

Extracting Relations from XML Documents

Eugene Agichtein^{1†}, C. T. Howard Ho², Vanja Josifovski², and Joerg Gerhardt^{2†}

¹ Columbia University, New York, NY, USA
eugene@cs.columbia.edu

² IBM Almaden, San Jose, CA, USA
{ho,vanja}@almaden.ibm.com

Abstract. XML is becoming a prevalent format for data exchange. Many XML documents have complex schemas that are not always known, and can vary widely between information sources and applications. In contrast, database applications rely mainly on the flat relational model. We propose a novel, partially supervised approach for extracting user-defined relations from XML documents with unknown schema. The extracted relations can be directly used by an RDBMS, or utilized for information integration or data mining tasks. Our method attempts to automatically capture the lexical and structural features that indicate the relevant portions of the input document, based on a few user-annotated examples. This information can then be used to extract the relation of interest from documents with schemas potentially different from the training examples. We present preliminary experiments showing that our method could be capable of extracting the target relation from XML documents even in the presence of significant variations in the document schemas.

1 Introduction

XML provides a standardized format for data exchange where relationships between entities are encoded by nesting of the elements. XML documents can have complex nested structure, while many applications prefer a simple and flat representation of the relevant information. Extracting information from XML documents into relations is of special interest, since the resulting relations would allow the use of SQL and the full power of RDBMS query processors. In such a scenario, a *mapping* is needed to specify the extraction of the required portions of XML documents to relations. Mapping specification is usually performed by an experienced user with knowledge of the content of the input document and resulting relations. If detailed description of the document structure is available in advance, a mapping can be defined once and used over all of the input documents.

In the case when the XML documents originate from a number of different sources with variations in their schema, or when the schema evolves

[†] Work done while visiting IBM Almaden.

over time, the mapping specification process can be long and labor-intensive. The user needs to provide a mapping for each new source, and update the queries as the document structures change. In order to relieve the user of this tedious task, we propose a system for mapping from XML to relations by generalizing from the user-provided examples, and applying the acquired knowledge on unseen documents with flexibility to handle variations in the document input structure and terminology (tag names). Such documents may be derived from HTML pages, or from business objects exported to XML. For example, consider the task of compiling a table of product prices and descriptions from different vendors, where each vendor exports their product catalogs as XML documents. These documents may encode the prices of products in a variety of ways, using different tag names and structures. With current technology, each vendor source would have to be wrapped manually to extract the tuples for the target table. Being able to extract key relations from such XML documents using a few user-specified examples would reduce the system setup time and allow for improved robustness in a case of schema change.

Our partially supervised approach is an adaptation of the general *nearest neighbor* classification strategy [3]. In this approach, the candidate objects are compared with a set of “prototype” objects. The candidates that are the closest to a prototype p are classified into the same class as p . In our setting, the goal of the classifier is to recognize the nodes (if any), in a given XML document that are needed to extract (map) the information in the document to the given target relation. The prototype objects are constructed based on the user-annotated example XML documents, and correspond to the nodes that contain the attributes to be mapped to the target relation. The similarity between the candidate nodes and the prototype nodes is computed using *signatures* that represent the position, internal structure, and the data values of the document nodes.

Preliminary experiments indicate that our method can be used to reliably detect relevant elements in unseen documents with similar, but different structure. The use of signatures as opposed to queries allows more flexibility in the structure captured from the training examples. For example, the terms in the signature can be related in a way that does not match the XQuery axes, or weights can be assigned to individual terms, which would allow specifying increased importance to some of the terms. Such features are not available in today’s XML query and transformation languages.

Related Work

Several commercial and research databases support mapping XML documents into user-defined relations. These mappings are specified by using XPath expressions and apply only to documents with schemas compatible with the expressions. If the schema is not available, a system such as XTRACT [2] can be used to infer a DTD. The documents in the collection are assumed to have the same structure, and elements can be described independently. Some systems allow building a summary

of several XML documents, as for example the DataGuides [4] techniques that emerged from the LORE project. A related approach taken by STORED [1] uses data mining techniques to store the information in an RDBMS for efficient querying. In contrast, we assume a given user-defined relation to which we want to map XML documents with variable schema.

