Filtering Web Documents for a Thematic Warehouse
Case Study : eDot a Food Risk Data Warehouse (extended)

Amar-Djalil Mezaour

Laboratoire de Recherche en Informatique **(LRI)**, Université Paris Sud
91405 Orsay cedex, France. **email : mezaour@lri.fr**

Abstract. Ordinary sources, like databases and general-pupose document collections, seems to be insufficient and inadequate to scale the needs and the requirements of the new generation of warehouses: thematic data warehouses. Knowing that more and more online thematic data is available, the web can be considered as a useful data source for populating thematic data warehouses. To do so, the warehouse data supplier must be able to filter the heterogeneous web content to keep only the documents corresponding to the warehouse topic. Therefore, building efficient automatic tools to characterize web documents dealing with a given thematic is essential to challenge the warehouse data acquisition issue. In this paper, we present our filtering approach implemented in an automatic tool called *"eDot-Filter"*. This tool is used to filter crawled documents to keep only the documents dealing with food risk. These documents are then stored in a thematic warehouse called ***"eDot"***. Our filtering approach is based on *"WeQueL"*, a declarative web query langage that improves the expressive power of keyword-based queries.

1 Introduction

A thematic warehouse can be a good solution for collecting, organising and storing large amounts of information related to a given thematic. This information can be provided by different and heterogeneous sources like : databases, databanks, document collections etc. Nowadays, The web is the most used source for supplying warehouses with data from web documents. The supplying process consists of a crawler that fetches the web to gather a lot of relevant documents to the warehouse. Due to the huge amount of available information in the web and its poor organisation, a crawler gathers also a lot of irrelevant and useless documents for the warehouse. Finding relevant material from the gathered documents can be a very hard task. To avoid the storage of irrelevant data in the warehouse, efficient and precise filtering tools must be used downstream of the data acquisition process. Two different filtering approaches can be implemented to filter the warehouse candidates documents (i.e., the gathered documents for a warehouse).

The simplest one consists in evaluating thematic keyword-based queries over search engines repositories to obtain a list of urls. These urls correspond to the documents that are more likely to be related to the warehouse topic (according to the search engine query evalutor). We call these documents the relevant documents. Note that, with this filtering approach, all the indexed documents (the relevant documents and the non relevant ones) in the chosen repositories constitute the candidates of the thematic warehouse to implement. These candidates are already known and crawled by the search engine crawler. So, intensive web crawling can be avoided since these documents (the relevant ones) are known and indexed by the existing search engines. The only difficulty consists on expressing the most precise keyword query that best describes the targeted thematic. This filtering approach is used in the Thesus project [1] to populate a thematic data warehouse. The thematic of Thesus deals with technology documents. The authors have manually constructed a set of keyword queries to describe the technology topic. This set of queries has been submitted to Google [2] in order to collect the urls of the documents dealing with technology (relevant documents). This approach is very dependent on the precision of the used search engines and of the quality of the submitted keyword queries. In addition to this, keyword-based queries are not enough expressive to express complex requirements or to query documents with poor textual content. For example, it is not possible to formulate a keyword query that can filter web documents according to their mime type, the fact that they contain tables having a given property...

The second filtering approach consists of achieving the filtering process during the web crawling. To do so, one has to build a specific web crawler suited to the thematic warehouse requirements. Such a crawler traverses the web in a restrictive mode to find the most possible relevant documents to the warehouse without exploring the entire web. This technique is known as Focused crawling [3–5] and is used to populate different topic-specific warehouses such as CORA [6] and CiteSeer [7], on-line scientific papers warehouse. Focused crawling is built upon an intensive learning process that requires an important training set as input. In fact, a classifier is trained to learn a filtering function from a preliminary set of web documents labelled as "relevant" and "irrelevant" documents. This preliminary set is called the training set. To produce a precise and efficient filtering function, the training set must contain enough relevant and representative documents. Constructing such a set leads to the same issue as populating the warehouse itself : how to locate representative documents in the web starting from nothing ! In addition to this, the labelling is often done manually which can be tedious for an important training set.

