21

Mining Text Data

Ronen Feldman
ClearForest Corp. & Bar-Ilan University, Israel

INTRODUCTION

Data mining and knowledge discovery seek to turn the vast amounts of data available in digital format into useful knowledge. Classic data mining concentrates on structured data stored in relational databases or in flat files. However, it is now clear that only a small portion of the available information is in a structured format. It is estimated that up to 80% of the data available in digital format is nonstructured data. Most notably, much of information is available in textual form with little or no formatting. Hence, the growing interest in text mining, which is the area within data mining that focuses on data mining from textual sources.

The first issue to address when performing text mining is to determine the underlying information on which the data mining operations are applied. Our approach to text mining is based on extracting meaningful concepts that annotate the documents. A typical text mining system begins with collections of raw documents without any labels or tags. Initially, documents are automatically labeled by categories and terms or entities extracted directly from the documents. Next, the concepts and additional higher-level entities (that are organized in a hierarchical taxonomy) are used to support a range of Knowledge Discovery in Databases (KDD) operations on the documents. The frequency of co-occurrence of concepts can provide the foundation for a wide range of KDD operations on collections of textual documents, such as finding sets of documents with concept distributions that differ significantly from those of the full collection, other related collections, or collections from other points in time.

A straightforward approach is to use the entire set of words in the documents as inputs to the data mining algorithms. However, as was shown by Rajman and Besançon (1997), the results of the mining process in this approach are often rediscoveries of compound nouns (such as that "Wall" and "Street" or that "Ronald" and "Reagan" often co-occur), or of patterns that are at too low a level to be significant (such as that "shares" and "securities" co-occur).

A second approach (Feldman, Aumann, Amir, Klösgen, & Zilberstein, 1997; Feldman & Dagan, 1995; Feldman & Hirsh, 1997; Feldman, Rosenfeld, Stoppi, Liberzon, & Schler, 2000) is to use tags associated with the documents and to perform the data mining operations on the tags. However, to be effective this requires:

- Manual tagging, which is unfeasible for large collections; or
- Automated tagging using any one of the many automated categorization algorithms.

This approach suffers from two drawbacks. First, the number of distinct categories that such algorithms can effectively handle is relatively small, thus limiting the broadness of the mining process. More important, the process of automated categorization requires defining the categories a priori, thus defeating the purpose of discovery within the actual text.

A third approach is text mining via information extraction (Applet et al., 1993a, 1993b; Cowie & Lehnert, 1996; Lin, 1995; Riloff & Lehnert, 1994; Tipster, 1993), whereby we first perform information extraction on each document to find events, facts, and entities that are likely to have meaning in the given domain, and then to perform the data mining operations on the extracted information. A possible "event" may be that a company has entered a joint venture or has executed a management change. The extracted information provides much more concise and precise data for the mining process than in a word-based approach and tends to represent

more meaningful concepts and relationships in the document's domain. On the other hand, in contrast to the tagging approach, the information-extraction method allows for mining of the actual information present within the text rather than the limited set of tags associated to the documents. Using the information extraction process, the number of different relevant entities, events, and facts on which the data mining is performed is unbounded—typically thousands or even millions, far beyond the number of tags that any automated tagging system could handle.

Although on a basic level one can rely on generic proper name recognition that is mostly domain-independent, the power of this text mining approach is most apparent when coupled with extraction specific to the domain of interest. Thus, for example, when mining a collection of financial news articles, we would want to extract pertinent information on companies, industries, and technologies—information such as mergers, acquisitions, management positions, and so forth.

We start by describing the typical architecture of a text mining system. We then describe the various techniques for document preprocessing. Next we discuss document categorization and term extraction and move to information extraction methodologies. We then describe how to generate and utilize a taxonomy based on the output of the preprocessing phase. The automatically created taxonomy is the basis for all the analytic text mining operations. We continue by describing the analytic level of a typical text mining system including various types of visualizations. We then outline the primary hurdles when implementing a text mining system. A summary concludes this chapter.

ARCHITECTURE OF TEXT MINING SYSTEMS

A text mining system is composed of three major components:

1. Information feeders: A component to enable the connection between various textual collections and the tagging modules. This component connects to any Web site, streamed source (such a news feed), internal document collections, and any other types of textual collections.
2. Intelligent tagging: A component responsible for reading the text and distilling (tagging) the relevant information. This component can perform any type of tagging on the documents such as statistical tagging (categorization and term extraction), semantic tagging (information extraction), and structural tagging (extraction from the visual layout of documents).
3. Business intelligence suite: A component responsible for consolidating the information from disparate sources, allowing for simultaneous analysis of the entire information landscape.

In addition, the output of the tagging component can be fed into external systems such as corporate databases, workflow systems, or file systems. The tagging system is connected to the business intelligence suite or to the external systems either by producing rich XML documents or by using an API (application programming interface) that can connect directly to the internal data structures of the tagging process.

An example of such an XML document is shown in Fig. 21.1. We can see several examples of annotation of entities (like Dynegy Inc as an instance of a company, or Roger Hamilton as an instance of a person) and relationships such as the CreditRating relationship, which correlates a rating, a trend, the rating company, and the rated company; and the PersonPositionCompany relationship, which correlates a person a position and a company.

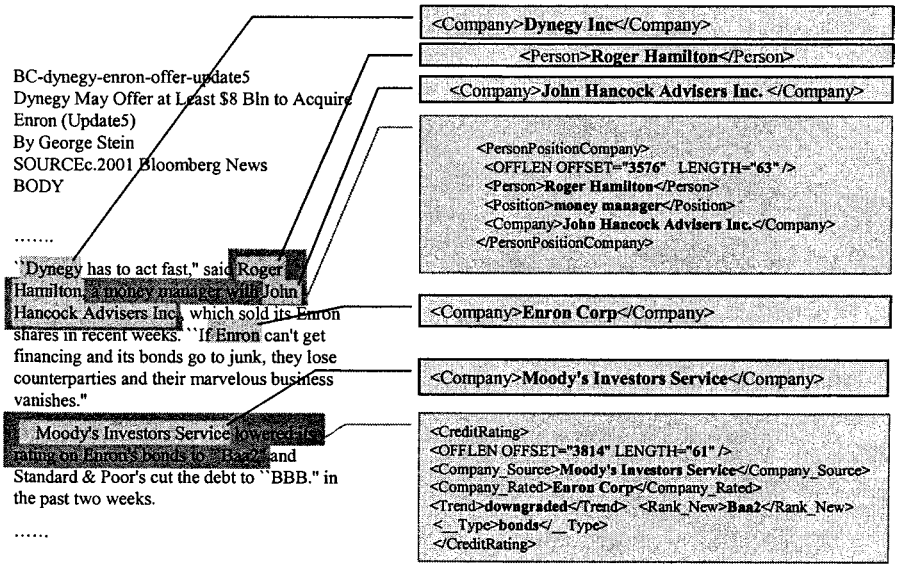


FIG. 21.1. XML annotation of an input document.

In Fig. 21.2 we can see the schematic diagram of a typical text mining system as was described. A more detailed description of the intelligent tagging component is shown in Fig. 21.3. Each of the taggers is using a separate training module that is based on annotated examples. A more detailed discussion of the training modules will be presented in the following sections. The training module for the structural tagging is producing document signatures that are then saved and mapped against new documents. The training module for the

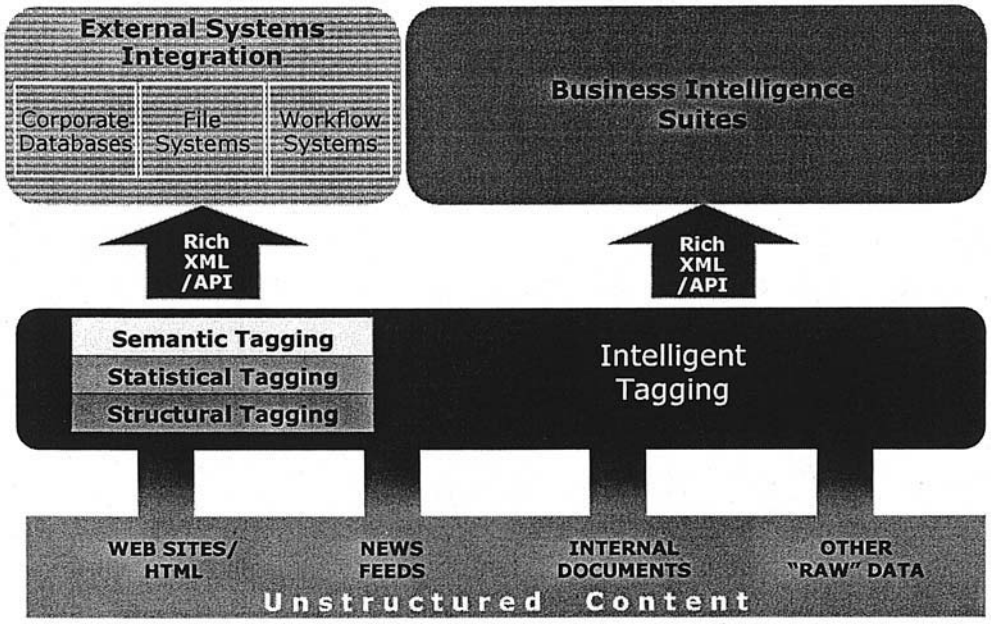


FIG. 21.2. Architecture of text mining systems.

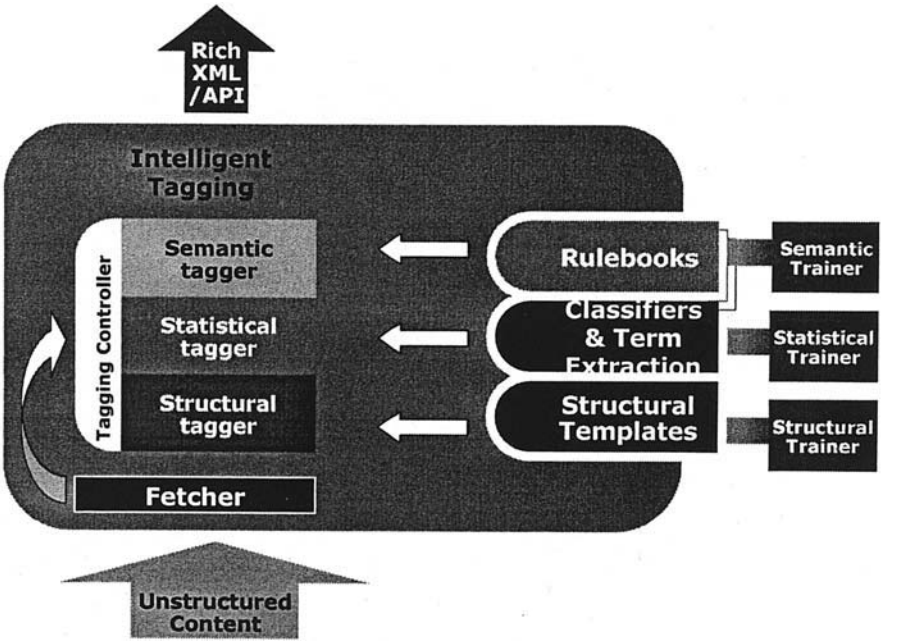


FIG. 21.3. Detailed description of the intelligent tagging component.

statistical tagging is producing classifiers for each of the categories, and the training module for the semantic training is producing information extraction rules based on annotated documents.

STATISTICAL TAGGING

Statistical tagging is based on the existence of a large collection of documents and usually relies on the presence of a training collection that is pretagged. The two main families of techniques within this approach are text categorization algorithms and term extraction algorithms. The next two subsections provide an overview of these families of algorithms.

Text Categorization

Text categorization (Cohen, 1992; Cohen & Singer, 1996; Dumais, Platt, Heckermann, & Sahami, 1998; Lewis, 1995; Lewis & Hayes, 1994; Lewis & Ringuette, 1994; Lewis, Schapire, Callan, and Papka, 1996; Sebastiani, 2002) is the activity of labeling natural language texts with thematic categories from a predefined set of categories. There are two main approaches to the categorization problem. The first is the knowledge engineering approach, in which the user is defining manually a set of rules encoding expert knowledge how to classify documents under given categories. The other approach is the machine learning approach (Sebastiani, 2002), in which a general inductive process automatically builds an automatic text classifier by learning from a set of preclassified documents.

An example of knowledge engineering approach is the CONSTRUE (Hayes, 1992; Hayes & Weinstein, 1990) system built by the Carnegie group for Reuters. A typical rule in the CONSTRUE system:

if DNF (disjunction of conjunctive clauses) formula then category
Example:

```
If   ((wheat & farm)       or
     (wheat & commodity)   or
     (bushels & export)    or
     (wheat & tonnes)      or
     (wheat & winter & ¬ soft))
     then Wheat
     else ¬ Wheat
```

The main drawback of this approach is the knowledge acquisition bottleneck. The rules must be manually defined by a knowledge engineer interviewing a domain expert. If the set of categories is modified, then these two professionals must intervene again. Hayes and Weinstein (1990) reported a 90% breakeven between precision and recall on a small subset of the Reuters test collection (723 documents). However, it took a tremendous effort to develop the system (several person-years), and the test set was not significant to validate the results. It is not clear that these superb results will scale up when a bigger system needs to be developed.

The machine-learning based approach is based on the existence of a training set of document that are pretagged using the predefined set of categories. A diagram of a typical ML based categorization system is shown in Fig. 21.4. There are two main methods for performing machine-learning based categorization. One method is to perform "hard" (fully automated) classification, in which for each pair of category and document we assign a truth value (either TRUE if the document belongs to the category or FALSE otherwise). The other approach is to perform a ranking (semiautomated) based classification. In this approach rather than returning a truth value the classifier return a *categorization status value* (*CSV*), that is, a number between 0 and 1 that represents the evidence for the fact that the document belongs to the category. Documents are then ranked according to their CSV value. Specific text categorization algorithms are discussed later.

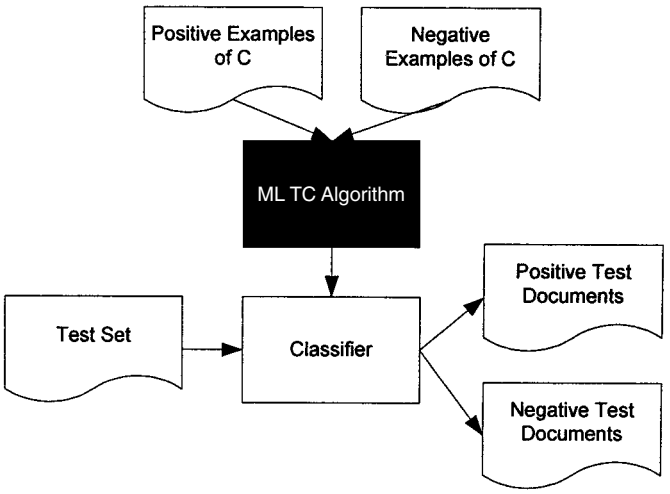


FIG. 21.4. Diagram of a typical machine-language based categorization system.

