Structured and Unstructured Search in Enterprises

Prabhakar Raghavan
Verity, Inc.
pragh@verity.com

1 Introduction

It is estimated that about a third of the time of a typical enterprise knowledge worker is spent searching for information. Such search and its derivative information retrieval functions are essential components of the infrastructure of *enterprise information portals*, which are the primary means through which enterprise employees access information throughout their business. We begin by listing some essential functions of such software, pointing out where appropriate how such functionality differs from that found in current *web* (as opposed to enterprise) portals that most readers may be familiar with.

1. The need to access information in diverse repositories including file systems, web servers, Lotus Notes, Microsoft Exchange, content management systems such as Documentum, as well as relational databases. In contrast, most web portals deal only with html content.

2. The need to respect fine-grained individual access control rights, typically at the document level; thus two users issuing the same search/navigation request may see differing sets of matching results due to the differences in their privileges. In contrast, most web services have only a single class of access control, or in some cases access levels that form a strict subset hierarchy (as in subscription services with multiple levels of subscription).

3. The need to index and search a large variety of document types (formats), such as PDF, Microsoft Word and Powerpoint files, etc., in many different languages.

4. The need to seamlessly and scalably combine structured (e.g., relational) as well as unstructured information in a document for enhanced navigation and discovery paradigms.

5. The need to combine search results from internal as well as external sources of information.

6. The integration of search with functions like taxonomy building, classification and personalization.

2 Verity's K2 Enterprise

We briefly review how these features are addressed in Verity's flagship K2 Enterprise product. We touch upon classification and personalization in this section, deferring a detailed discussion of structured versus unstructured search to Section 3. For each type of repository that is to be accessed (e.g., Documentum, Lotus Notes, databases through ODBC interfaces), Verity provides a *gateway*. The role of the gateway is to allow a spider to access the content in the repository, together with the associated security information (i.e., which users can access which documents). Each repository may contain documents in many different file formats; K2 Enterprise provides

Copyright 2001 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

filters to automatically detect and generate text versions from over 250 document formats. Using gateways and filters, the spider generates a full-text index that consists of two parts: (1) a positional full-text index that records, for each word, every occurrence in each document — such fine-grained positional information is necessary to handle phrase and proximity queries (e.g., *IBM* within 4 words of *Armonk*); (2) a structured index that records, for each document, the meta-data associated with it (much as a database table does).

We now highlight three specific capabilities likely to be of interest to this readership.

1. Arbitrary combinations of text and range queries. Consider the query: *Give me documents that contain the phrase "web server", are of type "Design document", language "German", format "sgml", price* < 10000. Clearly this query has a text/unstructured component (to match the phrase "web server") as well as a structured component. We discuss how Verity handles this type query in Section 3, and how this would differ from a solution that might use an RDBMS.

2. Automatically classifying documents by textual content as well as structure derived from meta-data. Adding structure to unstructured information is key to enhancing the value of enterprise content. Classification is a key step in such structure enhancement. This is briefly discussed in Section 2.1.

3. Delivering personalized views of information by re-ranking search results, recommending documents, and locating experts on the query subject. Both explicit profile information and implicit behavioral patterns are exploited in the personalization engine. This is briefly discussed in Section 2.2.

2.1 Automatic Classification

While searching provides an efficient way for users to find relevant information in business portals if they know what to search for, there is a different need for browsing and navigating information. Taxonomies are the most popular way for organizing documents into navigable structures. With a taxonomy, users can easily navigate through the category hierarchy to find relevant information. Scoped search within a category typically produces more relevant results than un-scoped search.

There are two main phases in deploying taxonomies in a portal environment: taxonomy construction, and maintenance. The former defines the category hierarchy, and the latter populates documents into the taxonomy once it is built, and modifies the taxonomy structure when needed. Verity refers to a populated taxonomy as a *knowledge tree* which consists of a taxonomy of browsable categories and databases that store the relationships of documents and categories. Fully automatic taxonomy construction and maintenance often leads to unsatisfactory results. Consequently, most taxonomies are built and maintained manually by human experts (examples include the directory structures of Yahoo! and the Open Directory Project). However, manual population of a large volume of documents is expensive.