To summarize, two different approaches can be used to implement a filtering tool for a thematic warehouse: using a list of thematic keywords as a filtering query over a collection of documents or using machine learning techniques, during or after the web crawl, to recognize the desired relevant documents. Each approach can be viewed as a process that describes and characterizes the warehouse thematic. We strongly believe that an elaborate description and characterisation of the warehouse thematic can be very useful in the filtering process. This is what motivates the use of ***WeQueL*** [8,9] in the implementation of *"eDotFilter"*, the filtering tool of the ***"eDot"*** warehouse. WeQueL is a declarative and multi-criteria web language that improves the expressive power of keywords queries. ***"eDot"*** is a project aiming at building automatic and semi-automatic tools for the implementation of thematic warehouses. The chosen thematic is food risk. The targeted web documents (relevant documents) are defined as documents containing microbiological data for food risk prevention and assessment.

We present in this paper an extension of the preliminary work on filtering web documents presented in [10].

We organised our article into 5 sections. In section 2, we present ***"WeQueL"*** the web query langage that is in use in our filtering process. We introduce in section 3 our filtering approach. The experiments to validate our approach are presented in section 4. Finally, we conclude with the improvements and the on-going work.

2 WeQueL : *Web Query Language*

WeQueL [8,9] is an attribute-value language that allows a user to define its needs by combining several and different keyword-based criteria in one query. Each combined criterion targets its keywords to a specific part of a web document or describes a particular property of a document. For example, the title criterion allows the user to specify a list of keywords to search them in the title of the documents to evaluate whereas the mime criterion restricts the evaluation to the documents having the specified mime type. WeQueL proposes 9 different criteria that are the basic building blocks for complex queries. We call those criteria *"atomic queries"*. An atomic query is an attribute-value like query having the following form:

$$q_a \ : \ \texttt{attribute} \quad \xrightarrow{\text{contains}} \quad [\texttt{value}_0, \ldots, \texttt{value}_N]$$
$$\qquad\qquad\qquad \xrightarrow{\text{evaluated by}} \quad \texttt{semantic\_constructor}$$

The attribute of a WeQueL atomic query specifies the part of the document or one document property to target. As mentioned above, WeQueL offers 9 different attributes to focus the search on 9 predefined parts (sections) or properties of a web document:

1. `page_title`: When used in an atomic query, this section targets the search of `value` strings in the title of a document. For an HTML document, we defined the title of a document as the set of words contained between the tags `<title>` and `</title>` or between `<h1>` and `</h1>` or in the value of the attribute `content` of the `<meta name="title">` tag.
2. `mime`: This section restricts the evaluation of the documents to those having their mime type mentioned in the values part of q_a. The values of an atomic query using `mime` section must be valid mime types (like `text/html`, `image/jpeg`, ...)..
3. `title_incoming_page`: This section targets the search of the specified values of q_a in each title of the documents that point to the evaluated document. Suppose that q_a is evaluated over a document d. Suppose that d' is a document pointing to d. Then, the evaluation of q_a over d is equivalent to evaluate q'_a over d' such as q'_a is the atomic query that uses the same values and the same semantic constructor as q_a but targets the title instead of the incoming title.
4. `url`: This section targets the search of `value` strings in the url of each evaluated document.
5. `outgoing_links`: The outgoing links section allows the user to focus the search of the values strings in the words appearing in the neighborhood of the outgoing links of a document. We defined the neighborhood of a link as the words appearing in : its anchor, the tokens of the urls that this link refers to and the 10 words before and after the link (before the `<a href>` tag and after the closing tag `</a>`).
6. `incoming_links`: An atomic query using an incoming links section is evaluated in the same manner as an atomic query using the previous section (outgoing links). The only difference lies in the fact that this section (incomong links) targets the neighborhood of the links pointing to the document to evalute.
7. `keywords`: This section is specific to HTML documents. It is used to target the content of the `content` attribute in the `<meta name = "keywords">` tag (when this last one exists in the document to evaluate).
8. $\texttt{object}_{[type='T']}$: This section allows the user to focus the evaluation on the words appearing an HTML object. An HTML object can be a table, a graphic, an applet... For the eDot needs, we handled only tables and lists (i.e., $T \in \{$`'table_or_list'`$\}$). We define a table in an html document as the words appearing between `<table>` and `</table>` tag, between `<dl>` and `</dl>` tag, between `<ul>` and `</ul>` and between `<ol>` and `</ol>`.