Definitions:

$D = \{d_1, d_2, \ldots, d_n\}$: the training document collection
$C = \{c_1, c_2, \ldots, c_k\}$: the set of possible categories to be assigned to the documents
$T = \{t_1, t_2, \ldots, t_m\}$: the set of terms appearing in the documents
w_{ij}: the weight of the jth term of the ith document
$\mathrm{CSV}_i(d_j)$: a number between 0 and 1 that represents the certainty that a category c_i should be assigned to document d_j
$Dis(D_i, D_j)$: the distance between document D_i and D_j; this number represents the similarity between the documents

Probabilistic Classifiers. Probabilistic classifiers view $CSV_i(d_j)$ in terms of $P(c_i \mid d_j)$, that is, the probability that a document represented by a vector $\vec{d}_j = \langle w_{1j}, \ldots, w_{mj} \rangle$ of (binary or weighted) terms belongs to c_i, and compute this probability by an application of Bayes' theorem

$$P(c_i \mid \vec{d}_j) = \frac{P(c_i)P(\vec{d}_j \mid c_i)}{P(\vec{d}_j)}.$$

To compute $P(d_j)$ and $P(d_j \mid C_i)$, we need to make the assumption that any two coordinates of the document vector, when viewed as random variables, are statistically independent of each other; this independence assumption is encoded by the equation

$$P(\vec{d}_j \mid c_i) = \prod_{k=1}^{|T|} p(w_{kj} \mid c_i).$$

Example-Based Classifiers. Example-based classifiers do not build an explicit, declarative representation of the category c_i, but rely on the category labels attached to the training documents similar to the test document. These methods have thus been called lazy learners, because they defer the decision on how to generalize beyond the training data until each new query instance is encountered. The most prominent example of example-based classifier is KNN (K nearest neighbor).

For deciding whether $d_j \in c_i$, KNN looks at whether the k training documents most similar to d_j also are in c_i; if the answer is positive for a large enough proportion of them, a positive decision is made, and a negative decision is taken otherwise. A distance-weighted version of KNN is a variation of KNN such that we weight the contribution of each neighbor by its similarity with the test document. Classifying d_j by means of KNN thus comes down to computing

$$CSV_i(d_j) = \sum_{d_z \in Tr_k(d_j)} Dis(d_j, d_z) \cdot C_i(d_z).$$

One interesting problem is how to pick the best value for k. Larkey and Croft (1996) use $k = 20$, whereas Yang and Chute (1994) and Yang and Liu (1999) found $30 \leq k \leq 45$ to yield the best effectiveness. Various experiments have shown that increasing the value of k does not significantly degrade the performance.

Propositional Rules Learners. There is a family of algorithms that try to learn the propositional definition of the category. One of the prominent examples of this family

of algorithms is Ripper (Cohen, 1992; Cohen & Singer, 1996). Ripper learns rules that are disjunctions of conjunctions. Here is an example of two rules that define the category "Ireland".

Ireland ← ira ∈ document, killed ∈ document.
Ireland ← ira ∈ document, belfast ∈ document.

Ripper builds a rule set by adding new rules till all positive exemplars are covered. Conditions are added to the rule until no negative exemplars are covered. Initially, examples were represented as feature vectors. Because the matrix was so sparse, each document was represented as a set. Ripper can also use negative features (i.e., w $\notin$ S).

One of the attractive features of Ripper is its ability to bias the performance toward higher precision or higher recall. This is done via the use of a special parameters call the Loss ratio. This is the ratio of the cost of a false negative to the cost of false positive. High loss ratio will increase recall and decrease precision.

Support Vector Machines. The support vector machine (SVM) algorithm was proven to be very fast and effective for text classification problems (Dumais et al., 1998; Joachims, 1998). SVMs were introduced by Vapnik in his work on structural risk minimization (Vapnik, 1979, 1995). A linear SVM is a hyperplane that separates with the maximum margin a set of positive examples from a set of negative examples. The margin is the distance from the hyperplane to the nearest example from the positive and negative sets. The diagram shown in Fig. 21.5 is an example of a two-dimensional problem that is linearly separable.

Clustering as a Preprocess Step for Categorization. Document clustering algorithms can be used in a preprocessing phase and enable finding the main themes in the documents without any need for corpus annotation. The fact that clustering is an instance of unsupervised learning makes it very suitable for the exploration phase of a text mining project. In addition, clustering algorithms were used in the link analysis phase to cluster together related entities. This approach proved to be very useful in identifying groups of related objects and in identifying their internal organization.

Comparison Between Text Categorization and Information Extraction. In contrast to the information extraction approach, in which the entities that tag the document are based on actual terms extracted from the document, text categorization tags

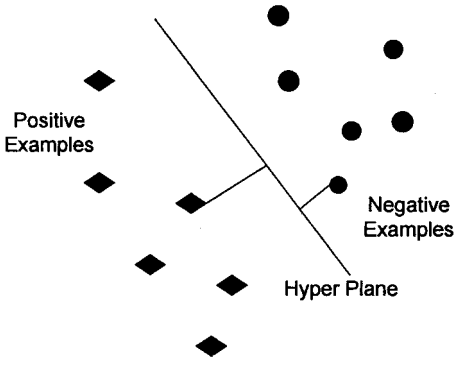


FIG. 21.5. Diagram of a two-dimensional linear SVM.

the document with concepts that do not need to be mentioned in the document itself. The main advantages of using a categorization approach are that it is less time consuming to prepare the training corpus, and there is no need to manually craft rules. On the other hand, one to five tags would be assigned to any given document. That would usually capture just some of the main topics of the document and certainly miss most of the important entities mentioned inside the document. In contrast, a document tagged by an information extraction system will have around 20 to 50 tags (for a two-to-three-page document). In a nutshell, information extraction was found to provide a much better infrastructure for text mining than was text categorization.

A Visual Interface to Document Classification. In Fig. 21.6 is a visual front end that enables the user to create classifiers for various categories and then test them on other document collections. In Fig. 21.6 we created 10 classifiers for categories such as "British banks," "CRM software," or "baseball." We then tested all classifiers on a test collection of 476, and focused on the performance of the baseball classifier on the test set. The system ranks all documents in a decreasing order certainty that baseball should be assigned to the document. The user can then set the threshold for the category to any of the scores of the ranked documents. This threshold will be used in the operation mode of the system for attaching the baseball tag to newly arriving documents. In this particular example the best threshold was 0.829, which provides a precision of 97% and recall of 98%.

Term Extraction

The term extraction module is responsible for labeling each document with a set of terms extracted from the document. An example of the output of the term extraction module is given in Fig. 21.7. The excerpt is taken from an article published by Reuters Financial on May 12, 1996. Terms in this excerpt that were identified and designated as interesting by the term extraction module are underlined.

The overall architecture of the term extraction module is illustrated in Fig. 21.8. There are three main stages in this module: linguistic preprocessing, term generation, and term filtering.

The documents are loaded into the system through a special reader. The reader uses a configuration file that informs it of the meaning of the different tags annotating the documents. In such a way, we are able to handle a large variety of formats. The Text Processing Language (TPL) reader packages the information into a Standardized General Markup Language file.

The next step is the linguistic preprocessing that includes tokenization, part-of-speech tagging and lemmatzations (i.e., a linguistically more founded version of stemming; see Hull, 1996). The objective of the part-of-speech tagging is to automatically associate morpho-syntactic categories such as *noun, verb, adjective,* and so forth, to the words in the document. Some systems use a rule based approach similar to the one presented in Brill (1995), which is known to yield satisfying results (96% accuracy) provided that a large lexicon (containing tags and lemmas) and some manually hand-tagged data is available for training.

SEMANTIC TAGGING

Rule based information extraction techniques rely on the fact that the information to be extracted from documents can be specified in such a way that a relatively small number of extraction rules are needed. The knowledge-intensive information extraction approach requires trained developers and is very laborious; however, some attractive features of this approach are the

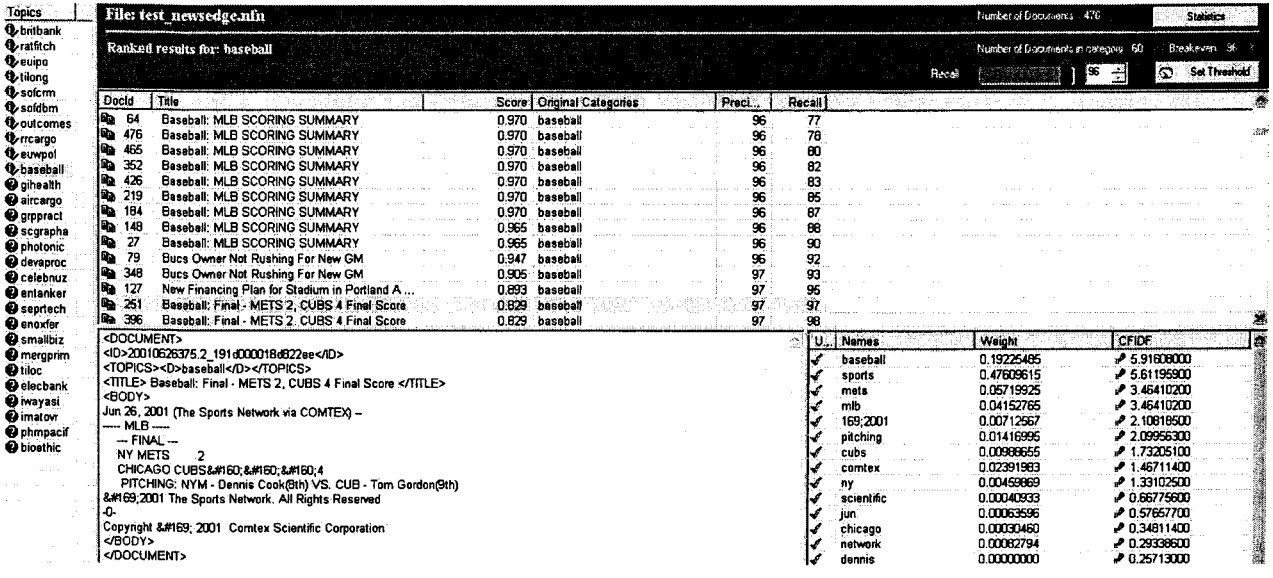


FIG. 21.6. A visual interface to classifying documents.

Profits at Canada's six big banks topped C$6 billion ($4.4 billion) in 1996, smashing last year's C$5.2 billion ($3.8 billion) record as Canadian Imperial Bank of Commerce and National Bank of Canada wrapped up the earnings season Thursday. The six banks each reported a double-digit jump in net income for a combined profit of C$6.26 billion ($4.6 billion) in fiscal 1996 ended Oct. 31.

But a third straight year of record profits came amid growing public anger over perceived high service charges and credit card rates, and tight lending policies.

Bank officials defended the group's performance, saying that millions of Canadians owned bank shares through mutual funds and pension plans.

FIG. 21.7. An example of the output of the term extraction process.

specificity of the extracted information and the precision and recall with which information can be extracted. The architecture of a typical information extraction system is shown in Fig. 21.9. We start our discussion of information extraction with a description of the Declarative Information Analysis Language (DIAL) information extraction language (Feldman et al., 2001, 2002).

DIAL

In this subsection we describe DIAL. DIAL is designed specifically for writing IE (information extraction) rules. The complete syntax of DIAL is beyond the scope of this chapter. Here we describe the basic elements of the language.

Basic Elements. The basic elements of the language are syntactic and semantic elements of the text and sequences and patterns thereof. The language can identify the following elements:

- Predefined strings such as "merger"
- Word class element: a phrase from a predefined set of phrases that share a common semantic meaning—for example, `WC-Countries`, a list of countries.

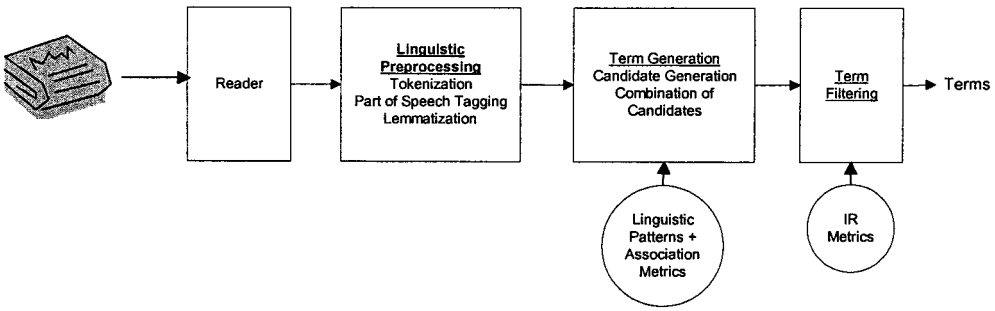


FIG. 21.8. Architecture of the term extraction module.

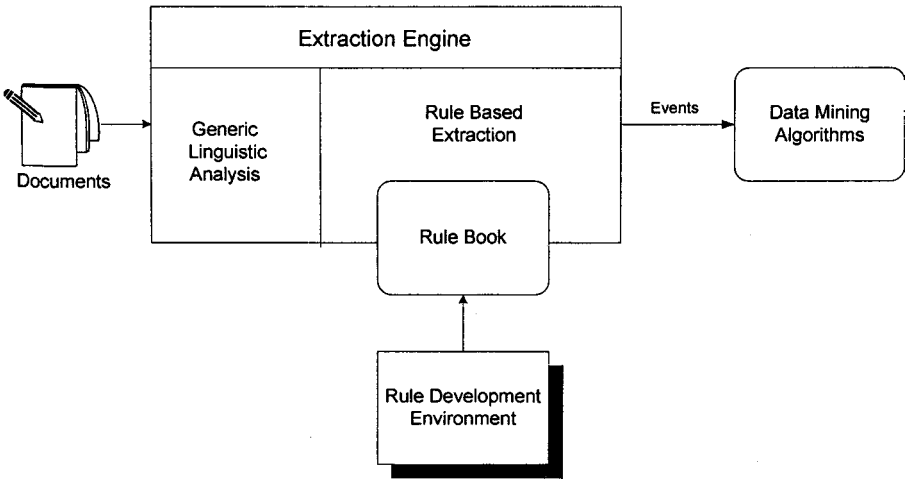


FIG. 21.9. Architecture of an information extraction system.

- Scanner feature (basic characteristic of a token), for example, `@Capital` or `@HtmlTag`
- Compound feature: a phrase comprising several basic features. Thus, `Match (@Capital & WCCountries)`, for example, will match a phrase that both belongs to the word class `WCCountries` and starts with a capital letter.
- Part-of-speech tag—for example, noun or adjective
- Recursive predicate call—for example, Company (C)

Constraints. Constraints carry out on-the-fly Boolean checks for specific attributes. These can be applied to fragments of the original text or to results obtained during processing extraction process.

The marker for a constraint is the word *verify*, followed by parentheses containing a specific function, which governs what it is checking for. For example:

```
verify ( StartNotInPredicate ( c , @PersonName ) )
```

ensures that no prefix of the string assigned to variable `c` is a match for the `predicate PersonName`.