Several interactive tools have emerged that allow mapping from XML documents to relational schemas, as for example [8]. The mappings produced by these tools are used for both shredding and storing XML documents and for view generation for online querying [5]. The techniques we use in our work draw on methods developed for the extraction of structured information from HTML and plain text, notably [7, ?, ?, ?, ?, ?]. The rest of the paper proceeds as follows: In Section 2 we present an overview of our system and describe our data model. In Section 3 we describe our method for generating and using *signatures* for extracting a relation from new XML documents (Section 4). We then present preliminary experimental results in Section 5 and conclude the paper in Section 6 with a description of our current activities and future work.

2 System Overview and Data Model

We use a partially supervised approach to extract a user-specified relation from XML documents based on a few user-tagged examples. The system works in two phases, *Training* and *Extraction*, shown in Figure 1. In the *Training* phase, the system is trained on a set of XML documents where the user has indicated the correct mapping of the XML elements to the attributes of the target table. The result of the training stage is a set of *signatures* that are used in the subsequent *Extraction* stage. During the *Extraction* phase, the target table is extracted from new XML documents that may have different tag names or structure than the example documents. As the first step of the extraction stage, the nodes of the input documents are *merged* in order to generate a “canonical” representation of the input document. Then, the signatures generated during training are used to find the candidate nodes in the canonical representation of each input document that are most likely to contain attributes for the target relation tuples. Finally, the mapping from the descendants of this node to the target attributes is derived. The resulting mapping can be translated trivially into XPath expressions to extract the tuples from the input document, or from any document with the same structure.

2.1 Data Model

Our system extracts a single *target relation*, $T(a_1, a_2, \dots, a_n)$, from a collection of XML documents. Representing the input XML document as a tree of nodes, each tuple $t \in T$ is extracted from a document subtree rooted at a node called *instance node*. More formally, we define an instance node I as a document element such that:

1. Children of I contain complete information needed to extract exactly one tuple of the target table.

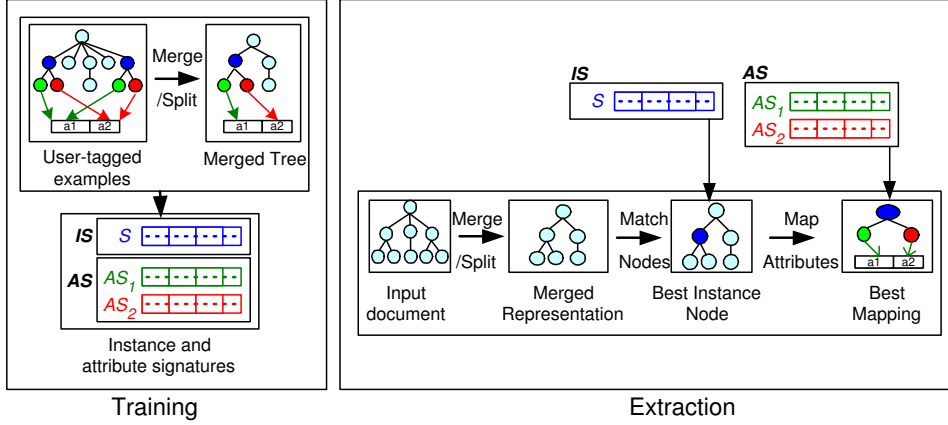


Fig. 1. Overview of our system: In the Training stage the system derives instance and attribute signatures used for extracting the target relation from a new XML document.

2. I is maximal, i.e., any ancestor node of I will contain complete information for more than one tuple in T .

Figure 2 illustrates the role of the instance node in a document representing a set of books, such as one that may be exported by a book vendor. The target relation is defined as $NewBooks(ISBN, BookTitle, Author, Price)$. The node *Item* in the *Books* category, shown, contains all the information in the attributes of its descendants that is needed to extract a tuple for the target relation. Therefore all the *Item* elements shown in this example are instance nodes.

The extraction of the relation from a new document d consists of first identifying a node in d that corresponds to I , and then mapping descendants of I to the attributes of the target relation. We now present our approach for automatically generating flexible signatures that can be used to recognize instance nodes in new XML documents with variations in label names and structure.