The values of a WeQueL atomic query are separated by commas. They are sequentially searched in the content of the targeted section of the document to evaluate. Each value represents a string that contains a single keyword or a list of keywords separated by white space characters. Consider "q_a" an atomic query that targets a given section "s" of web documents. Let's consider also $\texttt{value="w}_1\ \texttt{w}_2\ \ldots\ \texttt{w}_n\texttt{"}$ one string value specified in q_a. This value defines an occurrence order that we note: $\mathtt{w}_1 \mathtt{<} \mathtt{w}_2 \mathtt{<} \ldots \mathtt{<} \mathtt{w}_n$. The search of this string in the section s of a document consists in mapping each $\mathtt{w}_i$ with a word $\mathtt{w}'_i \in s$ such that :

1. $\mathtt{w}_i$ is a substring of $\mathtt{w}'_i \in s$. We refer to this inclusion by: $\mathtt{w}_i \subseteq \mathtt{w}'_i$;
2. the mapped words in s occurs in the same order as in `value` i.e., $\forall i \in \{1, \ldots, n\} : w'_i < w'_{i+1}$.

We note this mapping by μ.
Example: consider `value="food data"` to search in the title of 3 documents:

Document title	value-title mapping (μ)	matching success
`"food database"`	μ(`food`)=`food` $<$ μ(`data`)=`database`	yes
`"data food table"`	μ(`food`)=`food` $\not<$ μ(`data`)=`data`	no
`"on-line book store"`	μ(`food`)=μ(`data`)=∞	no

When all the words contained in a given value are mapped, this value is said *matched over the section "s"*. A word $\mathtt{w}' \in s$ is said to be *relevant* when: $\exists$ `value` $\in q_a$, $\exists w \in$ `value`: (`value` is matched) $\wedge$ (μ(`w`) =`w'`).

Semantics of a WeQueL atomic query The WeQueL language offers the possibility to determine how to exploit the matched values of an atomic query over a document to estimate the relevance of this document. In fact, the third part of an atomic query is reserved for the specification of the calculation method of a score that represents the relevance of the evaluated document according to the evaluated atomic query. We call this calculation method a "semantic constructor". WeQueL offers two categories of semantic constructors: logical constructors and numerical constructors. A logical constructor is a calculation method that interprets the matched values in a propositional logic setting. The calculated score is then equals to 0 or 1. We implemented 5 logical constructors: `"disjunct"` which assigns 1 to the atomic query score when at least one value is matched, `"conjunct"` which assigns 1 when all the values of q_a are matched, `"atleast(k)"` which assigns 1 when `k` or more values are matched, `"atmost(k)"` and `"exactly(k)"`. Unlike a logical constructor, a numerical constructor assigns a score comprised between 0 and 1. We implemented three numerical constructors. `"precision"` calculates the percentage of the relevant keywords in the targeted section of a document. When this score is high, this means that the evaluated document section contains a lot of the keywords mentioned in the atomic query. `"recall"` calculates the percentage of the matched values of the atomic query. A high `"recall"` score means that the evaluated section of a document covers a lot of the values specified in the query. Finally, `"fscore"` is a combination of the previous scores: $\frac{2 \times \mathtt{precision} \times \mathtt{recall}}{\mathtt{precision} + \mathtt{recall}}$.

Having the definition of the atomic queries, WeQueL provides a combination formalism that permits the simultaneous (i.e., in the same query) targeting of different parts and properties of a document. We call this formalism a "WeQueL combination" and we note it $\mathcal{C}$.

$$\mathcal{C} : 1 \cdot \underbrace{q_{\texttt{mime}} \cdot q_a^1 \cdot q_a^2 \cdots q_a^{N-1} \cdot q_a^N}_{\text{1-weighted part}} \langle \underbrace{\alpha_1 \cdot \widehat{q_a^1} , \ldots\ldots , \alpha_M \cdot \widehat{q_a^M}}_{\alpha\text{-weighted part}} \rangle$$