IE Rule Bases. The rule base can be viewed as a logic program. Thus, a *rule base,* Γ, is a conjunction of definite clauses C_i: $H_i \leftarrow B_i$ where C_i is a clause tag, H_i(called the *head*) is a literal, and $B_i = \{B_{i1} B_{i2} \ldots .\} = P_i \cup N_i$ (called the *body*) is a set of literals, where $P_i = \{p_{ij}\}$ is a set of pattern matching elements and $N_i = \{n_{ij}\}$ is a set of constraints operating on P_i. The clause C_i: $H_i \leftarrow B_i$ represents the assertion that H_i is implied by the conjunction of the literals in P_i while satisfying all the constraints in N_i.

An example of a DIAL rule is the following, which is 1 of 10 rules to identify a merger between two companies:

```
FMergerCCM(C1, C2) :-
Company(Comp1) OptCompanyDetails "and" skip(Company(x),
SkipFail, 10) Company(Comp2) OptCompanyDetails
```

```
skip(WCMergerVerbs, SkipFailComp, 20) WCMergerVerbs
skip(WCMerger, SkipFail, 20) WCMerger
verify(WholeNotInPredicate(Comp1, @PersonName))
verify(WholeNotInPredicate(Comp2, @PersonName))
@% @!
{ C1 = Comp1; C2 = Comp2 };
```

The rule looks for a company name (carried out by the predicate `Company`, which returns the parameter `Comp1`) followed by an optional phrase describing the company, and then the word *and*. The system then skips up to 10 tokens (within the same sentence, and while not encountering any phrase prescribed by the predicate `SkipFail`) until it finds another company, followed by an optional company description clause. The system then skips up to 20 tokens until it finds a phrase of the word class `WCMergerVerbs`. (This may be something like "approved," "announced," etc.). Finally, the system skips up to 10 tokens scanning for a phrase of the word class `WCMerger`. In addition, the rule contains two constraints ensuring that the company names are not names of people.

Each rulebook can contain any number of rules that are used to extract knowledge from documents in a certain domain. Here are a few examples of rulebooks that were developed in DIAL:

- Financial rulebook: containing 11,500 rules, can identify more than 50 different entity types including company names; people names; organizations; universities; products; positions; locations (cities, countries, states, and addresses); dates, and amounts. In addition, it can identify more than 120 different event types such as: mergers (including a fine-grained distinction between known merger, new merger, rumored merger, planned merger, and cancelled merger); acquisitions (with a similar distinction between acquisition types); joint ventures; takeovers; business relationships; investment relationships; customer–supplier relationships; new product introductions; analyst recommendations for stocks and bonds, associations between companies and people; associations between companies and technologies; associations between companies and products; and many others.
- Business intelligence rulebook: contains 7,000 rules, can identify thirty different entity types, including company names, people, positions, prices, and products. In addition, it can identify more than eighty events, including joint ventures; business relationships; mergers and acquisitions; customer–supplier relationships; rival relationship; layoffs; strategic reorganizations; investments; new management; IPO plans; senior appointments; product-related events (product features); awards; and so forth.
- Intellectual property rulebook: contains 100 rules and can identify 30 different types of entities in patent files, including inventor, assignee, examiner, and a set of noun-phrase classes based on the context in which they appear.
- A protein relationship rulebook: enabling the extraction of relationships between protein pathways from articles featured in Medline. This rulebook contains 500 rules and can identify 30 different types of entities, including proteins and 10 different relationships including phosphorylation.

Development of IE Rules

Developing an IE module for a new domain can be very tedious and time-consuming. To speed up the process, the developer needs an environment with a range of productivity tools. The focus of this section is on these editing and debugging tools. In addition, the environment should

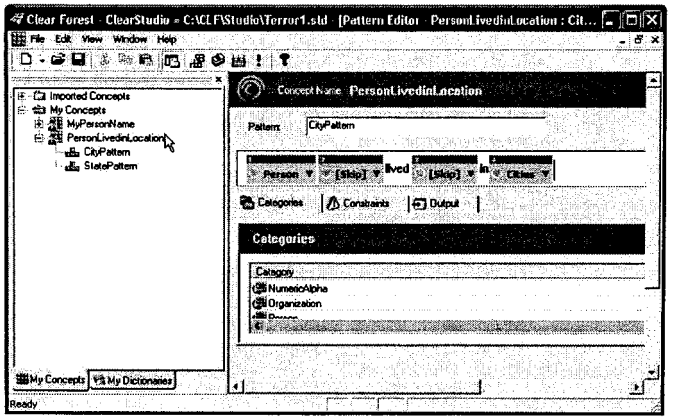


FIG. 21.10. Defining patterns by using visual pattern editors.

provide tools for checking the quality of the rulebook by examining its output on documents streams.

Visual Rule Authoring. To increase the productivity of the application developers, it is prudent to consider using rule editors and visual tools. The template editor enables the user to select a text fragment from any document and build a new pattern based on that. The user can generalize any part of the text fragments by selecting the part and replacing it with one of the predefined building blocks. The building blocks can be either primitives such as company name, person name, product, location and so forth, or a reference to predefined lexicons. Most of the pattern matching elements discussed previously can be defined using the visual pattern editor.

As an example, in Fig. 21.10, the pattern that was extracted from the sentence "Kamfar, also known as Amer Taiybkamfar, was reported to have lived in Florida for the last 18 months" is shown. The pattern generalizes the sentence to extract a relationship that correlates a person with a city or state. It was found that the users are much more comfortable working at this abstraction level than using a formal language to define the grammar of the patterns.

Debugging Tools. The DIAL environment includes a variety of tools for monitoring the integrity and performance of the rule base during its development. The tools available reflect the many types of problems that may arise, ranging from simple syntax errors that prevent the code's compilation (e.g., omitting a vital punctuation mark or misspelling the name of a predicate or word class) to inefficiencies in the rules themselves that lead to inaccuracies in the results (e.g., *Bank of England* as a company), or events that are missed altogether. The user can then make modifications to the rule and rerun the code to ensure that the problem has been fixed or the accuracy improved.

Central to all these operations is the interpreter, which can act on the code line-by-line without precompilation. This is used to check the code for syntactical integrity before it is used for any information extraction. The offending line is highlighted, usually with an accompanying comment in the output pane, allowing the user to zoom in on the problem (see Fig. 21.11).

Once the code has passed the compilation test, it is tried out on a number of sample texts. The following debugging tools are available:

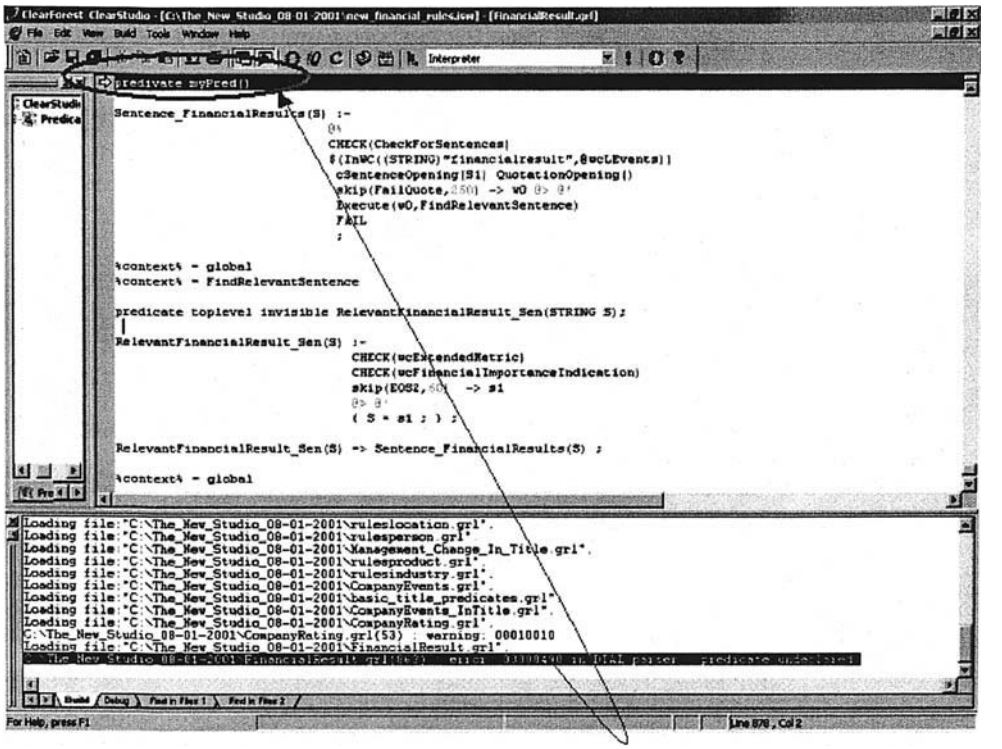


FIG. 21.11. ClearLab's built-in interpreter pinpoints syntax errors that prevent compilation.

• Reviewing event tables for rapid spotting of erroneous events. These can then be double-clicked to highlight the text fragment in the source text that caused the problem. This is usually enough to alert the user to the nature of the special case that caused the erroneous output and to make adjustments to the rule to prevent such occurrences in future.

• Right-clicking events. This allows the user to go directly to the relevant predicate behind the event—and specifically to the culprit definition in the code—and amend it as necessary.

• Tracer. This is used to monitor the incidence of *recall errors*—events that should have been caught but were not. A relevant rule is applied to the specific text in question. The success or failure of each component of the rule is then clearly shown in a report, featuring green check marks (success), red crosses (failure), and blue question marks (unchecked section), as in Fig. 21.12. Appropriate action can then be taken as necessary to improve the relevant rule(s).

• Event diff. This utility that allows you to assess the comparative effectiveness of incremental changes to the rules by comparing the list of events extracted using the new and the old versions. Typically, this is done soon or immediately after changing or adding any number of predicates.

• Profiler. This tool analyzes the rule file's performance, how long each predicate took to process a given document collection, and which in particular need tweaking, revising, or even complete removal to streamline the IE process. Its report is created as part of the compilation process, and thus the tool typically is applied at the end of the development process.

• Low-level debugging tool. It is similar to the tracer in that it tests the code against a sample text of the user's choosing (Fig. 21.13B). But it is more comprehensive because it examines the processing by the entire rule file up to a breakpoint of their choosing (A). All aspects of

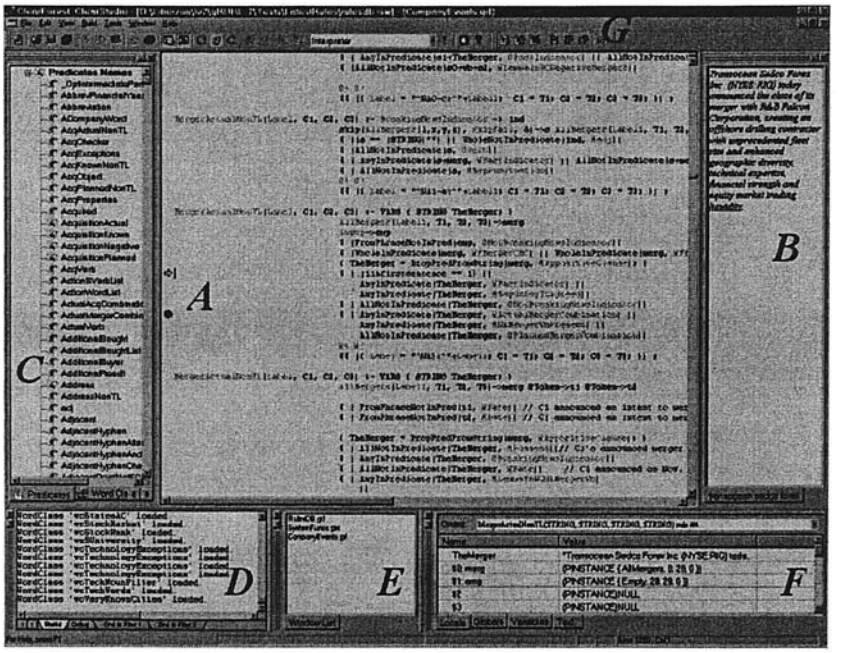


FIG. 21.12. The low-level debugger tool built into the environment allows the whole or part of the rulebook (up to a user-defined breakpoint, A) to run on any sample text (B), and all aspects of the results to be examined (D, E, F).

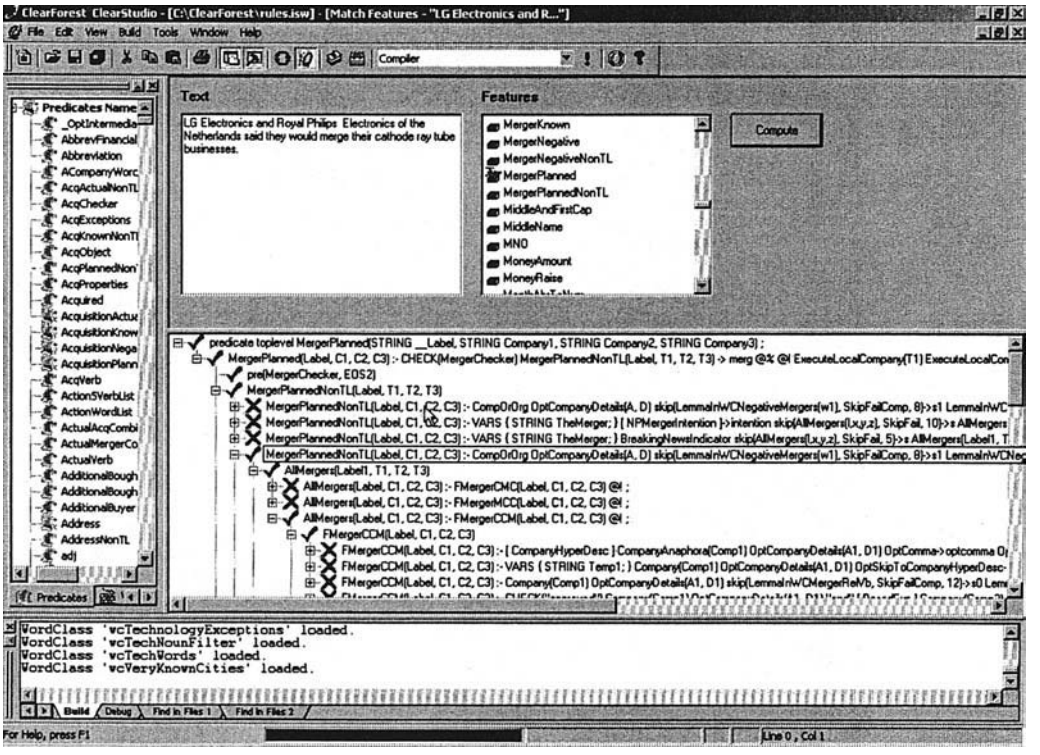


FIG. 21.13. The tracer tool allows specific predicates to be tested for effectiveness on sample texts that are pasted or typed into the relevant section.