3 System Training: Generating Instance and Attribute Signatures

Our approach of deriving instance and attribute signatures to extract a target relation uses as input a set of user-supplied example XML documents. First, we pre-process the input documents to derive a *merged* document representation (Section 3.1). As we will discuss, the merged representation will allow us to describe the example documents more completely. We then generate *instance signatures* (Section 3.2) that capture the position and internal structure of the instance nodes. Then we describe *attribute signatures* (Section 3.3) that capture the structural

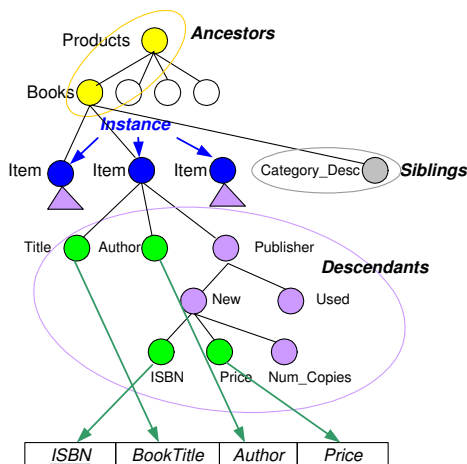


Fig. 2. Example of Extracting a Table from XML documents.

and data characteristics of the nodes that correspond to the attributes of the tuples in the target relation.

The training begins with a set of annotated example XML documents, with special tags specifying the instance nodes and the attribute mappings.³ For each example we solicit two types of input from the user:

1. Identify the instance node I (e.g., the *Item* node)
2. Identify the descendants of I that contain the values for the attributes in the target relation.

In all machine learning approaches, one of the major problems is data sparsity, where the manually annotated examples do not fully represent the underlying distribution. Some of the nodes in the initial examples may be optional (and therefore missing), and the data values may not be repeated enough across the remaining attributes to generate a reliable signature. Therefore, we propose *merging* the nodes in the input documents to create the “canonical” representation of the document tree as we describe next.

3.1 Merging Nodes in the Input Document

A relational table usually represents a set of similar entities, such that each entity corresponds to one tuple. We can therefore expect that an XML document mapped to a table will also contain a set of nodes for

³ In our prototype we use reserved XML element names to specify the mapping. This allows for use of XML parsing and processing over the training documents. The annotated documents can be produced from the original documents by a user using a GUI tool.

a set of related entities. Intuitively, the nodes representing the same class of entities will have similar structure and relative position in the document. Often such XML documents will be produced by a single source and will have some regularity in their structure. We can exploit this regularity *within a single document* by merging “similar” nodes. As a result, we will have more rich signatures and reduce the complexity of the subsequent extraction phase. More importantly, merging nodes in the input document will allow us to reduce noise (e.g., missing optional nodes), resulting in more complete signatures (Sections 3.2 and 3.3) that will later be used for extraction.

The Merge Algorithm: Our procedure for merging nodes is shown graphically in Figure 3. Intuitively, sibling nodes with the same tag name, and with similar internal structure can be assumed to represent similar objects. Using this observation, we merge sibling nodes that share the same prefix path from the root, and have similar internal structure. The user-annotated instance nodes are merged just as any other nodes, resulting in the more complete examples of the instance nodes for signature generation. Our algorithm proceeds in two stages: First we *Merge* all nodes that share the same prefix path from the root, and then we *Split* the nodes in the resulting tree that are too heterogeneous.⁴

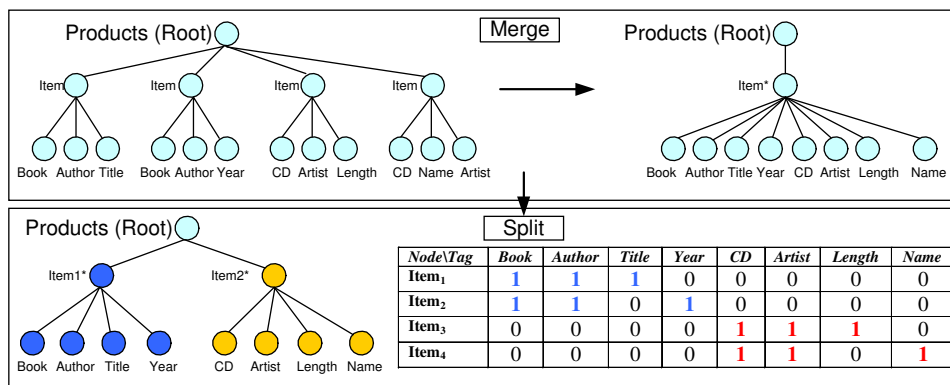


Fig. 3. Operation of the Merge algorithm for merging similar nodes in the input document.

