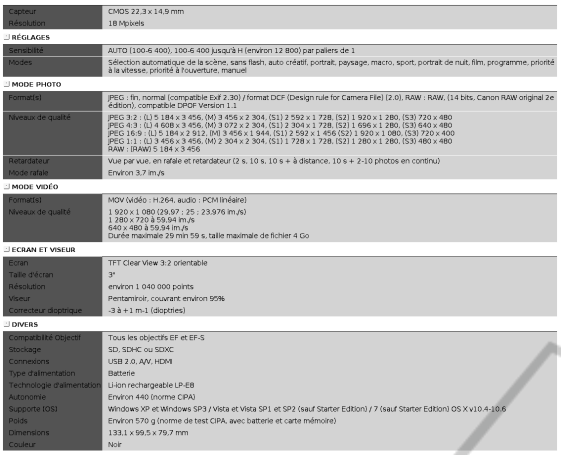

Figure 4: Third step - visual alignment.

Based on the same previously shown example in figure 2 and 3, we applied this method and present the result in figure 4. As before, feature names and values are respectively colored in dark and light grey.

3.3.3 Name-value Association

The last step of our method is to associate each marked feature name with its corresponding value.

We use again visual clues instead of tree-based clues to avoid issues described in the previous section (formatting tags, improper HTML table tags, . . .). We believe that visually closest names/value pairs should be associated together. We resort to a euclidean distance between the coordinates of the box centers.

3.4 Extraction Rule Induction

All previous steps consist only in the generation of machine-labeled examples, and replace the laborious manual work of human labeling.

The data structure recognition process yields a large set of samples (pairs of feature names and values) from a small set of pages. The major drawback is that all visual clues are given by a rendering engine. In practice, we don't want to use this kind of method for the extraction of a complete web site, so we used another format for the extraction rules, more flexible and which can be used in other systems.

Different methods have been employed for the induction of wrappers based on labeled examples, including string-based extraction rules (Muslea et al., 2001), regular-expressions (Crescenzi et al., 2001), or tree automata (Kosala et al., 2006). We prefered to use XPath rules instead because of its wide use in web information systems as well as its flexibility.

Based on previously machine-labeled examples we can automatically induce extraction rules in three parts:

1. *XPath 1*: Path to the product specification block
2. *XPath 2*: Path to all feature pairs, relative to *XPath 1*
3. *XPath 3*: Path to feature name and value for each feature pair, relative to *XPath 2*

Instead of building a strict XPath as described in section 3.2.2, we can take advantage of the flexibility of the XPath language which can handle HTML attributes for node localization. The failure of a strict XPath rule caused by the existence of optional elements can be avoided in most cases with this method. Thus, if a node has a unique "id" or "class" attribute, we use this information and use strict position numbers as a last resort.

Although the formalism is the same, the method used for the automatic induction of these rules is different from the one we used in the section 3.2.2, where each path should locate only one segment per page, which was mandatory for the correct evaluation of content variability. In fact, at this stage, the XPath is not restricted for the same purpose (identifying a unique node) because we want more genericity for two reasons:

1. Extracting an unlimited number of features. In particular the *XPath 2* can match multiple nodes in each page
2. Being able to handle unseen pages

For these reasons, for each rule, we try to induce an XPath which can validate as much machine-labeled examples as possible. This can be achieved by using disjunction over HTML attributes usually used for CSS classes.
Example: `.../TABLE/TR[@class='allparams even' OR @class='allparams odd']`

Using one of these attributes is not always possible. The worst case in when we have a different strict XPath for each node. In this case, the system builds multiple rules. However, this case never happened on the web sites from our evaluation set.

4 EVALUATION

4.1 Corpus

To the best of our knowledge, there is no annotated data to evaluate product feature extraction.

We have considered evaluating the proposed approach using a cross-validation method and the Icecat

knowledge base. However due to text variability (as discussed in section 3.3), this proved difficult, leading us to produce our own manual annotations.

We have created a novel collection of product features by downloading a sample of 9 different French major e-commerce web sites: boulanger.fr, materiel.net, ldlc.fr, fnac.com, rueducommerce.fr, surcouf.com, darty.fr, cdiscount.com and digitphoto.com. The ldlc.fr web site changed its page template during our experiments so we evaluated our method on the first and second version of this web site (resp. ldlc.fr (v1) and ldlc.fr (v2)). This emphasizes an interesting aspect of our method which is its robustness to structure changes: even if extraction rules change, product features are usually kept as is. Thus, our method can readily induce new rules without human intervention.