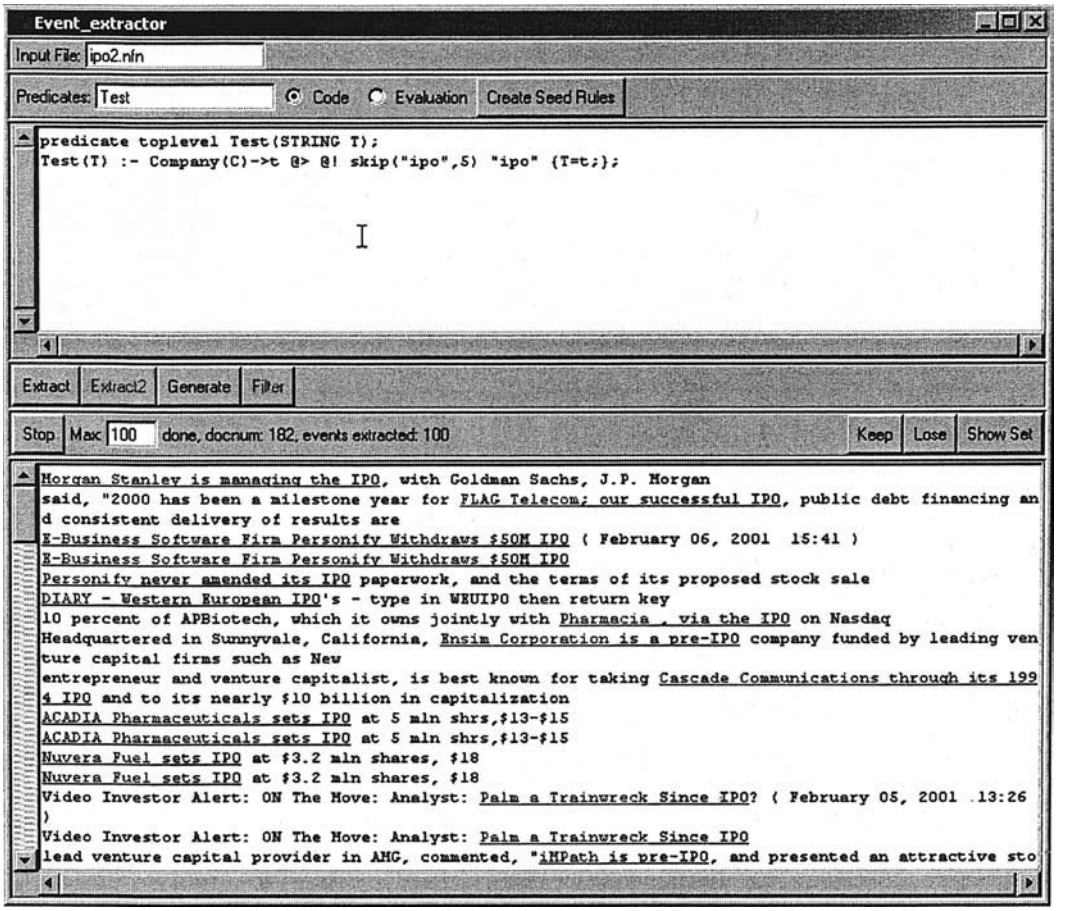


FIG. 21.14. The pattern locator tool.

the process may be examined subsequently, from the full list of predicates and functions in the rule file (C), to the word classes and other resources actually loaded, the active windows, and variables (E, F). *Stepinto, Stepover,* and *Stepout* tools (G) allow one to examine each rule call-by-call individually within the same stack frame or outside it.

- Pattern locator tool. This enables the user to enter a DIAL pattern and locates all instances of the pattern in a document collection. It then enables generalizing a selected subset of those instances. Portions assigned to variables in the pattern are shown in red and blue. The full instance of the patterns is underlined (see Fig. 21.14).

Contrasting Rule Engineering with Automatic Rule Learning. In contrast to the rule based approach, in which most of the effort is focused on writing rules, other systems use a machine learning approach in which IE rules are automatically learned based on an annotated corpus (Lehnert et al., 1991; Riloff & Lehnert, 1994; Soderland, Fisher, Aseltine, & Lehnert, 1995). Most of the effort in this case would then be devoted to annotating the corpus with entities and relationships. Although experience has shown that the accuracy of entity extraction rules was quite on par with the handcrafted rules, this was not the case for automatically learned relations extraction rules. From our experience, we managed to get by

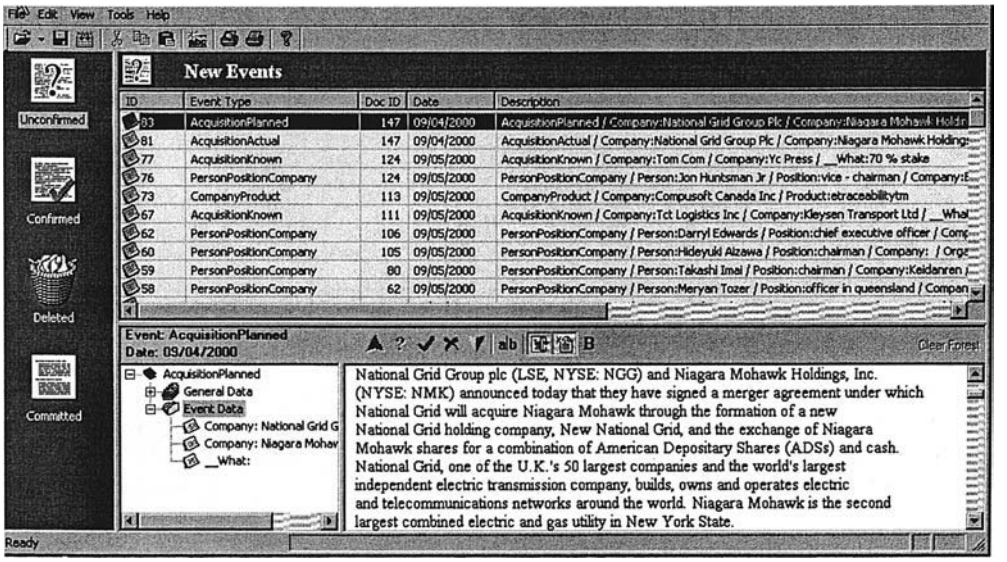


FIG. 21.15. The main auditing display.

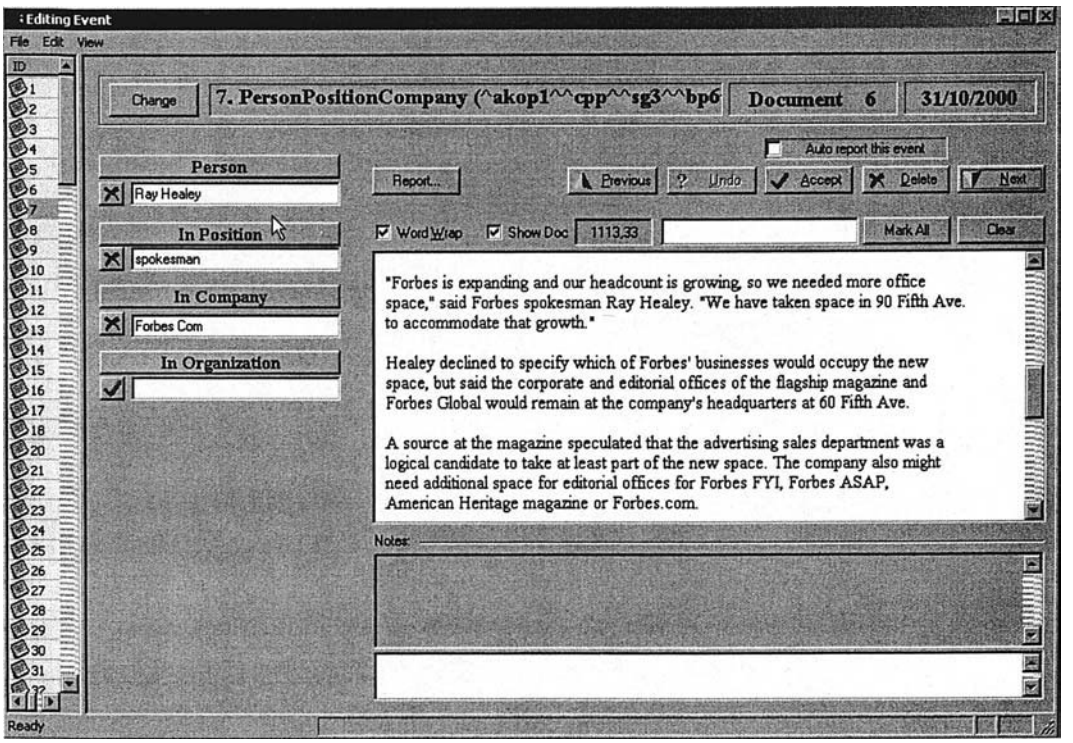


FIG. 21.16. Auditing a Person Position Company event.

using handcrafted rules producing a breakeven score of around 92% (for event and fact extraction). Machine learning based approaches were able to achieve only around 70% breakeven.

Auditing Environment

The auditing environment enables the user to view all events extracted from the document collection and easily fix erroneous events. The errors may be of several types: The event is completely wrong, and in that case we will delete it altogether; the event is correct, but some of the fields are incorrect, in which case we fix the erroneous fields; or the event is accurate (and all fields are accurate as well). In this case we may update the taxonomy or the thesaurus with new entities that may have been discovered in this event.

In Fig. 21.15 we can see the main screen of the auditing environment. The top pane shows all the events that were found in the collection. The lower pane shows the text fragment from which the event was extracted along with the fields that comprise the event.

In Fig. 21.16 we can see how to audit a specific PersonPositionCompany event. The tool allows the user to see the text fragment from which the event was extracted in context. In addition, it enables the user to change any of the individual fields that comprise the event. If the field value is already in the taxonomy (under the right category) then a check mark appears next to the field, otherwise, an X will appear. When the user clicks on any of the fields, he or she gets a screen that makes it possible to add the entry to the taxonomy and thesarus or to change the value for the given field. Such a screen is shown in Fig. 21.17. The user gets the value of "president and CEO" for the position field for the event PersonPositionCompany. The system shows similar values from the same category (president, in this case), and the user has the option to add this as a new value of position in the taxonomy or to select another value.

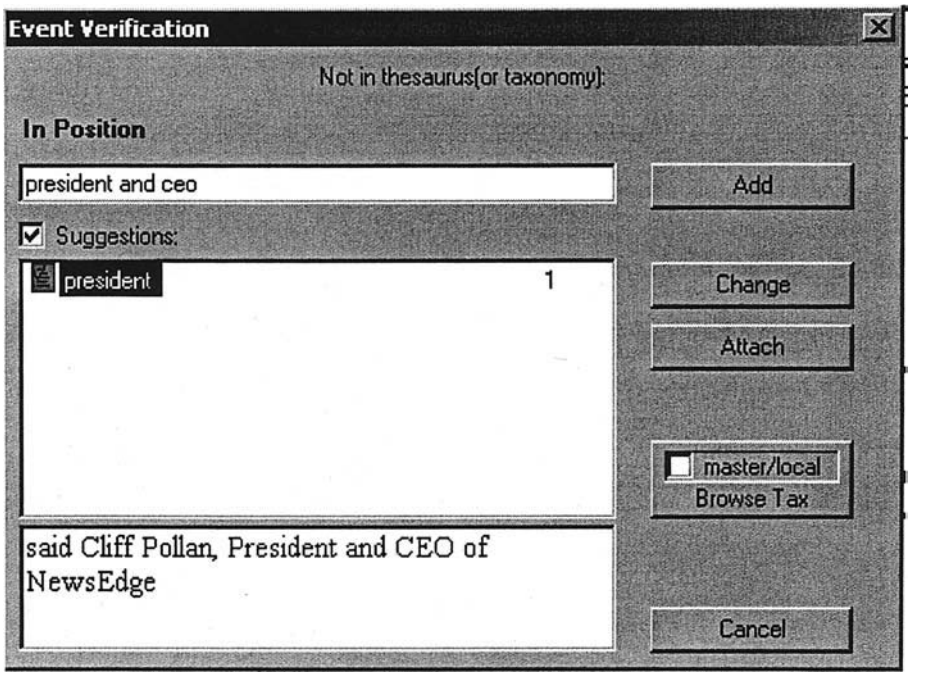


FIG. 21.17. Updating the taxonomy and the thesaurus with new entities.

STRUCTURAL TAGGING

Most text-processing systems simplify the structure of the documents they process. The visual form and layout of the documents is ignored and only the text itself is processed, usually as linear sequences or even bags of words. This allows the algorithms to be simpler and cleaner at the cost of possible loss of valuable information. In this section we show an approach that ignores the content of words while focusing on their superficial features, such as size and position on the page. Such an approach is not a rival but a complement to the conventional text extraction systems and can also function as a preprocessor or a converter.

We implemented this approach in a system called PES (PDF extraction system). PES accepts its input in the form of Acrobat PDF documents. A document page in PDF format is represented by a collection of primitive objects, which can be characters, simple graphic shapes, or embedded objects. Each primitive has properties, such as font size and type for characters, and position on the page, given as coordinates of the object's bounding rectangle. We are interested in an automatic process that accepts a formatted document as input and returns a predefined set of elements of the document, each assigned to a corresponding field, for example, "AUTHOR = . . . , TITLE = ..." The set of field names and which parts get assigned to them is problem-dependent and may be different for different types of documents. Thus, we seek a system that learns how to extract the proper document elements based on examples provided by a domain expert. In PES, described in this chapter, a domain expert annotates a set of documents, marking the fields to be extracted. Each annotated document functions as a template, against which new documents can be matched.

At the heart of the extraction system we have the following problem:

Given:

1. Document A (a template)
2. Set of primitives in A (annotated fields), denoted P_A
3. Document B (a query document)

Find:

1. The degree of similarity between documents A and B
2. The set of primitives in B that corresponds to P_A

The first step in the process of finding the primitives of B that correspond to P_A, is to find similarities between the original document A and the new document B. The simplest way to match two documents is coordinate-wise: Return as answer all objects of the query document that fit into the bounding rectangle of the marked subset in the template. The disadvantages of such an approach are obvious. If the same field has different visual sizes, for example, a document title containing different number of words and text lines, or if the field is shifted a bit, the system will not identify the correct match. Nevertheless, the coordinates form a good basis for more refined heuristics, as the same fields tend to reside in more or less the same place across documents. However, the correspondence must be established between objects, not coordinates. In addition, the correspondence must be between higher-level groups and not only between primitive objects.