A WeQueL combination is an enumeration of weighted atomic queries where: one and only one atomic query must refer to the mime type and there can not be atomic queries duplicates (i.e., a combination can't have two title atomic queries). According to the used semantic constructors and the assigned weights, a WeQueL combination contains two disconnected parts: the part of the 1-weigthed part and the part of the α-weighted part where $\alpha < 1$. The 1-weighted part contains the mime type atomic query and only logical queries. This part expresses the necessary conditions that a document must satisfy (i.e., evaluated to 1) to carry on the evaluation to the next part. The α-weighted part contains logical and/or numerical atomic queries that will determine the final score of each evaluated document. This final score is calculated over a document d by the following formula:

$$score(d, \mathcal{C}) \;=\; score(d, q_{\texttt{mime}}) \times \prod_{i=1}^{N} score(d, q_a^i) \sum_{j=1}^{M} \alpha_j \cdot score(d, \widehat{q_a^j})$$

To avoid the developement of a specific syntactic parser, we defined an equivalent syntax for WeQueL queries based on the XML language. Therefore, we defined an XML-schema that describes how to build a WeQueL query. This schema is available at: `http://www.lri.fr/~mezaour/WeQueLschema.xsd`. Thus, a WeQueL query is an XML document that must conform to this schema.

3 eDotFilter : *the eDot filtering module*

"eDot" is the acronym of *"**E**ntrepôt de **D**onnées **O**uverts sur la **T**oile"* which means *"data warehouses open to the web"*. Four partners are involved in this project : IASI-LRI team (`http://www.lri.fr/iasi/`), GEMO-INRIA project (`http://osage.inria.fr/verso/`), BIA-INAPG team (`http://www.inapg.inra.fr/recherche/`) and Xyleme start-up (`http://www.xyleme.com/`). For more details on the project see `http://www-rocq.inria.fr/~amann/edot/`.

"eDotFilter" module is the filtering tool that determines from a set of crawled candidates documents, the ones to include in the eDot warehouse. To do so, our module uses a WeQueL combination query to estimate the relevance of eDot candidates to the food risk thematic. We call this combination the "filtering query". The filtering query describes, in a declarative way, the content properties of the documents to include in the eDot warehouse (i.e., dealing with food risk). Microbiologist experts defined a food risk document as an HTML document having atleast one HTML table that contains the most possible keywords from given food risk ontologies. In eDot, we use two food risk ontologies : sym'previus [11] and ComBase [12]. Therefore, we used in the filtering query an atomic query that targets the html tables with all the keywords of the eDot food risk ontology. We also used the set of few relevant document examples given by our experts to enhance the previous atomic query with other different atomic queries. We wrote a java program that parses the content of these relevant examples and suggests in its output the values (strings of two consecutive words) to put in the different atomic queries. For example, our program extracted from the title tag of one of our relevant examples the string: `"fda/cfsan - approximate ph of foods and food products"`. This string is cleaned from its common words (like articles: of, and...) and splitted into a list of 6 suggestions to put in a title atomic query: `fda cfsan, cfsan approximate, approximate ph, ph foods, foods food` and `food products`. We manually discard irrelevant suggestions (`fda cfsan, cfsan approximate, foods food`). The choice of the semantic constructors and the weights to use with each atomic query were determined after discussion with experts according to the eDot relevant document specifications. We finally obtained an advanced WeQueL combination [13] having 8 atomic queries. The second step consists of calculating the relevance score of each eDot candidate document. To do so, we developed in java the WeQueL query engine that parses the HTML content of each candidate and the content of the filtering query xml file to output the evaluation score. The final output is ordered from the highest candidate score to the lowest one. The filtering is then done by setting a threshold relevance and retaining only the candidates that have a score above this threshold. As it is hard to predict this threshold, it is experimentally set.

4 Experimental results

The crawling module of the eDot warehouse gathered from the web 2058 candidate documents as an input to "eDotFilter". The html content of these 2058 candidates were fetched, parsed and evaluated over the filtering query. As a result, we obtained a ranked list of pairs `(candidate_url,score)`. In this experiment we intented to show the robustness of the score ranking achieved by our filtering tool. In other words, are the food risk documents ranked first or not? We intended also to show that the numerical evaluation score of each candidate "*d*" over the filtering query reflects the effective relevance of "*d*" to the considered thematic. To do so, we used a `ROC` curve [14] and a `lift` curve to achieve our experimental goals.

