Mining Crawled Data and Visualizing Discovered Knowledge

V. Dubois, M. Quafafou, B. Habegger
IRIN, 2 rue de la Houssinière 44072 NANTES Cedex 03, FRANCE
{dubois, quafafou, habegger}@irin.univ-nantes.fr

Abstract

This paper presents a challenging project which aims to extend the current features of search and browsing engines. Different methods are integrated to meet the following requirements: (1) Integration of incremental and focused dynamic crawling with meta-search; (2) Free the user from sifting through the long list of documents returned by the search engines; (3) Extract comprehensive patterns and useful knowledge from the documents; (4) Visual-based support to browse dynamic document collections. Finally, a new paradigm is proposed combining the mining and the visualization methods used for search and exploration.

1 Introduction

Search of information from the web becomes more and more complex and there are many factors biasing the success of such information seeking processes [1]. In some situations keywords are not sufficient to express the desired information needs. An iterative process is often necessary to discover a useful piece of information. We think that the third search engine generation will be certainly "smart" integrated engines that enrich data web crawling and searching features with a collection of interactive, visual, mining and more generally intelligent functions to make both the search and exploration processes more dynamic, fruitful, understandable and collaborative. This paper contributes to achieve this "dream" by integrating different components to meet the following requirements: (1) integration of incremental and focused dynamic crawling with meta-search, (2) free the user from sifting through the long list of documents returned by the search engines, (3) extract comprehensive patterns and useful knowledge from the documents, (4) visual-based support to browse dynamic document collections.

Section 2 is dedicated to the dynamic aspects of searching and crawling. Section 3 describes the foundations of our web mining method which is based on probabilistic networks and functional dependencies. An incremental graph visualization method with a dynamic structure is given in section 4. Both mining and visualization methods are closely related as the result of each one is used to refine the results of the other. Section 5 presents experimental results and shows the interest of our approach considering the “mad cow” query. Finally section
6 is a conclusion that presents the current state of our work and underlines the main future developments.

2 Dynamic search and crawling

A query is generally reduced to a single or a small set of words and each URL in the returned list is referred to as a hit. In order to return such list in a reasonable time, search engines generally use a local index and/or local collection of web pages. Such indexes and collections are built by using a crawler to automatically create its index and collect web pages that are stored in huge databases. However, this approach faces different problems essentially related to the huge size of the web [5] and the storage of its complete image. The rate at which pages change is a second crucial problem to deal with. Predictive web changing models [7] are at the basis of the major strategies for re-indexing the expected modified pages. The web is updated by millions of users in an uncontrolled way. Important existing pages may be deleted and links to authority pages may be broken which increases the users frustration during web browsing.

The objective of dynamic searching and crawling is to resolve these problems. The difference of dynamic searching when compared to static searching is that it does not limit its results to those found in an outdated index but explores the web in its current state. Dynamic searching is limited by the efficiency cost, essentially the number of irrelevant downloaded pages, of a crawler that retrieves and analyzes pages on the fly. This type of crawling is often referred to as focused crawling in that it is guided by a given query or topic of interest. Building such efficient crawlers is a challenge for dynamic web exploration [2].

A crawling process can basically be split into two main subparts: (1) downloading a page and (2) extracting the links of a page. These two components communicate together by respectively sending to the other the newly downloaded pages and the last extracted links. In this scheme the crawling strategy can be defined in terms of filters placed before and after each of these parts.

The work presented in this paper only makes use of two types of filters which are sufficient to evaluate the idea we put forward. The first, a contextual filter, uses the nearness of a link to prune links related to servers which previously gave bad links. The second, a content filter, uses statistical methods to calculate a representative document vector and rejects any document whose vector doesn't contain one of the keywords. Of course, only the main words are selected according to both document-dependent and document collection-dependent measures according to term importance as used in automatic text analysis [6].

Putting together static and dynamic searching avoids the static and incomplete information stored in the databases of search engines while still having a set of a priori relevant documents to initiate the dynamic crawling. The usefulness of this hybrid approach is detailed in [5]. In this case the seed URLs used to start crawling are the result of a static search: the list of URLs returned by a query on a given subject to a classic search engine.
3 Dependencies and graph

A graph is a natural and universal structure giving an abstract representation of knowledge. From a semantic point of view, we want to represent the influence between words in a graph. As a starting point, we turned towards functional dependencies. Functional dependency is a key concept in database community. A functional dependency, noted $A \rightarrow b$ holds in a relation whenever the set $A$ of attribute values allows to determine the value of the attribute $b$. We use functional dependencies as a starting point to define a crude dependence graph $G(r)$

$$(a \rightarrow b) \in G(r) \Leftrightarrow (\exists A, a \in A, A \rightarrow b \text{ holds in } r \text{ and is minimal})$$

It is possible to refine this by using approximative dependency instead of classical one. But approximative dependency property is a boolean property: it either holds or doesn’t. So, we improve this definition by scoring potential dependencies. Interesting local scores are found in the bayesian networks community. We use such scores to evaluate approximative dependencies: if $A$ would be a good set of parents for $b$ in a bayesian network, we consider that $A$ carries important information on $b$, i.e. $b$ depends on $A$. In the case where $A = \{a\}$, MML (Minimum Message Length) score is equivalent to mutual entropy measure. This way, MML score may be considered as a generalisation of the mutual entropy.