The PDF document representation does not contain any information about text lines, paragraphs, columns, tables, and other meaningful groups of primitives. The format is designed

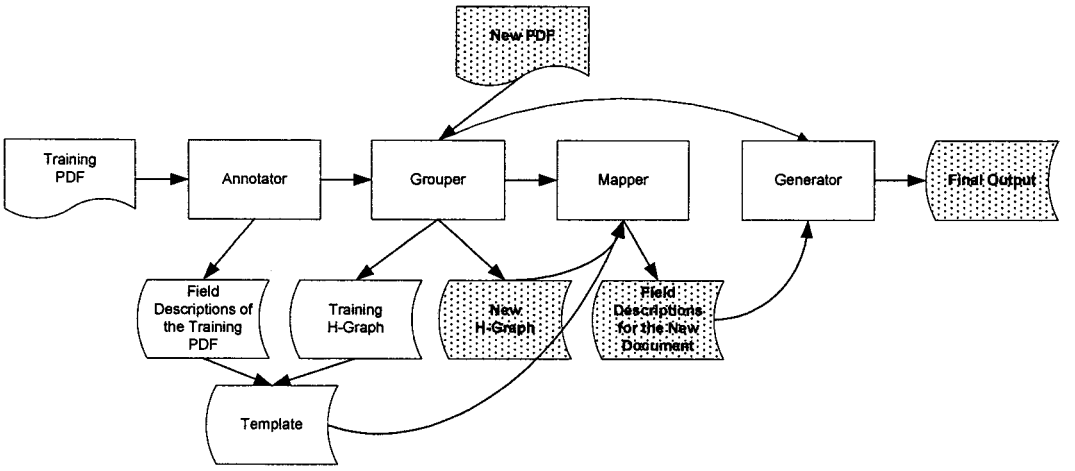


FIG. 21.18. Architecture of the PES.

for human reading, where the human mind does the necessary grouping unconsciously. For an information extraction system to take advantage of the visual clues available in the PDF format, the system must perform perceptual grouping as the first stage of processing a document. The approach we take is to take the physical/visual representation of the document and transform it into a complex abstract representation consisting of nested objects and relationships between them. We call this step *perceptual grouping*. Once perceptual grouping has been performed, the resulting structure is independent of the specific document type. This approach allows us to provide a general procedure applicable to diverse formats and rapidly adaptable to new formats.

Once the document structures are generated, these structures can be used to extract information. A representative set of documents is annotated by a domain expert, with parts of the documents' structures being assigned to certain fields. These documents serve as templates to be matched against the new documents. In the process of structural mapping, a correspondence is created between two document structures, mapping the objects in a template document to the objects in the nonannotated query document.

PES contains several components: annotator, grouper, mapper, and extractor. Annotator is a GUI tool that allows the user to mark fields in a PDF document and store their names and positions in a separate file. Grouper takes a PDF document as input, does the grouping, and saves the document structure. Mapper's input is a template (document structure + fields data) and a document structure for a query document. The template is mapped onto the query document, and the elements assigned to the various fields are produced as output, together with the overall quality of the mapping. Extractor takes a document structure and the selected elements and outputs the elements' text. The architecture of PES is shown in Fig. 21.18.

TAXONOMY CONSTRUCTION

One of the crucial issues in performing text mining is the need for term taxonomy. A term taxonomy also enables the production of high-level association rules, which are similar to general association rules (Srikant & Agrawal, 1995). These rules capture relationships between

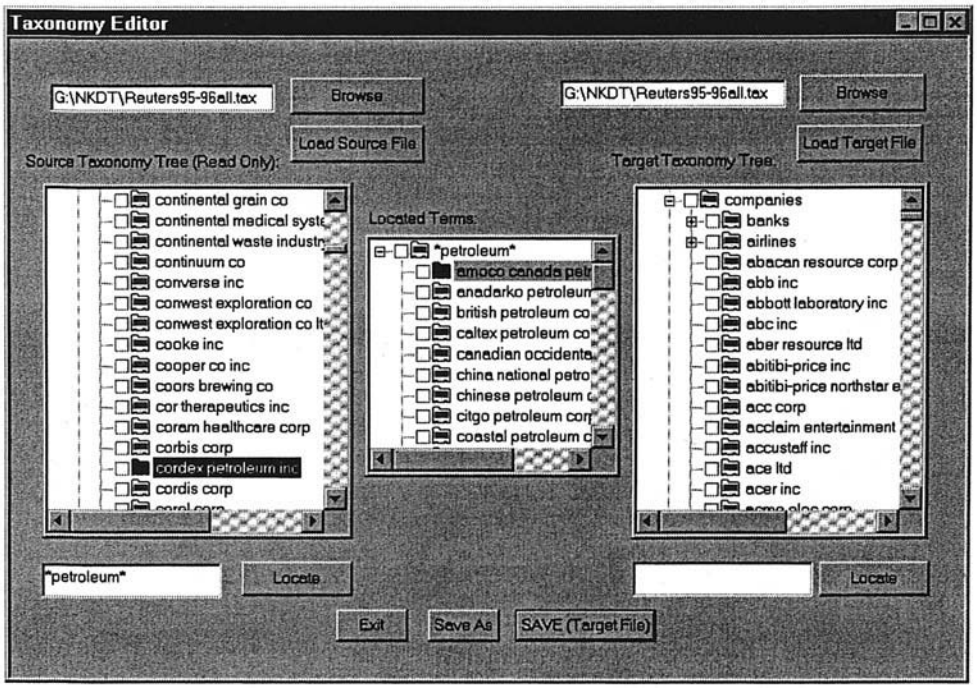


FIG. 21.19. Taxonomy editor.

groups of terms rather than between individual terms. A taxonomy is also important in other text mining algorithms such as maximal association rules and frequent maximal sets (Feldman et al., 1997).

A taxonomy also enables the user to specify mining tasks in a concise way. For instance, when trying to generate association rules, rather then looking for all possible rules the user can specify interest only in the relationships of companies in the context of business alliances. To do so, we need two nodes in the term taxonomy marked "business alliances" and "companies." The first node contains all terms related to alliance such as "joint venture," "strategic alliance," "combined initiative" and so forth, whereas the second node is the parent of all company names in our system.

Building term taxonomy is a time-consuming task. Hence, there is a need to provide a set of tools for semiautomatic construction of such taxonomy. One such tool is the taxonomy editor shown in Fig. 21.19. This tool enables the user to read a set of terms or an external taxonomy and use them to update the system's term taxonomy. The user can drag entire subtrees in the taxonomies or specify a set of terms via regular expressions. In Fig. 21.19 we can see the terms found when specifying the pattern *petroleum*. The initial set of terms is the set of all terms extracted from a collection of 64,000 Reuters documents from 1995–1996 (shown in the left tree), the terms matching the query are shown in the middle tree, and the right tree is the target taxonomy. The user can utilize the entries in the middle tree to create a new node in the target taxonomy (such as "petroleum companies").

The taxonomy editor also includes a semiautomatic tool for taxonomy editing called the taxonomy editor refiner (TER). TER compares generated frequent sets against the term taxonomy. When most of the terms of a frequent set are determined to be siblings in the taxonomy

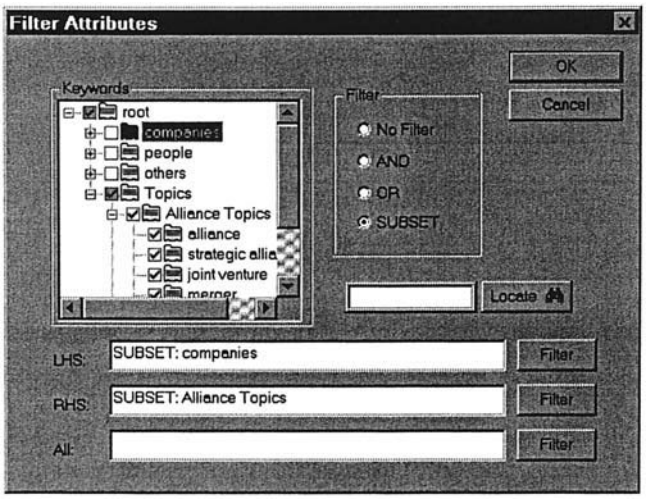


FIG. 21.20. Specifying a filter to display only association rules with "Companies" to the left of the rule and "Alliance Topics" to the right.

hierarchy, the tool suggests adding the remaining terms as siblings as well. For example, if our taxonomy currently contains 15 companies under "tobacco companies" and the system generates a frequent set containing many tobacco companies, one of which does not appear in the taxonomy, the TER will suggest adding this additional company to the taxonomy as a tobacco company. The TER also has a term clustering module, again suggesting that terms clustered together be placed as siblings in the taxonomy.

In the example presented in Fig. 21.20 the user is interested in business alliances between companies. The user therefore specifies a filter for the association rules generation algorithm, requesting only association rules with companies on the left hand side of the rule and business alliance topics on the right hand side. Fig. 21.20 shows the filter definition window.

Using the Reuters document corpus described previously, the system generated 12,000 frequent sets that comply with the restriction specified by the filter (with a support threshold of five documents and confidence threshold of 0.1). These frequent sets generated 575 associations. A further analysis removed rules that were subsumed by other rules, resulting in a total of 569 rules. A sample of these rules is presented in Fig. 21.21. The numbers presented at the end of each rule are the rule's support and confidence.

The example in Fig. 21.21 illustrates the advantages of performing text mining at the term level. Terms such as "joint venture" would be totally lost at the word level. Company names, such as "santa fe pacific corp" and "bank of boston corp," would not have been identified either. Another important issue is the construction of a useful taxonomy such as the one used in Fig. 21.21. Such a taxonomy cannot be defined at the word level because many logical objects and concepts are, in fact, multiword terms.

In addition to analysis tasks, we have in the system a set of tools for exploring the document collections based on the created taxonomy. In Fig. 21.22 we can see such an interactive distribution-browsing tool. We started by computing the distribution of all alliance-related topics. The most frequent topic was "join-venture," for which we then computed the company distribution. IBM was the company that cooccurred the most with "joint-venture." We then chose to compute the company distribution of MCI (in the context of "joint venture"); Sprint

america online inc, bertelsmann ag ⇒ joint venture 13/0.72

apple computer inc, sun microsystems inc ⇒ merger talk 22/0.27

apple computer inc, taligent inc ⇒ joint venture 6/0.75

sprint corp, tele-communications inc ⇒ alliance 8/0.25

burlington northern inc, santa fe pacific corp ⇒ merger 9/0.23

lockheed corp, martin marietta corp ⇒ merger 14/0.4

chevron corp, mobil corp ⇒ joint venture 11/0.26

intuit inc, novell inc ⇒ merger 8/0.47

bank of boston corp, corestates financial corp ⇒ merger talk 7/0.69

FIG. 21.21. A sample of the association rules that comply with the constraints specified in the rule filter shown in Fig. 21.20.

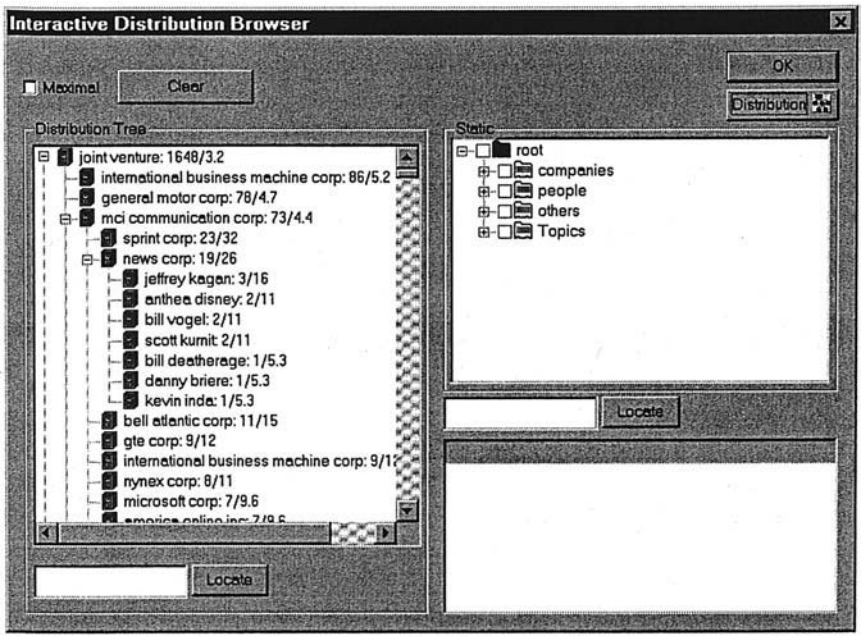


FIG. 21.22. Interactive exploration of term distributions.

was the company with the highest frequency. We then chose to compute the people distribution of News Corp. in the context of "joint venture" and "MCI Communication Corp."

IMPLEMENTATION ISSUES OF TEXT MINING

In this section we outline several of the practical problems that one would typically face when trying to develop a text mining system. We describe the problems and review some of the common solutions to these problems.

Soft Matching

Soft matching is the problem of matching synonyms that refer to the same entity. As is often the case, different authors may use different phrases when referring to the same entity. Sometimes, even the same author may refer to the entity in different ways. In addition, in some cases the multiple names of the same entity are due to errors and variation in spelling. In this section we refer just to proper name reference to entities; the problem of pronouns will be discussed later in the anaphora resolution section.

Examples. We list here several common scenarios that cause the text mining system to have multiple names for the same entity.

1. Punctuation variations: WalMart versus Wal-mart or Wal Mart, Microsoft Corp. versus. Microsoft Corp
2. Spelling mistakes: Microsoft versus Micorsoft
3. Use of abbreviations: GM for General Motors, IBM for International Business Machines
4. Formal name versus Informal name: Microsoft versus Microsoft Corp. or Microsoft Corporation
5. "Nicknames": Big Blue for IBM

This problem of soft matching (Tejada, Knoblock, & Minton, 2001) is applicable to many types of proper names such as those of companies, organizations, countries, cities (Big Apple for NYC), people, product names, and so forth.

Solutions. The most widely used technique for correcting spelling mistakes is using a soundex algorithm. This algorithm can match words that have a similar phonetic pronunciation. The problems of abbreviations and nicknames can be solved by using a lookup table that will contain for each entity all known abbreviations and nicknames. The problems of punctuation variations and variation in the formality of the entity names can be solved by coding name conversion rules. An example of such a conversion rule is X Corporation or X Corp. are mapped to X. Note that the application of such broad rules across documents can be dangerous at times and link together names that do not refer to the same entity (for example, there might be Highland Partners, Highland Corporation, and Highland Inc., each referring to a different entity). In many systems a more conservative approach is used and only names that appear within the same documents are candidates for becoming synonyms.

Temporal Resolution

When extracting events and facts from the documents we want to associate them with the date and time in which they occurred. This time-stamp will be used for temporal analysis of the document collection (see the later section about Trend Graphs).

There are two main families of date and time formats: absolute format and relative format. Examples of absolute date formats are 10/5/2002, or January 5, 2001. Examples of relative date formats are 3 days ago, yesterday, last month, a year ago, and so forth.

To "time-stamp" any event mentioned inside a document we need to perform the following two-step procedure:

1. Determine the absolute data of the document: This can be done either by analyzing the document and extracting the date of the document, or if no date can be extracted directly from within the document, use the date the document was created.
2. Identify the relative phrase that describes when the event took place and, based on the absolute date of the document, compute the absolute date of the event.

Some of the challenges in identifying the absolute date of the document are related to the large variety of possible date formats (American date, European date, short notation, long notation, abbreviations, etc.). Most of those problems can be solved by coding date conversion rules that convert all dates to one canonic form. Some temporal phrases that are fuzzy in nature pose an even more difficult problem because it is hard to resolve them into a specific date. Examples are "at a later date," "in the very near future," and others. The solution in this case is to utilize fuzzy logic, and rather than providing a sharp date, to provide a fuzzy set that represents a fuzzy date.