A `ROC` curve shows how good and how earlier the relevant documents are covered across the established ranking. The relevant documents are represented in the Y-axis and the non relevant in the X-axis. A `ROC` curve starts from the first position of the ranking and ends at the last one. At each position "i", the curve grows by one unit if the document ranked at position "i" is relevant. When this document is not relevant, the curve keeps its last value. A `ROC` curve represents a continuous and increasing function that is bounded by 1. The quality of the ranking system is then mesured by the AUC: Area Under Curve according to this classification table [15]:

ROC curve AUC	Quality
$0.9 < AUC \leq 1.0$	Excellent
$0.8 < AUC \leq 0.9$	Good
$0.7 < AUC \leq 0.8$	Fair
$0.6 < AUC \leq 0.7$	Poor
$0.0 \leq AUC \leq 0.6$	Fail

To construct our `ROC` and `lift` curve, we manually labeled the 2058 evaluated documents into two categories A and B: A for "food-risk documents" (relevant documents) and B for "non food-risk documents" (non relevant documents). We obtained: 12 documents for A and 2046 for B. The twelve relevant documents of category A were ranked by our filtering tool among the 40 first positions of the ranking. The 9 first positions of the filtering ranking corresponds to relevant documents (i.e., $\frac{9}{12} = 75\%$ of relevant documents were covered by the 9 first positions of the ranking). So we have two possibilities to choose the minimal threshold that will determine the documents to include in the eDot warehouse: the score of the document at position 9 (to be very precise) or the score of the document at position 40 (to cover all relevants). Knowing that all the relevant documents among the evaluated candidates were highly ranked, we built our validation approach using the 100 first positions of the ranking and we obtained the `ROC` and `lift` curves as shown in the diagrams 1 and 2 at page 9.

We can see from the increasing rate of the `ROC` curve of figure 1 that most of the relevant documents were covered in the first positions of the filtering ranking. The AUC corresponding to this curve is 0.9036 which shows the efficiency of our filtering approach according to the classification given above. After analyzing manually the content of a sample documents, our experts noticed that food risk documents obtained high scores whereas the non relevant ones obtains low scores. This is why the lift curve of figure 2 keeps the precision to 1 for the 9 first positions and then it decreases this precision to the final precision of the 100 sampled documents (i.e., $\frac{12}{100} = 12\%$). This shows experimentally that the numerical score of our filtering approach is a good estimator of the thematic-relevance of a document.

5 Conclusions

In this paper, we have presented "eDotFilter" a filtering tool that uses WeQueL, a multi-criteria web query language that is more expressive than simple