Verity's Intelligent Classifier [5] provides several taxonomy construction methods, each of which can also be used to categorize documents into a taxonomy: (1) Construct a taxonomy manually through its graphical user interface. (2) Extract a hierarchy of categories of a taxonomy from URL paths or file paths. (3) Fetch categories from indexed meta data. This method can be used when categorical information is explicitly listed in a field in the collection. (4) Generate a taxonomy from document clusters produced by Verity's clustering algorithm, which groups similar documents into clusters. A hierarchy of clusters are generated by recursively breaking large clusters into smaller clusters.

Verity Intelligent Classifier provides the following methods for defining classification rules for each category in the taxonomy. (1) Manually construct classification rules using the Verity Query Language (VQL); (2) Learn classification rules for categories automatically from exemplary documents associated with these categories in the taxonomy; (3) Interactively refine classification rules by providing relevance feedback to the test results of previously built rules. The core technology that supports these automatic features is Verity's regularized Logistic Regression Classifier (LRC). LRC automatically learns a classification rule from a set of documents

that are labelled as relevant or irrelevant to a category. Let a document be represented by a feature vector $\mathbf{x} = [t_1, t_2, \cdots, t_d]^T$, and r be the relevancy measure of the document with respect to the topic, $0.0 \leq r \leq 1.0$. Given a set of relevant documents and a set of irrelevant documents for a category, the LRC learning algorithm learns a regression function $log(r/(1-r)) = w_1 t_1 + w_2 t_2 + \ + w_d t_d + b = f(\mathbf{w}, \mathbf{x})$, such that the separation between the relevant and irrelevant documents is maximized. Structure risk minimization theory [6] guarantees a minimized upper bound on the classification error on future documents.

Once the regression function is determined, a future document can be assigned to the category if the relevancy score r for the document is greater than a pre-specified threshold. This classification rule can be conveniently represented by Verity's powerful query language. Our experiment shows that on Reuter's benchmark data set, LRC achieves the state-of-the-art performance at 88% precision-recall break-even rate [1].

Once a knowledge tree is defined in Verity Intelligent Classifier, it can be exported to Verity Knowledge Organizer [2]. Administrators can configure how to display search results and the category structure to end users. End-users can browse through the documents in a collection by category and drill down to a subject of interest. They can also limit the scope of their searches only to the categories of interest.

2.2 Personalization

The Verity Personalization Engine enhances the end-user's information discovery experience to the next level beyond search and taxonomies. It uses a tensor space model that represents different entities in the system such as products, documents, users and queries as tensors in a tensor space. These vectors adapt to latent patterns in user behavior in order to dynamically personalize the results of subsequent searches in different ways, as outlined below. The engine builds and maintains a profile for each user, based on a notion of a *transaction* that updates the profile. Some examples of transactions are:

User <**joe**> bought <**Toaster X**> upon query <**"toaster"**>

User <**john**> viewed <**Document Y**> upon query <**"personalization"**>

The user profiles are used to support the following features: (1) Adaptive Ranking – repeated selection of an item for a given query causes the relevance of the item to be boosted for all users on similar queries. (2) Document Recommendation – products/documents are recommended based on a combination of the current query and the user information, based on the user's past behavior. (3) Document Similarity – documents or products that are similar to a selected item are recommended. (4) Expert Location – based on a document that is being viewed, or a query that has been issued, experts/users in the organization are recommended. (5) Community – a dynamic user community can be presented, based on the current user's profile.