Anaphora Resolution

One of the main challenges in developing comprehensive text mining systems is anaphora resolution, or the ability to resolve coreferences (Frantzi, 1997; Hobbs 1986; Hobbs, Stickel, Appelt, & Martin, 1993; Ingria & Stallard, 1989; Lappin & McCord, 1990). Consider, for example, the following text fragment from the Chicago Tribune:

> Mohamed Atta, a suspected leader of the hijackers, had spent time in Belle Glade, Fla., where a crop-dusting business is located. Atta and other Middle Eastern men came to South Florida Crop Care nearly every weekend for two months.
>
> Will Lee, the firm's general manager, said the men asked repeated questions about the crop-dusting business. He said the questions seemed "odd," but he didn't find the men suspicious until after the Sept. 11 attack.

It is fairly easy to conclude that "Atta" refers to Mohammad Atta; it is a little more difficult to conclude that "he" refers to Will Lee. However, it is much more difficult to infer that "men" refers to Mohammad Atta and his friends, because this coreference appears in a different paragraph and does not include any direct reference to Atta. In general, it was found that resolving proper names and aliases (such as GM for General Motors) was fairly easy; resolving pronominals such as "he," "she," and "we" was harder; and resolving definite noun phrases such as "the ruthless man" was the most difficult and the most error prone. The approach taken is a knowledge based approach in which for each referring phrase all accessible antecedents are collected. The accessible antecedents are computed based on the type of the referring phrase.

For proper names all previous entities serve as candidates. For pronouns, entities that appear within the previous sentences of the current paragraph are used. For definite noun phrases all entities that appear within the current paragraph and the preceding paragraph are used. One exception to this heuristic is that for entities of the form "the X" (where X was one of company, organization, corporation, etc.) the scope was extended to the whole preceding text. To select the right antecedent from the set of all possible candidates, the candidates that are incompatible with the referring phrase are eliminated (either due to gender, type, or plurality). From the filtered set the final candidate is selected according to the following heuristics (in order of importance):

1. Prefer the candidate that appears earlier in the current sentence
2. Prefer the candidate that appears earlier in the previous sentence
3. Prefer the candidate that appears later within other sentences (prioritized in descending order of their position in the document)

To compute the effectiveness of these anaphora resolution heuristics for the example from the Chicago Tribune, the number of pairs (of referring expression and antecedent) that correctly matched was computed. It was found that in 82% of cases the right match for the referring expression was performed.

To Parse or Not to Parse?

Based on actual empirical evaluation, it was found that it is enough to focus just on the core constituents and use shallow parsing augmented by "smart skips." These skips enable the information extraction engine to skip irrelevant parts and focus just on the important phrases of each sentence. Other researchers have attempted to use full parsing as a component in their information systems and have concluded that it was not worthwhile to invest the extra effort. Specifically, full parsing was included in the SRI TACITUS system (implemented for Message Understanding Conference [MUC]-3) (Hobbs et al., 1992, 1993) and the NYU PROTEUS system (implemented for MUC-6) (Grishman, 1996). Neither of these systems gained any improvement in accuracy due to the full parsing employed. The main problem with using full parsing is that, due to the combinatorial explosion of possible parses, it is very slow and very error prone.

Database Connectivity

Without database connectivity a text mining application is isolated. The application developer must consider database interfaces both for bringing data into the text mining application and for outputting and storing the extracted information.

Input. During the development of many text mining applications, it was noticed that often the analyst would like to use background information about entities, which is not available in the documents. This background information enables the analyst to perform filtering and clustering and to automatically color entities in various colors. For instance, when dealing with relationships between companies, an analyst would often like to use the Standard Industrial Code (SIC) assigned to companies. The SIC enables the analyst to focus, for instance, on relationships between software companies and hardware companies. The relationships are extracted from the documents, but often the actual SIC is not mentioned in the documents. To enable the usage of such background information, a direct connection to an external database

is needed. A database gateway that can connect to external relational databases was developed. This gateway can create virtual nodes in the taxonomy based on properties stored in the database. As an example, all the information from Hoovers was stored in a database and a utility enabled defining groups of companies on the fly based on their various properties. Such a group might be all companies with headquarters in New York City, the industry code "Financial Services—Investment Firms," and more than 1,000 employees. This utility allows one to create flexible taxonomies that are based on live data residing in relational databases and apply them to entities extracted from any stream of documents.

Output. The ability to output the extracted features from mining text into a database is important for practical text mining applications. Text mining applications typically need to perform various kinds of postmining analysis on the features extracted. Database output of features significantly aids this process.

API. The text mining application typically would mine volumes of text (for example, tens of thousands of news articles per day), and it is essential to store the text mining results in a relational database in a seamless manner. The text mining application must provide an API to perform this function.

VISUALIZATIONS AND ANALYTICS FOR TEXT MINING

When developing a text mining system, one of the crucial needs is the ability to browse through the document collection and be able to "visualize" the various elements within the collection. This type of interactive exploration enables one to identify new types of entities and relationships that can be extracted and better explore the results of the information extraction phase.

We provide an example by using a visualization tool called ClearResearch (Aumann et al., 1999; Feldman et al., 2001, 2002). This visualization tool enables the user to visualize relationships between entities that were extracted from the documents. The system enables the user to view collocations between entities or a semantic map that will show entities that are related by any of a user-definable set of relationships.

We first give some basic definitions and notations.

Definitions and Notations

Let T be a taxonomy. T is represented as a DAG (directed acyclic graph), with the terms at the leaves. For a given node $v \in T$, we denote by *Terms* (v) the terms that are decedents of v.

Let D be a collection of documents. For terms e_1 and e_2 we denote $sup_D\ (e_1, e)$ the number of documents in D that indicate a relationship between e_1 and e_2. The nature of the indication can be defined individually according to the context. In the current implementation we say that a document indicates a relationship if both terms appear in the document in the same sentence. This has proved to be a strong indicator. Similarly, for a collection D and terms e_1, e_2 and c, we denote by $sup_D\ (e_1, e_2, c)$ the number of documents that indicate a relationship between e_1 and e_2 in the context of c (e.g., relationship between the UK and Ireland in the context of peace talks). Again, the nature of indication may be determined in many ways. In the current implementation we require that they all appear in the same sentence.

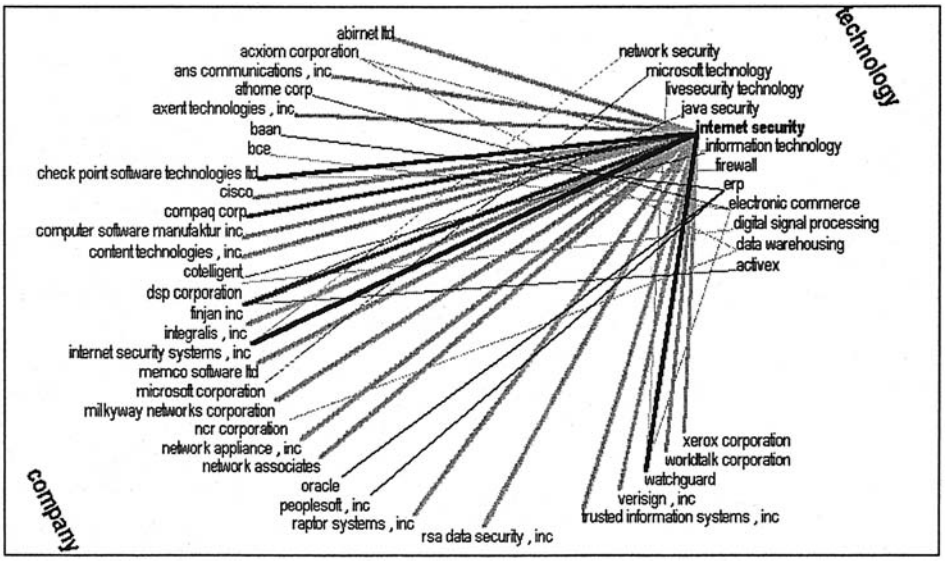


FIG. 21.23. Category connection map. The graph presents the connection between "companies" and "technologies." The information is based on 5,413 news articles obtained from Marketwatch.com. The user chose to highlight the "internet security" connections.

Category Connection Maps

Category connection maps (Aumann et al., 1999) provide a means for concise visual representation of connections between different categories, for example, between companies and technologies, countries and people, or drugs and diseases. To define a category connection map, the user chooses any number of categories from the taxonomy. The system finds all the connections between the terms in the different categories. To visualize the output, all the terms in the chosen categories are depicted on a circle, with each category placed on a separate part on the circle. A line is depicted between terms of different categories that are related. A color coding scheme represents stronger links with darker colors. An example of a category connection map is presented in Fig. 21.23. Here, the map is for the categories "companies" and "technologies." The underlying data set consists of 5,413 news articles downloaded from Marketwatch.com.

Formally, given a set $C = \{v_1, v_2, \ldots, v_k\}$ of taxonomy nodes and a document collection D, the category connection map is the weighted G defined as follows. The nodes of the graph are the set $V = terms\ (v_1) \cup terms\ (v_2) \cup \cdots \cup terms\ (v_k)$. Nodes $u, w \in V$ are connected by an edge if u and w are from different categories and $sup_D\ (u, w) > 0$. The *weight* of the edge (u, w) is $sup_D\ (u, w)$.

An important point to notice regarding category connection maps is that the map presents in a single picture information from the entire collection of documents. In the specific example of Fig. 21.23, there is no single document that has the relationship between all the companies and the technologies. Rather, the graph depicts aggregate knowledge from hundreds of documents. Thus, the user is provided with a bird's-eye summary view of data from across the collection.

Category connection maps are dynamic in several ways. First, the user can choose any node in the graph, and the links from this node are highlighted. In the example in Fig. 21.23, the user chose "Internet Security," and all the edges emerging from this node are highlighted. In

addition, a double-click on any of the edges brings the list of documents that support the given relationship, together with the most relevant sentence in each document. Thus, in a way the system is the opposite of search engines. Search engines point to documents, in the hope that the user will be able to find the necessary information. Category connection maps present the user with the information itself, which can then be backed by a list of documents.

Relationship Maps

Relationship maps provide a visual means for concise representation of the relationship between many terms in a given context. To define a relationship map the user defines:

- A taxonomy category (e.g., "companies"), which determines the nodes of the circle graph (e.g., companies)
- An optional context node (e.g., "joint venture"), which will determine the type of connection to be found among the graph nodes

Formally, for a set of taxonomy nodes vs, and a context node C, the relationship map is a weighted graph on the node set $V = terms\,(vs)$. For each pair $u, w \in V$ there is an edge between u and w, if there exists a context term $c \in C$, such that $sup_D\,(u, w, c) > 0$. In this case the weight of the edge is $\Sigma_{c \in C}\ sup_D\,(u, w, c)$. If no context node is defined, then the connection can be in any context. Formally, in this case the root of the taxonomy is considered as the context.

A relationship map for "companies" in the context of "joint venture" is depicted in Fig. 21.24. In this case the graph is clustered, as described below. The graph is based on 5,413 news documents downloaded from Marketwatch.com. The graph gives the user a summary of the

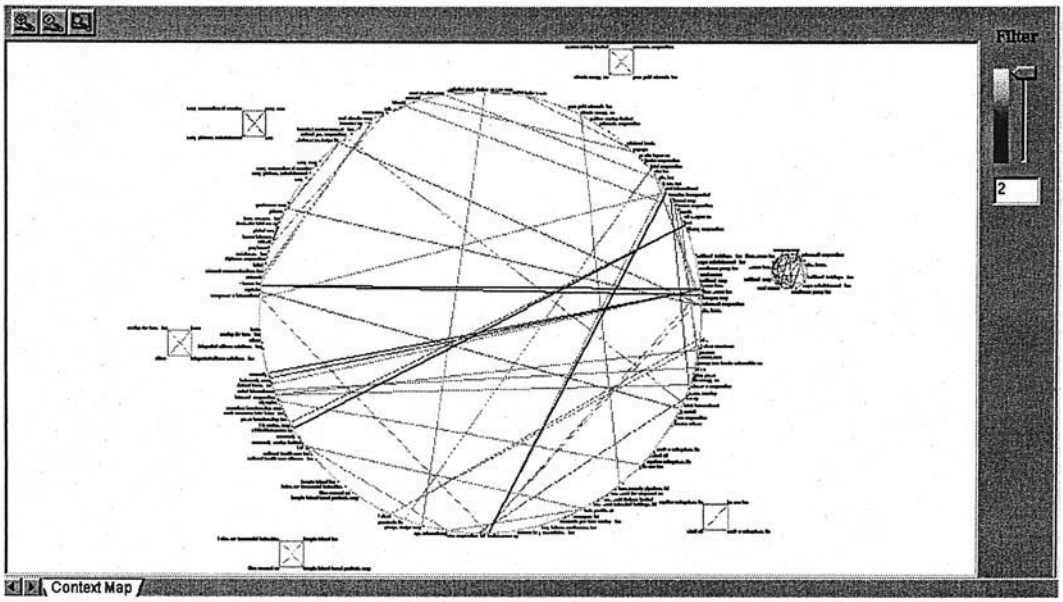


FIG. 21.24. Relationship map. The graph presents the connections between companies in the context of "joint venture." Clusters are depicted separately. The information is based on 5,413 news articles.

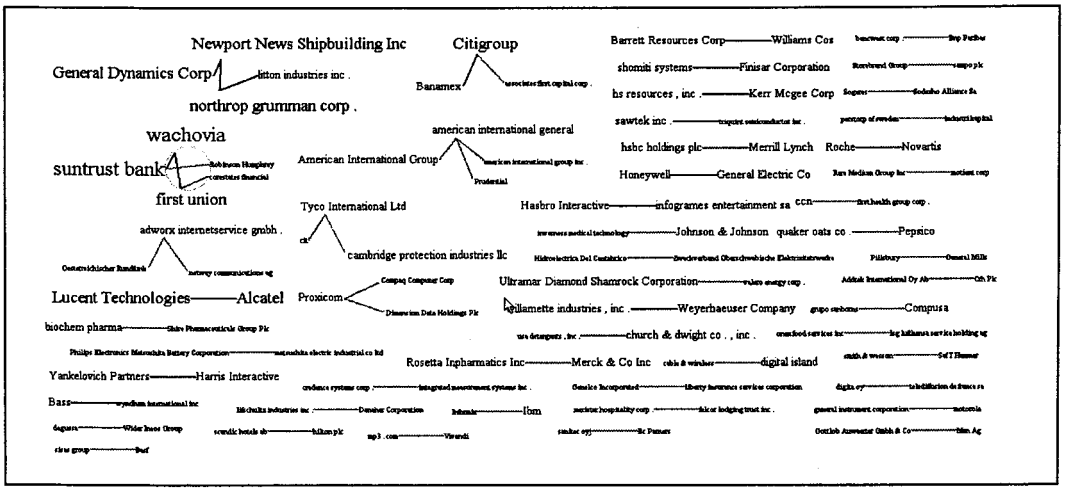


FIG. 21.25. A relationship map among all companies connected by any kind of acquisition relationship.

entire collection in one visualization. The user can appreciate the overall structure of the connections between companies in this context even before reading a single document.

A different type of visualization of relationships is shown in Fig. 21.25, which shows relationships between companies that are related by some type of acquisition relationship (planned, historic, or actual acquisition).