Merge We traverse the input tree in a *top-down* fashion, recursively merging siblings with the same label into one supernode. In the example,

⁴ In practice, it would be more efficient to avoid merging nodes with completely heterogeneous internal structures. For clarity and generality, we present a two-step implementation.

all siblings *Item* that have the same label are merged into *Item**. The children of *Item** are the union of children of each original *Item* node. Currently we only merge nodes at the same level. In the future, we may want to merge nodes that have the same label, but occur in slightly different depths in the input tree. It is not clear if this is a desired behavior and most likely depends on the application.

Split In some XML document, sibling nodes with the same tag might have completely different structure. The goal of the split phase is to correct the merged nodes generated in the previous phase. This process allows us to distinguish between nodes that are semantically equivalent but happen to have *missing information*, and nodes having the same label, but which are actually *heterogenous*. The main criterion for splitting is whether there are disjoint subsets among the set of children of the merged node. In the example above, the merged node *Item** contains 2 disjoint subsets ((Book, Title, Author, Year) and ((CD, Artist, Length, Name)). Thus, *Item** would be split into 2 nodes, *Item₁** and *Item₂**. The split procedure splits the nodes in the merged tree in a *top-down* fashion. At each node, the set of the children is examined. If the set contains at least two disjoint sets of children, the current node is split, and children are allocated accordingly. Finding the disjoint sets of children can be done efficiently by using the matrix shown in Figure 3. In this matrix, a “1” in the position *i, j* indicates that a node *i* contains a child with the label *j*. Using the matrix we can quickly find the connected (and disjoint) entries. This approach can be extended to splitting nodes that are *weakly* connected, and not completely disjoint.

As we discussed, the purpose of merging is to create a more complete representation of the input document. We now describe how we use this representation to generate *instance* and *attribute* signatures that we will use subsequently for extracting the target relation from new, previously unseen documents.

3.2 Instance Signatures

Recall that our goal is to generate signatures that will allow us to find instance nodes in new documents with both structure and label names potentially different from the example documents. To support such flexibility, we need to capture both the *position* in the document and *internal structure* of the instance node. Further, the representation of the signature should be such as to allow finding the instance node in documents with structure and tag names different from the example documents observed in the *Training* stage. To accomplish this, we divide the document tree into four regions:

1. **A:** *Ancestors* of *I* (some number of levels up the tree).
2. **S:** *Siblings* of *I*.
3. **C:** *Descendants* of *I*.
4. **I:** *Self*: Tag of instance node *I* itself.

The *Siblings* and the *Ancestor* nodes intuitively describe the position of the instance node in the document. The *Descendants* component allows us to describe the internal structure of the instance node.

From these tree regions, we build the *instance signature* S of each example. We represent S as a set of vectors $S = \{\langle A, S, C, I \rangle\}$ where each vector represents the respective tree region. More specifically, we represent each tree region using the tag names of the nodes in the region just like the vector-space model of information retrieval represents documents and queries [9]. Recall, that in this representation each unique tag name corresponds to a dimension in the vector space, and therefore the order of the tag names in the input document can be ignored. For example, the vector A generated as part of signature to represent the *Ancestors* region for the *Item* node in Figure 2, would contain terms $\langle Products, Books \rangle$. In future work, we plan to investigate different weighting schemes for the terms in the vector. We could also use other reported techniques for representing XML structures in a high dimensional vector space, e.g., [6], but it is not clear which representation would work best for our application. Therefore, for our initial experiments we chose the minimal representation described above.

3.3 Attribute Signatures

So far we have discussed the characterization of the position and internal structure of the *instance node*. These signatures would allow us to find instance nodes despite variations in the structure of the document. Similarly, we want to support variations in the internal structure of the descendants of the instance node that map to the attribute values in the target relation.