For each web site, a gold standard was produced by randomly selecting 100 web pages which did not belong to any category in particular (Movies & TV, Camera & Photo, ...) and annotating product features (name and corresponding value). Finally the corpus is composed of 1 022 web pages containing 19 402 feature pairs.

4.2 Experimental Settings

For each web site, we ran our method as follows:

- We randomly chose 5-10 unseen web pages from randomly chosen categories
- We retrieved the corresponding feature sets from the Icecat knowledge base. Association between a web page and a feature set was achieved automatically by looking at the product name and the page title
- We applied the proposed method and induced XPath extraction rules
- Finally, we applied those rules to our gold standard web pages in order to extract product features

We have used standard metrics to assess the quality of our extractions:

- Precision, defined as the ratio of correct features extracted to the total number of features extracted
- Recall, defined as the ratio of correct features extracted to the total number of all available features.

4.3 Results

As shown in table 1, our method offers very high performance. Most of the time, the system gives a perfect extraction, due to a good *templateness* and little variability in the whole web site. This proves that our initial hypotheses and the choice of the XPath formalism were relevant. Actually, our custom formalism derived from XPath correctly captures what is regular in templated web pages: HTML structure (tags) and attributes (such as the "class" attribute which provides rendering clues sometimes). Moreover, dividing our extraction rules in three parts (see section 3.4) allows us to extract features precisely and robustely which leads to high precision. Our sequential approach is a major difference with previous methods that considered all text fragments in web pages. However, as it is clearly iterative, failure of one step of the method is irrecoverable which is exactly why extractions on web sites 8 and 10 failed.

More interestingly, we observe mixed results on web sites 6 and 7. The lower recall for web site 6 can be explained by a misrepresentative sample. The extraction rules do not cover all existing HTML attributes that locate the specification block due to the absence of examples while inducing the rules. The noise extracted for web site 7 is due to the alignment hypothesis. In-depth analysis reveal that several table cells, aligned with product feature names or values, are mislabeled. For instance, features relative to a computer screen are preceded by a "Screen" cell erroneously labeled as a feature name.

We tried to overcome some of those problems by providing more input pages for these sites. We decided to limit the number of input pages to 10 in order to respect our initial goal to use few input pages. In fact, we made the hypothesis that *SBS* scores on these sites were wrong due to a lot of differences in the DOM trees. The use of more pages gives a more precise evaluation of text variability and increases the probability of crossing known features on web pages. Results shown in table 2 and in-depth analyses confirm this hypothesis.

On site 8, when providing only 5 pages as input, a block containing a lot of features written in plain text was selected instead of the specification block. This problem was avoided when more pages were provided. On site 6, recall did not increase which means that there are still unseen cases in the test set.

Results for site 10 show another issue, which cannot be handled by our method regardless of how many pages we use as input. The main context which leads to the failure of the specification block detection is when we cannot compare the same segments on all pages. Manual analysis of each step for this web site show that this case happened here. In fact, we don't have any specific HTML attributes (the usual "id" and "class") for locating web page segments, and there are different optional elements on each page too. The

Table 1: Evaluation of product feature extraction.

		Input pages	Pages	Features	Precision	Recall
1	boulanger.fr	5	100	1 390	1.00	1.00
2	materiel.net	5	100	2 960	1.00	1.00
3	ldlc.fr (v1)	5	102	1 324	1.00	1.00
4	ldlc.fr (v2)	5	102	1 498	1.00	1.00
5	fnac.com	5	101	1 856	1.00	1.00
6	rueducommerce.fr	5	140	2 190	1.00	0.723
7	surcouf.com	5	102	2 125	0.76	1.00
8	darty.fr	5	127	2 917	#	0.00
9	cdiscount.com	5	48	1 271	1.00	1.00
10	digit-photo.com	5	100	1 871	#	0.00
	Total	#	1 022	19 402	0.97	0.77

Table 2: Impact on recall of increasing the number of input pages.

		Input pages	Pages	Features	Precision	Recall
6	rueducommerce.fr	10	140	2 190	1.00	0.723
8	darty.fr	9	127	2 917	1.00	0.94
10	digit-photo.com	10	100	1 871	#	0.00
	Total (for all sites)	#	1 022	19 402	0.97	0.87

combination of both leads to the comparison of different parts of the web pages, thus giving a wrong measure of content variability. Moreover, the vote cannot be done because equivalent blocks don't have the same identifier on all web pages. Even if this case shows a clear disadvantage of our pipeline approach, average results indicate that the idea behind the construction of a path based on the XPath formalism is still relevant.