A spring graph is a two-dimensional graph in which the distance between two elements should reflect the strength of the relationships between the elements. The stronger the relationship the closer the two elements should be. An example of a spring graph is shown in Fig. 21.26. The graph represents the relationships between the people in a document

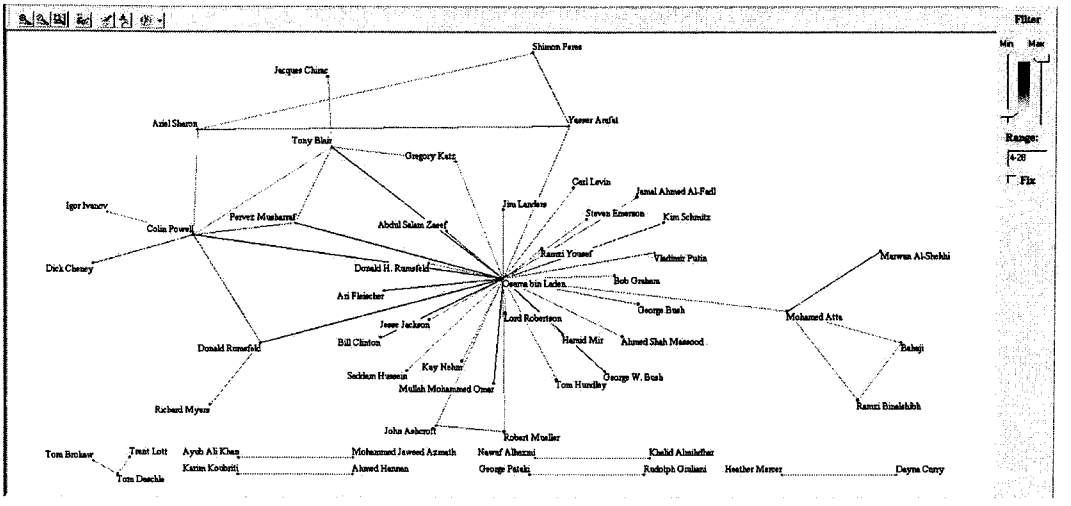


FIG. 21.26. Spring graph that shows the relationship between people around the 9/11 event (source: Yahoo News).

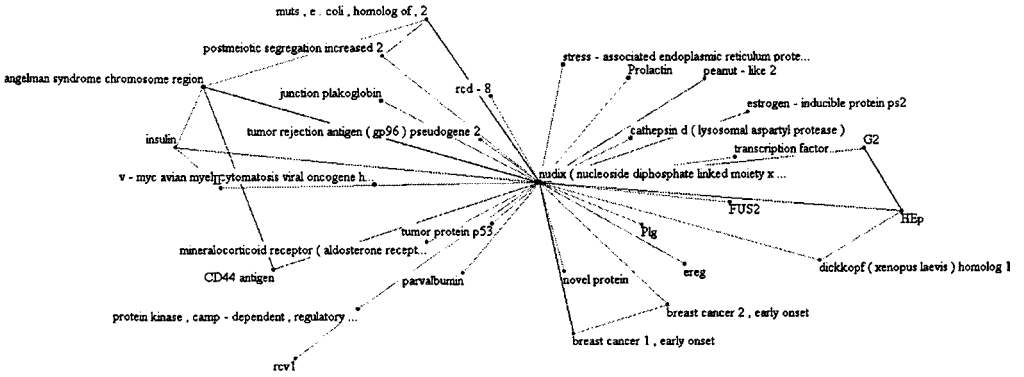


FIG. 21.27. Spring graph that shows the cooccurrence relationship between gene phrases (in the context of various cancer types). (Source: Medline.)

collection containing 2,210 news articles focusing on the 9/11 events. We can see that Osama bin Laden is at the center connected to many of the other key players related to the tragic events.

Another example of a spring graph is shown in Fig. 21.27. This figure depicts the co-occurrence (within the same sentence) relationships between gene phrases in the context of any type of cancer. The document collection is 50,000 Medline articles.

Clustering. For relationship map we use clustering to identify clusters of nodes that are strongly interrelated in the given context. In the example of Fig. 21.24, the system identified six separate clusters. The edges between members of each cluster are depicted in a separate small relationship map, adjacent to the center graph. The center graph shows connections between terms of different clusters and those with terms that are not in any cluster. We now describe the algorithm for determining the clusters.

Note that the clustering problem here is different from the classic clustering problem. In the classic problem we are given points in some space and seek to find clusters of points that are close to each other. Here, we are given a graph in which we are seeking to find dense subgraphs of the graph. Thus, a different type of clustering algorithm is necessary.

The algorithm is composed of two main steps. In the first step we assign weights to edges in the graph. The weight of an edge reflects the strength of the connection between the vertices. Edges incident to vertices in the same cluster should be associated with high weights.

In the next step we identify sets of vertices that are dense subgraphs. This step uses the weights assigned to the edges in the previous step.

We first describe the weight assignment method. To evaluate the strength of a link between a pair of vertices u and v, we consider the following two criteria.

Let u be a vertex in the graph. We use the notation $\Gamma(u)$ to represent the neighborhood of u. The *cluster weight* of (u, v) is affected by the similarity of $\Gamma(u)$ and $\Gamma(v)$. We assume that vertices within the same clusters have many common neighbors.

Existence of many common neighbors is not a sufficient condition, because in dense graphs any two vertices may have some common neighbors. Thus, we emphasize the neighbors that are close to u and v in the sense of *cluster weight*. Suppose $x \in \Gamma(u) \cap \Gamma(u)$; if the *cluster weights* of (x, u) and (x, v) are high, there is a good chance that x belongs to the same cluster as u and v.

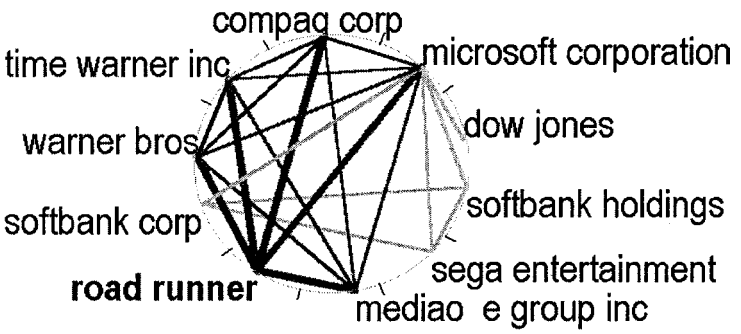


FIG. 21.28. Cluster details (details of one of the clusters from Fig. 21.24).

We can now define an *update operation* on an edge (u, v) that takes into account both criteria:

$$w(u, v) = \sum_{x \in \Gamma(u) \cap \Gamma(v)} w(x, u) + \sum_{x \in \Gamma(u) \cap \Gamma(v)} w(x, v)$$

The algorithm starts with initializing all weights to be equal, $w(u, v) = 1$ for all u, v. Next, the update operation is applied to all edges iteratively. After a small number of iterations (set to five in the current implementation) it stops and outputs the values associated with each edge. We call this the *cluster weight* of the edge.

The cluster weight has the following characteristic. Consider two vertices u and v within the same dense subgraph. The edges within this subgraph mutually affect each other. Thus, the iterations drive cluster weight $w(u, v)$ up. If, however, u and v do not belong to the dense subgraph, the majority of edges affecting $w(u, v)$ will have lower weights, resulting in a low *cluster weight* assigned to (u, v).

After computing the weights, the second step of the algorithm finds the clusters. We define a new graph with the same set of vertices. In the new graph we consider only a small subset of the original edges, the weights of which were the highest. In our experiments we took the top 10% of the edges. Because now almost all of the edges are likely to connect vertices within the same dense subgraph, we thus separate the vertices into clusters by computing the connected components of the new graph and considering each component as a cluster.

Fig. 21.24 shows a circle context graph with six clusters. The clusters are depicted around the center graph. Each cluster is depicted in a different color. Nodes that are not in any cluster are colored gray. Note that the nodes of a cluster appear both in the central circle and in the separate cluster graph. Edges within the cluster are depicted in the separate cluster graph. Edges between clusters are depicted in the central circle. Fig. 21.28 shows the details of one of the clusters (the cluster on the right). This cluster represents a group of companies with a strong interrelationship in the context of joint venture.

Interactive Features. The graphs are interactive. A double-click on any line segment brings up the list of documents that support the connection, together with the relevant sentence within each document. Fig. 21.29 shows this list of documents supporting the edge connecting "road runner" and "Microsoft" in Fig. 21.28. The system highlights the main terms relevant for the query. In this case these are "road runner," "Microsoft," and the context "joint venture." Another double-click on any document in the document list brings up the full text of the document. The most relevant sentence from each document is presented, and the relevant terms are highlighted.

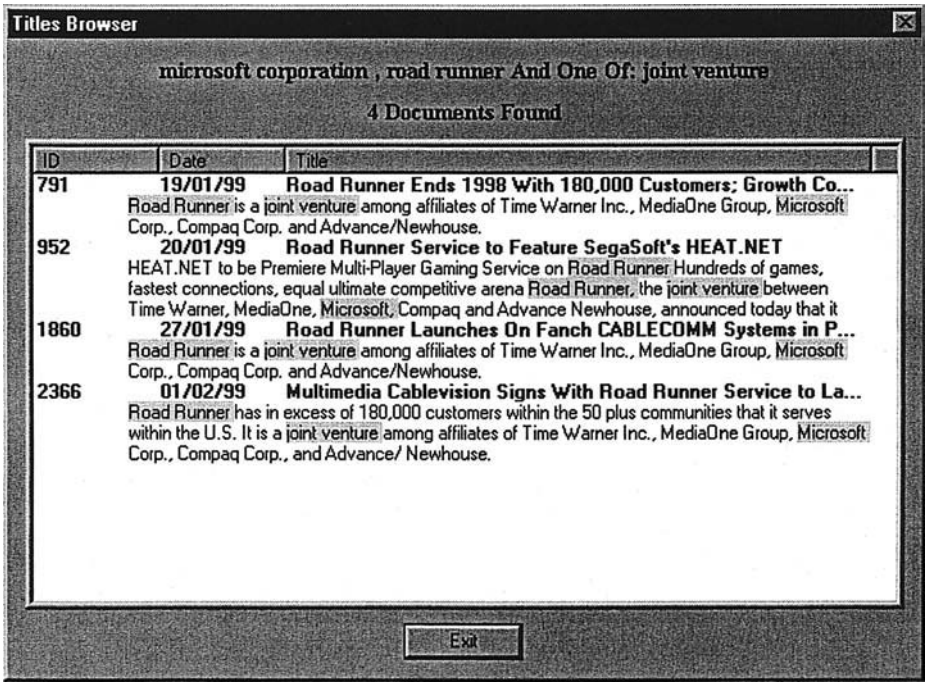


FIG. 21.29. Document list (the list of documents supporting the connection between "road runner" and "microsoft" in Fig. 21.28).

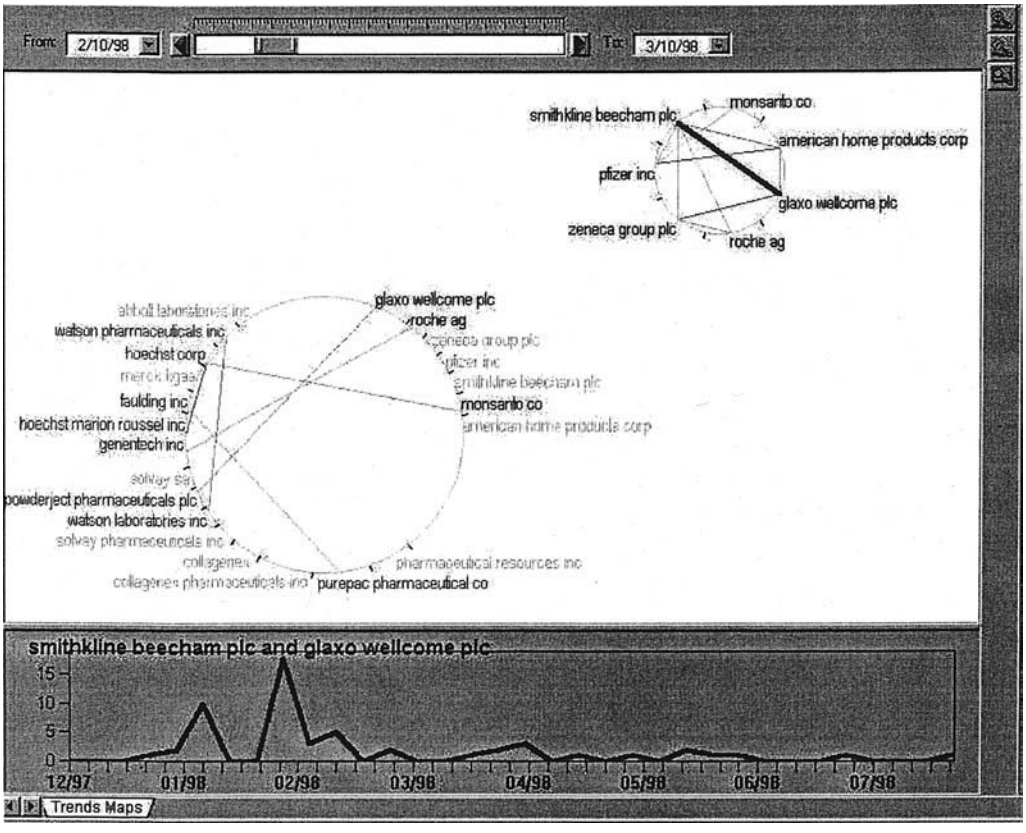
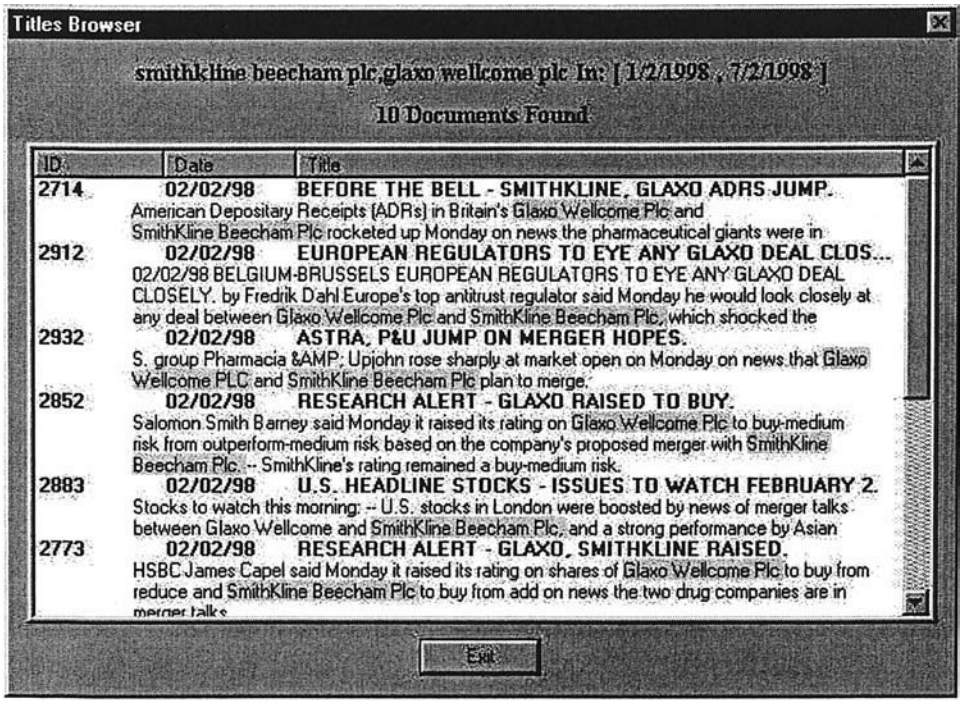


FIG. 21.30. Trend maps. Connections between pharmaceutical companies. Based on 64,000 Reuters news articles from 1998.