To capture the characteristics of the attributes of the target relation as they appear in the example documents, we build an *attribute signature* $AS(\{D\}, S\{A, S, C, I\})$, for each attribute of the target relation, which consists of two components:

- 1: Data signature D for the column over all known instances of the attribute to represent the distribution of values expected to be found in each attribute. (We can use a technique similar to the one described in [8].)
- 2: Structure signature $S(A, S, C, I)$, defined equivalently to the instance signature S , where the current instance node is used as the document root, and I refers to the set of tags of all elements in the example documents that map to this attribute.

We will use these signatures to map descendants of instance nodes found in test documents to attributes in the table.

3.4 Signature Similarity

We now define the *similarity* of signatures that we will use to extract the target relation from new documents (Section 4). Intuitively, signatures of nodes that are located in similar positions in the document tree and have similar internal structures should have a high similarity value. For this, the similarity measure should consider all components of the signature.

More formally, we define the *Similarity* between signatures $Sig_i(A, S, C, I)$ and $Sig_j(A, S, C, I)$ as:

$$\begin{aligned} Similarity(Sig_i, Sig_j) = & w_A \cdot Sim(A_i, A_j) + w_S \cdot Sim(S_i, S_j) \\ & + w_C \cdot Sim(C_i, C_j) + w_I \cdot Sim(I_i, I_j) \end{aligned} \quad (1)$$

where $Sim(a, b)$ is defined as $\frac{a \cdot b}{|a| \cdot |b|}$, or cosine of the angle between vectors a and b , which is a common way to quantify similarity in information retrieval.

The *Similarity* function combines the *Sim* values between the positional and structural components of the signatures. Currently, all the components of the signature are weighted equally. However, depending on the application needs, the relative importance of different tree regions (as reflected by the weights of their respective vector components, e.g., w_A), may be tuned either by the user, or by using machine learning techniques. We define similarity between attribute signatures equivalently to the way we define similarity between instance signatures (Equation 1). The only difference is that we also add the similarity of the respective data components (vector D in the attribute signature definition). Relative importance of the structural and data components of AS has been studied previously in the context of relational schema mapping in [8].

4 Extraction

Having derived the sets of instance signatures (IS) and attribute signatures (AS), we proceed to extract the target relation from the new, previously unseen XML documents. The extraction proceeds in three stages. First, similar nodes of the input document are merged using the Merge algorithm (Section 3). Then, the instances nodes in the merged document representation are identified. Finally, the descendants of the discovered instance node are mapped to the attributes of the target relation.

Identifying Instance Nodes To discover the most likely instance node, we traverse the merged document tree in a bottom-up fashion. For each node X we generate the instance signature S_X . We then compute the similarity of S_X and each instance signature in IS that was generated during training. The score of X is then computed as the maximum of these similarities. The node with the highest score is then chosen as the candidate instance node.

Mapping Attributes For each target column T_i , we compute the similarity between the attribute signature AS_i and the value of each descendant of the candidate instance node. Since merged instance nodes are expected to have small number of descendants, and the target table a relatively small number of attributes, this exhaustive approach is feasible. The mapping that maximizes the total similarity, computed as the product of similarities over all the target attributes, is chosen as the best mapping.

We can use the results of this step as feedback to the previous step of identifying instance nodes. For example, if we cannot find a satisfactory attribute mapping from the best candidate instance node, we then try the candidate instance node with the next highest score.