Let $s(A, b)$ be the MML score associated to node $b$ and parent set $A$. This score depends on the data. We define the overall dependence (ov) between two nodes $a$ and $b$ by $ov(a \rightarrow b) = \max_{A \subseteq \Omega - \{b\}} s(A, b)$.

We define the dependence graph as the graph $G$ such that:

$$a \rightarrow b \in G \Leftrightarrow (ov(a, b) > 0) \land \left| \{ A \subseteq \Omega - \{b\} | s(A, b) > ov(a, b) \right| \leq n$$

The set of parents $A$ of a given node $b$ is the union of the at most $n$ best parent sets $P_i(b)$ according to $s$. Note that if $\forall A \subseteq \Omega - \{b\}, s(A, b) \leq 0$, then $b$ has no parents (no one is good enough as parent). Using a maximum number of parent sets prevents having a graph with too many links, and provides an easy way to control the number of edges. It takes advantage of the score, and it is not possible when using only classical (exact or approximative) dependencies.

Textual data can be easily converted to a data table using words as attributes. Instead of using exact MML scores, we perform an approximation, taking advantage of the binary nature of attributes. The most useful property of this approximation is computational. It is based on the principle that conditional information quantity $L(b|A)$ may be evaluated by counting the minimum error any predictor would commit if given the values of $A$ and guessing the values of $b$. This error is in turn evaluated statistically by data instance comparisons. For an extensive demonstration of this approximation properties, see [9].

Our MML-based local score is the following: $S(A, b) = L(b|A) - 2|A|$, where $L(b|A)$ is the mean encoding length for the value of $b$ for one instance, given the values of the instance on $A$. It is approximated by $L(b|A) \simeq - \log(e(b|A))$, where $e(b|A)$ is the least expected error probability on $b$ given $A$. $e$ is in turn approximated by $e(b|A) \simeq \frac{1}{2} - \frac{\sqrt{2 \cdot \mu(b|A_{\text{true}}) - 1}}{2}$, with $X_{\text{true}}$ as the event that two
random instances share the same value on attribute X. This last approximation requires a full page of calculus, which is part of [9]. If the dependency is exact \((e = 0)\), the given formulas are all exact. The higher \(e(b|A)\) grows, the less accurate is our approximation. Hopefully, our goal is to find the best dependencies.

4 Incremental Graph visualization

The first challenge encountered for automatically displaying a graph is the high number of nodes and links. Automatically displaying graphs is a computationally hard problem. Only a 2D visualization is considered, so, a bounded visualization space is easily mapped to the screen. The added constraints are mainly related to the visualization aspects: 1) Space: the graph spreads across all the available visualization space; 2) Edge length: the distribution of edge lengths has a low variance after a sufficient number of iterations; 3) Edge number: a minimum number of edges crossing each other.

In our approach words that are closely related appear in the same area of the map when the other words are moved away. This approach overlaps some proprieties of maps produced by Kohonen’s self-organizing algorithm (SOM). The interest of this kind of neural network has been shown for interactive exploration of document collections [3]. WEBOSM is a major system developed to organize automatically full-text document collections using the SOM algorithm. In addition, empirical studies compare maps generated by firstly the SOM algorithm and then by human subjects considering the same set of documents. Lin’s conclusions [4] during this study clearly shows the interest of maps for searching and browsing: (1) to assist users to spot an area of the display, (2) to help users to memorize the display structure and (3) to support user decisions. To display a static graph, SOM may be use by the following way: the input space is the visualization space, and the output space is the word space (given a point, the network return the nearest word in this space).

In addition to the previous displayed constraints, we need to handle dynamic aspects. First, data may arise at any time. Second, we have to display the graph quite often. The solution we chose is to rely on an iterative process that in turn uses the word positions as an heuristic to find best parents, and then uses the graph to find better word positions in the 2D visualization space. The main advantage is that already learned positions are used as an heuristic to compute the new network topology whenever data changes.

5 Experimental results

An important task is the Global Multi-level structure exploration. We consider here around 440 documents the crawled and analyzed. All the information contained is analyzed and a probabilistic network is learned structuring the information contained in all crawled documents. As we have said before, the dynamic modification of this structure is guided by two criteria. The first one is the search for the graph optimizing a scoring function, whereas the second is related to its
visualization. The graph with the best appearance is kept. Multi-level explorations can be achieved by our proposed system. In fact the user may explore the global structure of the whole graph or only focus his attention on a local part of the structure represented by a connected component. In an example where we have set the query to “mad cow” and selected the words “cow”, “cows” and “hormones”, only connected components containing at least one word from the selected ones are drawn. It involves other words like “British”, “organs”, “danger”, “contamination”. This information and the structure can play an important role in querying reformulation and textual data analysis.

6 Conclusion

We have presented in this paper a dynamic search and exploration system helping to find useful information and to understand events and phenomena. We propose here an integrated environment coupling different dedicated methods as dynamic search, mining and visualization. A first prototype was developed using both C++ and Java. We have to further develop both theoretical and practical aspects of this work. Currently, we are improving the synchronization between our different processes especially the ones related to the crawling task. More sophisticated applications have been developed.

References

8. Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Srikhar Rajagopalan, Raymie Stata, Andrew Tomkins, Janet Wiener, Graph structure in the web, WWW9 conference 2000.
9. V. Dubois and M. Quafafou, Discovering Graph Structures in high Dimensional Spaces. In Data Mining II (proceedings of Data Mining 2000, Cambridge)