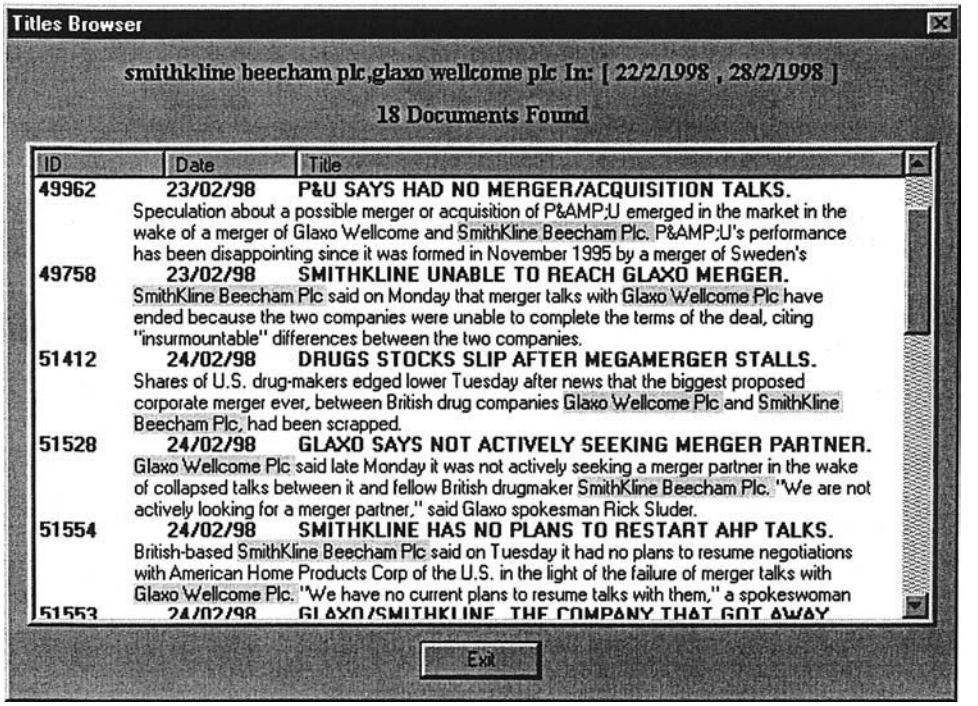


FIG. 21.31. Document lists for the two peaks of the graph in Fig. 21.30. The top list is for the first peak, the bottom list for the second.

Thus, with the context connection graphs the user first gets the global picture of all connections in a given context, then can zoom-in on any of the clusters or drill-down to the supporting documents. This allows for an effective and efficient discovery process.

Trend Graphs

Trend graphs are designed to allow the user to view changes in relationships over time, thus identifying trends and patterns. The basis for the definition of trend graphs is that of context connection graphs. Given a context connection graph, the associated trend graph is a dynamic graph that enables one to see the connections within each time period. Fig. 21.30 depicts a trend graph for the connection between pharmaceutical companies (no context). The circle in the upper right corner is the only cluster in the graph. The main graph is in the lower left corner. This graph is based on 64,000 news articles of Reuters from 1998.

Note the slider and the dates on the top part of the screen. The dates indicate that the edges shown in the current picture represent only connections based on articles published between 2.10.98 and 3.10.98. Moving the slider changes the time period, thus showing how the connections change over time. In addition, the length of the time interval can be changed.

Clicking on any single edge shows the trend for that pair. In the example in Fig. 21.30 the trend for the "smithkline beecham plc" and "glaxo wellcome plc" is shown. Notice the two peaks in the graph, one in January 1998 and one a month later in February 1998. Fig. 21.31 gives the list of articles supporting each peak. A quick look at these lists shows that the first peak is related to a merger announced between the two companies. The second peak is when the merger was called off! Thus, the graph helps to discover instantly the key elements of dynamics of the relationship between the two companies.

SUMMARY

Due to the abundance of available textual data, there is a growing need for efficient tools for text mining. Unlike structured data, in which the data mining algorithms can be performed directly on the underlying data, textual data requires some preprocessing before the data mining algorithm can be successfully applied. Information extraction has proved to be an efficient method for this first preprocessing phase. Text mining based on information extraction attempts to hit a midpoint, reaping some benefits from each of the extremes while avoiding many of their pitfalls. On the one hand, there is no need for human effort in labeling documents, and we are not constrained to a smaller set of labels that lose much of the information present in the documents. Thus, the system has the ability to work on new collections without any preparation, as well as the ability to merge several distinct collections into one (even though they might have been tagged according to different guidelines that would prohibit their merger in a tagged-based system). On the other hand, the number of meaningless results is greatly reduced, and the execution time of the mining algorithms is also reduced relative to pure word-based approaches. Text mining using information extraction thus hits a useful middle ground in the quest for tools for understanding the information present in the large amount of data that is only available in textual form. The powerful combination of precise analysis of the documents and a set of visualization tools enables the user to easily navigate and utilize very large document collections.

REFERENCES

Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D., & Tyson, M. (1993a). FASTUS: A finite-state processor for information extraction from real-world text. In *Proceedings of IJCAI-93* (pp. 1172–1178).

Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D., Kameyama, M., & Tyson, M. (1993b). The SRI MUC-5 JV-FASTUS information extraction system. In *Proceedings of the Fifth Message Understanding Conference (MUC-5)*.

Aumann, Y., Feldman, R., Ben Yehuda, Y., Landau, D., Lipshtat, O., & Schler, Y. (1999). Circle graphs: New visualization tools for text-mining. In *Principles of Data Mining and Knowledge Discovery, Third European Conference (PKDD'99)* (pp. 277–282). Prague: Springer.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging, *Computational Linguistics, 21,* 543–565.

Cohen, W. (1992). *Compiling prior knowledge into an explicit bias*. Working notes of the 1992 *AAAI Spring Symposium on Knowledge Assimilation,* Stanford, CA, March.

Cohen, W., & Singer, Y. (1996). Context Sensitive Learning Methods for Text categorization. In Proceedings of *SIGIR'96*.

Cowie, J., & Lehnert, W. (1996). Information extraction. *Communications of the Association of Computing Machinery, 39,* 80–91.

Daille, B., Gaussier, E., & Lange, J. M. (1994). Towards automatic extraction of monolingual and bilingual terminology, In *Proceedings of the International Conference on Computational Linguistics* (pp. 515–521).

Dumais, S. T., Platt, J., Heckerman, D., & Sahami, M. (1998) Inductive learning algorithms and representations for text categorization. In *Proceedings of the Seventh International Conference on Information and Knowledge Management (CIKM'98)* (pp. 148–155). New York: ACM Press.

Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19 (1).

Feldman, R., Aumann, Y., Amir, A., Klösgen, W., & Zilberstien, A. (1997). Maximal association rules: A new tool for mining for keyword co-occurrences in document collections, *Proceedings of the Third International Conference on Knowledge Discovery 1997* (pp. 167–170). Menlo Park, CA: AAAI Press.

Feldman, R., Aumann, Y., Finkelstein-Landau, M., Hurvitz, E., Regev, Y., & Yaroshevich, A. (2002). "A comparative study of information extraction strategies." *CICLing 2002,* (pp. 349–359). Lecture Notes in Computer Science 2276, Berlin: Springer.

Feldman, R., Aumann, Y., Liberzon, Y., Ankori, K., Schler, J., & Rosenfeld, B. (2001). A domain independent environment for creating information extraction modules. *CIKM 2001,* (pp. 586–588). New York: ACM Press.

Feldman, R., & Dagan, I. (1995). KDT—knowledge discovery in texts. In *Proceedings of the First International Conference on Knowledge Discovery,* (pp. 112–117). Menlo Park, CA: AAAI Press.

Feldman, R., & Hirsh, H. (1997). Exploiting background information in knowledge discovery from text. *Journal of Intelligent Information Systems 9,* 83–97. Norwell, MA: Kluwer Academic Publishers.

Feldman, R., Rosenfeld, B., Stoppi, J., Liberzon, Y. & Schler, J. (2000). A framework for specifying explicit bias for revision of approximate information extraction rules. *Proceedings of the Sixth International Conference on Knowledge Discovery 2000* (pp. 189–199). New York: ACM Press.

Fisher, D., Soderland, S., McCarthy, J., Feng, F., & Lehnert, W. (1995). Description of the UMass systems as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference* (pp. 127–140).

Frantzi, T. K. (1997). Incorporating context information for the extraction of terms. In Proceedings of *ACL-EACL'97* (pp. 501–503). San Francisco: Morgan Kaufmann.

Frawley, W. J., Piatetsky-Shapiro, G., & Matheus, C. J. (1991). Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro & W. J. Frawley (Eds.), *Knowledge Discovery in Databases* (pp. 1–27). Cambridge, MA: MIT Press.

Grishman, R. (1996). The role of syntax in information extraction. In *Advances in text processing: Tipster program phase II*. San Francisco: Morgan Kaufmann.

Hahn, U., & Schnattinger, K. (1997). Deep knowledge discovery from natural language texts. *Proceedings of the Third International Conference on Knowledge Discovery 1997* (pp. 175–178). Menlo Park, CA: AAAI Press.

Hayes, P. (1992). Intelligent high-volume processing using shallow, domain-specific techniques. In *Text-based intelligent systems: current research and practice in information extraction and retrieval* (pp. 227–242).

Hayes, P. J., & Weinstein, S. P. (1990). CONSTRUE: A system for content-based indexing of a database of news stories. In *Proceedings of Second Conference on Innovative Applications of Artificial Intelligence* (pp. 49–64). Menlo Park, CA: AAAI Press.

Hobbs, J. (1986). Resolving pronoun references, In B. J. Grosz, K. Sparck Jones, & B. L. Webber (Eds.), *Readings in Natural Language Processing* (pp. 339–352). San Francisco: Morgan Kaufmann.

Hobbs, J. R., Appelt, D. E., Bear, J., Israel, D., Kameyama, M., & Tyson, M., FASTUS: A system for extracting information from text. In *Proceedings of Human Language Technology* (pp. 133–137). Princeton, N.J.

Hobbs, J. R., Stickel, M., Appelt, D., & Martin P. (1993). Interpretation as abduction. *Artificial Intelligence, 63,* 69–142.

Hull, D. (1996). Stemming algorithms—a case study for detailed evaluation. *Journal of the American Society for Information Science, 47,* 70–84.

Ingria, R., & Stallard, D. (1989). A computational mechanism for pronominal reference. *Proceedings of the Association for Computational Linguistics* (pp. 262–271). San Francisco: Morgan Kaufmann.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of Tenth European Conference on Machine Learning* (pp. 137–142). New York: ACM Press.

Justeson, J. S., & Katz, S. M. (1995). Technical terminology: some linguistic properties and an algorithm for identification in text. In *Natural Language Engineering, 1,* 9–27.

Klösgen, W. (1992). Problems for knowledge discovery in databases and their treatment in the statistics interpreter EXPLORA. *International Journal for Intelligent Systems, 7,* 649–673.

Lappin, S., & McCord, M. (1990). A syntactic filter on pronominal anaphora for slot grammar. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics* (pp. 135–142).

Larkey, L. S., & Croft, W. B. (1996). Combining classifiers in text categorization. In *Proceedings of SIGIR-96* (pp. 289–297). New York: ACM Press.

Lehnert, W., Cardie, C., Fisher, D., Riloff, E., & Williams, R. (1991). Description of the CIRCUS system as used for MUC-3. In *Proceedings of the Third Message Understanding Conference* (pp. 223–233).

Lent, B., Agrawal, R., & Srikant, R. (1997). Discovering trends in text databases. In *Proceedings of the Third International Conference on Knowledge Discovery* (pp. 227–230). Menlo Park, CA: AAAI Press.

Lewis, D. D. (1995). Evaluating and optmizing autonomous text classification systems. In *Proceedings of SIGIR-95* (pp. 246–254).

Lewis, D. D., & Hayes, P. J. (1994). Guest editorial (Special issue). *ACM Transactions on Information Systems, 12,* 231.

Lewis, D. D., & Ringuette, M. (1994). A comparison of two learning algorithms for text categorization. In *Proceedings of Symposium on Document Analysis and Information Retrieval* (pp. 81–93).

Lewis, D. D., Schapire, R. E., Callan, J. P., & Papka, R. (1996). Training algorithms for linear text classifiers. In *Proceedings of SIGIR '96* (pp. 298–306). New York: ACM Press.

Lin, D. (1995). University of Manitoba: Description of the PIE system as used for MUC-6. In *Proceedings of the Sixth Conference on Message Understanding*.

Piatetsky-Shapiro, G., & Frawley, W. (Eds.) (1991). *Knowledge Discovery in Databases*. Menlo Park, CA: AAAI Press.

Rajman, M., & Besançon, R. (1997). Text mining: Natural language techniques and text mining applications. In *Proceedings of the Seventh IFIP 2.6 Working Conference on Database Semantics.* London: Chapman & Hall.

Riloff, E., & Lehnert, W. (1994). Information extraction as a basis for high-precision text classification, (Special issue). *ACM Transactions on Information Systems* (pp. 296–333). New York: ACM Press.

Sebastiani, F. (2002). Machine learning in automated text categorization, *ACM Computing Surveys, 34,* 1–47.

Soderland, S., Fisher, D., Aseltine, J., & Lehnert, W. (1995). Issues in inductive learning of domain-specific text extraction rules. In *Proceedings of the Workshop on New Approaches to Learning for Natural Language Processing at the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1314–1321). Menlo Park, CA: AAAI Press.

Srikant, R., & Agrawal, R. (1995). Mining generalized association rules. In *Proceedings of the 21st VLDB Conference* (pp. 407–419). San Francisco: Morgan Kaufmann.

Sundheim, B. (Ed.). (1993). *Proceedings of the Fifth Message Understanding Conference (MUC-5)*. San Mateo, CA: Morgan Kaufmann.

Tejada, S., Knoblock, C. A., & Minton, S. (2001). Learning object identification rules for information integration. *Information Systems, 26,* 607–633.

Tipster Text Program (Phase I), (1993). In *Proceedings, Advanced Research Projects Agency*.

Vapnik, V. (1979). *Estimation of Dependencies Based on Data* (Springer-Verlag, Trans.). Moscow: Nauka.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Berlin: Springer-Verlag.

Yang, Y., & Chute, C. G. (1994). An example-based mapping method for text categorization and retrieval. *ACM Transactions on Information Systems, 12,* 252–277.

Yang, Y., & Liu, X. (1999). A re-examination of text categorization methods. In *Proceedings of SIGIR '99* (pp. 42–49).