5 Preliminary Experiments

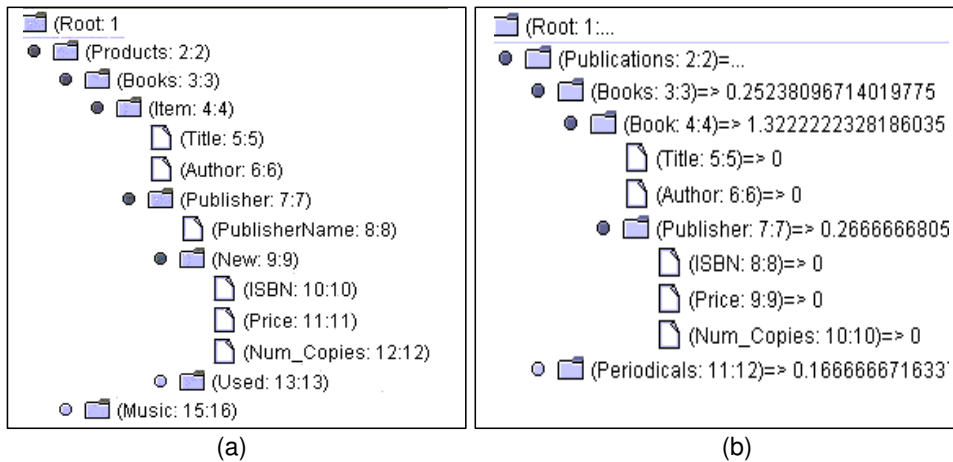


Fig. 4. The merged representation of the training document (a), and a test document (b) document with scores for each potential instance node. The node *Book*, the target instance node, is assigned the highest score by the system.

For our exploratory experiments we have considered a scenario where the target relation is *NewBooks*, described in Section 2. We want to extract this relation from XML documents such as may be exported by book publishers and vendors. We used the same tagged example document as shown in Figure 2 for training our system. The instance node in the example is the *Item* node, as displayed by our system prototype in Figure 4(a). Our system uses this example to generate the instance and attribute signatures to be used for extraction.

The original XML document structure and the tag names were modified significantly to create test documents. A sample modified document is shown in the merged representation (Figure 4(b)). The instance node that contains all the information needed to extract a tuple for *NewBooks* now has the tag name *Book*, and the *Products* node now has a new tag name *Publications*. Additionally, internal structure of the instance node was changed. Such variations in structure would break standard XPath expressions that depend on the element tags in the document to find

the instance node. However, our system prototype consistently assigned the highest score to the correct instance nodes in all tested variations, including the test structure shown in Figure 4(b). These exploratory results are encouraging and we are currently working on a more extensive empirical evaluation of our approach.

6 Conclusions and Future Work

We have presented a novel approach for partially supervised extraction of relations from XML documents without consistent structures (schemas) and terminologies (tag names). These XML documents may be derived from HTML pages, or obtained from exporting business objects to XML format. Extracting relations from schema-less XML documents using the approach presented in this paper can speed deployment of web-based systems and make their maintenance easier in presence of evolving schemas. We introduced the concept of the instance node, which is crucial in identifying the target node (object) that contains information for the attributes of the target relation. Second, we partitioned the neighboring nodes of the instance node into three different regions (siblings, ancestors, and descendants) and derived their respective signatures. We then defined a classification model based on spatial proximity of these tree regions to the instance node, each region having different semantic associations with the instance node. Third, the relative influence of these regions in finding the instance node in new documents can be simply adjusted by “turning a knob”, i.e., by changing the weights of the corresponding components in the similarity calculation. Finally, the Merge algorithm described in Section 4.1 enables our system to capture the notion of semantically equivalent XML nodes, which have stronger semantics than simply having the same tag names.

We are currently exploring extending our model to allow the user to identify *hint* nodes - those nodes that do not contain information for attributes in the target relation, yet may indicate presence of the instance node. We also plan to experiment with different signature representations and weighting schemes, and alternative similarity definitions.

References

1. Alin Deutsch, Mary Fernández, and Dan Suciu. Storing semi-structured data using STORED. In *SIGMOD*, 1999.
2. Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, and Kyuseok Shim. XTRACT: a system for extracting document type descriptors from XML documents. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 165–176, 2000.
3. Geoffrey W. Gates. The reduced nearest neighbor rule. In *IEEE Transactions on Information Theory*, 1972.
4. R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Twenty-Third International Conference on Very Large Data Bases*, pages 436–445, 1997.

5. Vanja Josifovski and Peter Schwarz. XML Wrapper - reuse of relational optimizer for querying XML data. *Submitted for publication.*, 2002.
6. Dao Dinh Kha, Masatoshi Yoshikawa, and Shunsuke Uemura. An XML indexing structure with relative region coordinate. In *ICDE*, pages 313–320, 2001.
7. Craig A. Knoblock, Kristina Lerman, Steven Minton, and Ion Muslea. Accurately and reliably extracting data from the web: A machine learning approach. *IEEE Data Engineering Bulletin*, 23(4):33–41, 2000.
8. Renee J. Miller, Mauricio A. Hernandez, Laura M. Haas, Ling-Ling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.
9. Gerard Salton. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley, 1989.