5 CONCLUSIONS

In this paper, we have tackled the problem of product feature extraction from e-commerce web sites. Starting from a small set of rendered product web pages (typically 5 to 10), our novel method makes use of a small external knowledge base and visual hypotheses to automatically produce feature annotations. The proposed method, designed as a pipeline, is composed of three sub-tasks: product specification identification, feature matching and data structure recognition, and, finally, extraction rule induction. Those extraction rules are then applied to extract new product features on unseen web pages.

We have carried out an evaluation on 10 major French e-commerce web sites (roughly 1 000 web pages) and have reported interesting results.

We are considering several leads for future work. First, as the proposed approach is built as a pipeline, it offers high precision and no noise but a single failure leads to a complete failure of the method. Thus, we will explore more global approaches which could avoid such effect. In particular, as results show the importance of having a representative set of web pages for inducing the extraction rules, we will develop a method for building such sets. Secondly, we will extend the proposed approach to handle more data structures such as colon-separated product features. Finally, while the method is domain independent, which is an interesting property for large and cross-domain web sites, we will focus our work on small web sites such as small specialized portals.

ACKNOWLEDGEMENTS

We would like to thank Mickaël Mounier for his contribution on the rendering engine and the annotation tool. We also gratefully acknowledge Marie Guégan for her helpful comments on this paper. This work was partially funded by the DGCIS (French institution) as part of the Feed-ID project (no. 09.2.93.0593).

REFERENCES

Arasu, A., Garcia-Molina, H., and University, S. (2003). Extracting structured data from Web pages. *Proceedings of SIGMOD '03*, page 337.

Chang, C.-h. and Kuo, S.-c. (2007). Annotation Free Information Extraction from Semi-structured Documents. *Engineering*, pages 1–26.

Chang, C.-H. and Lui, S.-C. (2001). IEPAD: information extraction based on pattern discovery. *Proceedings of WWW' 01*.

Crescenzi, V., Mecca, G., and Merialdo, P. (2001). RoadRunner: Towards Automatic Data Extraction from Large Web Sites. *Very Large Data Bases*.

Gibson, D., Punera, K., and Tomkins, A. (2005). The volume and evolution of web page templates. In *Special interest tracks and posters of the WWW' 05*.

Kosala, R., Blockeel, H., Bruynooghe, M., and Vandenbussche, J. (2006). Information extraction from structured documents using k-testable tree automaton inference. *Data & Knowledge Engineering*, 58(2):129–158.

Kushmerick, N. (1997). *Wrapper induction for information extraction*. PhD thesis, University of Washington.

Liu, B. and Grossman, R. (2003). Mining data records in Web pages. *Proceedings of SIGKDD' 03*, page 601.

Muslea, I., Minton, S., and Knoblock, C. A. (2001). Hierarchical Wrapper Induction for Semistructured Information Sources. *Autonomous Agents and MultiAgent Systems*, 4(1):93–114.

Rosenfeld, B. and Feldman, R. (2007). Using Corpus Statistics on Entities to Improve Semi-supervised Relation Extraction from the Web. In *Proceedings of ACL' 07*, pages 600–607.

Senellart, P., Mittal, A., Muschick, D., Gilleron, R., and Tommasi, M. (2008). Automatic wrapper induction from hidden-web sources with domain knowledge. *Proceeding of WIDM '08*, page 9.

Wang, J. and Lochovsky, F. (2002). Wrapper induction based on nested pattern discovery. *World Wide Web Internet And Web Information Systems*, pages 1–29.

Wong, T.-L. and Lam, W. (2007). Adapting Web information extraction knowledge via mining site-invariant and site-dependent features. *ACM Transactions on Internet Technology*, 7(1):6–es.

Wong, T.-L., Lam, W., and Wong, T.-S. (2008). An unsupervised framework for extracting and normalizing product attributes from multiple web sites. *Proceedings of SIGIR' 08*, page 35.

Wong, Y. W., Widdows, D., Lokovic, T., and Nigam, K. (2009). Scalable Attribute-Value Extraction from Semi-structured Text. *2009 IEEE International Conference on Data Mining Workshops*, pages 302–307.

Wu, B., Cheng, X., Wang, Y., Guo, Y., and Song, L. (2009). Simultaneous Product Attribute Name and Value Extraction from Web Pages. *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 295–298.

Zhao, H., Meng, W., Wu, Z., Raghavan, V., and Yu, C. (2005). Fully automatic wrapper generation for search engines. In *Proceedings of WWW' 05*.

Zhao, S. and Betz, J. (2007). Corroborate and learn facts from the web. *Proceedings of SIGKDD' 07*, page 995.