3 Text Versus Structured Queries

We begin by highlighting some of the issues in text versus structured search in the enterprise. We mentioned earlier that the index used by a typical enterprise search engine included support for structure as well as unstructured search. We first address the question: how is the structured information typically derived? There are three principal means: (1) from meta-data supplied with the document, say in XML form; (2) using classification techniques such as those described in Section 2.1; (3) by spidering a relational database. In Section 3.1 below we discuss some settings in which it is advantageous to search structured information outside an RDBMS. We pause to highlight a second aspect here: spidering an RDBMS is most useful in the case of, say, catalog information in a B2C or B2B setting. However, most deployments of relational databases entail an *enterprise application* built on top of the database – for supply-chain or customer relationship management, as examples. In these cases, simply spidering the database loses sight of the business logic embedded in the application. Vendors are now attempting to build "application gateways" that spider *through the application*, in the hope of preserving some of the business logic. Such efforts still appear to be in their early stages.

3.1 Combined Text and Structured Querying

In the most general settings, each document has some unstructured text as well as a number of structured attributes. Some of these attributes may assume numerical values; for instance, a description of an automobile might have the year in which the car is manufactured and its price. A typical query is a conjunction of an arbitrary text query with an arbitrary range query. In many applications, it is desirable to retrieve and rank documents that simultaneously meet both the unstructured and the structured query components, without recourse to two retrieval systems. Using an RDBMS to solve the problem would result in query responses that would be unacceptably poor. Furthermore, an RDBMS cannot support many features of a full-text engine such as query-time spell correction and proximity searching (mentioned earlier). (Needless to say, this solution does not provide for the numerous other features a typical RDBMS provides, such as logging, recovery, etc.) In addition to retrieving documents that meet the text and parametric queries in a scalable fashion, it is important to be able to rank the results not only by text query scores, but also by sorting along field values (e.g., price). This allows for the efficient navigation of a results list, allowing the user to further refine (or relax) the query at hand.

Our solution consists of augmenting a full-text index with an auxiliary parametric index that allows for fast search, navigation and ranking of query results. We now describe the elements of this index, and how they are organized in order to support all of the operations described above. Each field is represented as a set (known as a BucketSet) and each unique item in that field is represented as a member of that set along with position and frequency information (this datum is known as a Bucket). A set of BucketSet structures make up a completed structure known as the parametric index which maps directly to the corpus from which it was generated. Extra meta-data are stored along with the BucketSets to allow ranges (either text, numeric or date) to be extracted.

It is the mechanism by which the BucketSets and Buckets interact (through their parametric representation) that allows complex set operations (based on nested intersects and unions) to be applied to them. This mechanism results in: (a) real time cardinality statistics for these operations, and (b) an iterative refinement of the operations applied to the database, which can result in the reduction of old Buckets and the inclusion of new Buckets (depending on the semantics applied). In the simple case where A and B are both Buckets of the same set, "$A \bigcap B$" can remove items, whilst "$A \bigcup B$" can add items.

We can also express the text search in the parametric domain (i.e., a text search maps to a BucketSet where each item (hit) is considered as a separate unique Bucket in that set with no context until applied to a parametric index). Both kinds of BucketSets can interact with the same set of operators. Extra speed is obtained by performing query optimizations in real time – for instance, by examining the relative sizes and complexities of the BucketSets when intersecting them and ensuring that the smallest or least complex controls the query optimization. The text search component is built on Verity's VDK kernel [3] with a K2 search engine on top [4]. The approach has been shown to scale to millions of documents with 6-10 fields of structured attributes. Interactive performance is feasible for several concurrent users, using a standard desktop PC as the server.

References

[1] S.T. Dumais, J. Platt, D. Heckerman and M. Sahami. Inductive learning algorithms and representations for text categorization. *Proc. ACM CIKM*, 1998.

[2] Verity, Inc. *Knowledge Organizer,* v. 2.0, 2000.

[3] Verity, Inc. *Verity Developer Kit (VDK): APR Reference Guide.* v. 3.1, 2000.

[4] Verity, Inc. *K2 Toolkit: API Reference Guide*, v. 2.0, 1999.

[5] Verity, Inc. *Intelligent Classifier*, v. 2.6. 2001.

[6] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.