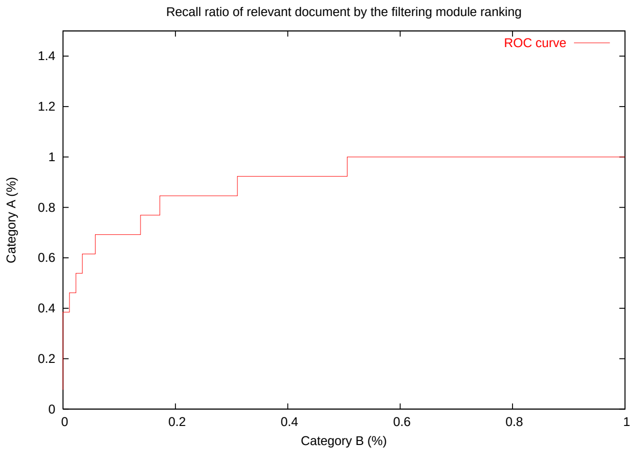

Fig. 1. ROC curve

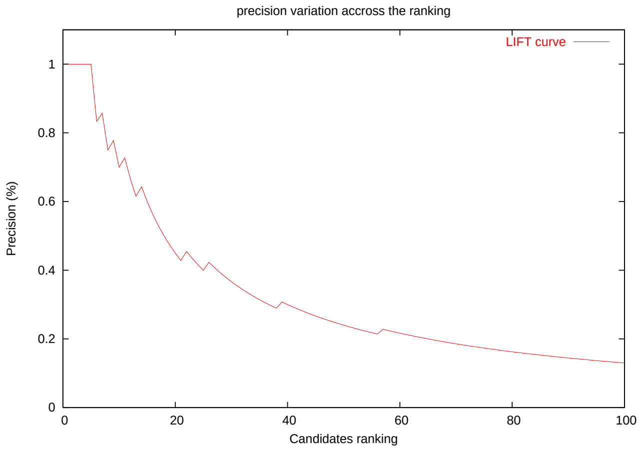

Fig. 2. Lift curve

keywords and less costly than learning the filtering function from relevant examples. This tool is used to filter the crawled candidates to include in the eDot food risk warehouse. We showed in our experiments the benefit of a wequel query having a numerical semantic in the estimation of the relevance of a document to the food risk thematic. 12 relevant documents among 2058 crawled candidates can seem very unsufficient to build a robust warehouse. This is due to the imprecision of the general-purpose crawler used to gather candidates for the eDot warehouse. Therefore, we are developping alternative crawling techniques that combines our filtering approach with a self-learning technique similar as the one used in the intelligent crawling [16] to fetch the web in a more efficient manner. The intended goal is to learn the topology of the web graph near the thematic documents so that the crawler can focus in a more efficient way its crawling priorities.

References

1. Halkidi, M., Nguyen, B., Varlamis, I., Vazirgiannis, M.: (2002) Organising web documents into thematic subsets using an ontology (THESUS). In: Actes électroniques des Journees Web Semantique, Paris
2. Brin, S., Page, L.: (1998) The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems **30**, 107–117
3. Rennie, J., McCallum, A.K.: (1999) Using reinforcement learning to spider the web efficiently. In: Proc. 16th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA 335–343
4. Diligenti, M., Coetzee, F., Lawrence, S., Giles, C.L., Gori, M.: (2000) Focused crawling using context graphs. In: 26th International Conference on Very Large Databases, VLDB 2000, Cairo, Egypt 527–534
5. Chakrabarti, S., van den Berg, M., Dom, B.: (1999) Focused crawling: a new approach to topic-specific Web resource discovery. Computer Networks (Amsterdam, Netherlands: 1999) **31**, 1623–1640
6. whizbang: Cora version 2.0 : Computer science research paper search engine (website) http://cora.whizbang.com.
7. CiteSeer: http://citeseer.nj.nec.com/cs (website)
8. Mezaour, A.D.: (2003) Focused Search on the Web using WeQueL. In: Proceedings of the 10th International Workshop on Knowledge Representation meets Databases (KRDB 2003), Hamburg, Germany. 63–74
9. Mezaour, A.D.: (2004) Recherche ciblée de documents sur le web. Revue RNTI-E2, numéro spécial EGC'2004 **2**, 491–502
10. Mezaour, A.D.: (2004) Filtering Web Documents for eDot, a food risk warehouse. In: Proceedings of the 2nd International Conference on Computational Intelligence (ICCI 2004), Istanbul, Turkey, Prof. Dr. Ali OKATAN 249–252.
11. Sym'Previus: Système de prévision du comportement des micro-organismes dans les aliments : http://www.symprevius.net/index.htm (website)
12. ComBase: Combined database for predictive microbiology: http://wyndmoor.arserrc.gov/combase/ (website)
13. Mezaour, A.D.: (2004) Numerical filtering query of edot : http://www.lri.fr/∼mezaour/wequel/edot/qnum.xml
14. Ferri, C., Flach, P., Hernandez-Orallo, J.: (2002) Learning decision trees using the area under the ROC curve. In: Proceedings of 9th International Conference on Machine Learning, ICML'02. 139–146
15. Nebraska university, the area under an ROC curve: http://gim.unmc.edu/dxtests/roc3.htm (website)
16. Aggarwal, C.C., Al-Garawi, F., Yu, P.S.: (2001) Intelligent crawling on the world wide web with arbitrary predicates. In: World Wide Web. 96–105