A Domain Independent Environment for Creating Information Extraction Modules

Ronen Feldman, Yonatan Aumann
ClearForest Corporation
1 World Trade Center
NY, NY
212-432-1515
ronen@clearforest.com

Yair Libetzon, Kfir Ankori
ClearForest Corporation
1 World Trade Center
NY, NY
212-432-1515
yair@clearforest.com

Jonathan Schler, Benjamin Rosenfeld
ClearForest Corporation
1 World Trade Center
NY, NY
212-432-1515
jonathan@clearforest.com

ABSTRACT

Text-Mining is a growing area of interest within the field of Data Mining and Knowledge Discovery. Given a collection of text documents, most approaches to Text Mining perform knowledge-discovery operations either on external tags associated with each document, or on the set of all words within each document. Both approaches suffer from limitations. This paper focuses on an intermediate approach, one that we call text mining via information extraction, in which knowledge discovery takes place on focused, relevant terms, phrases and facts, as extracted from the documents.

Categories and Subject Descriptors

Text Mining

General Terms

Visualizations, Development Environments, Information Extraction.

1. 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 structured format. It is estimated that up to 80% of the data available in digital format is non-structured 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 Data Mining on a collection of unstructured text is to determine the underlying information on which the Data Mining operations are applied. A straightforward approach is to use the entire set of words in the documents as inputs to the Data Mining algorithms. However, 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 ([2,3]) 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 importantly, the process of automated categorization requires defining the categories a priori, thus defeating the purpose of discovery within the actual text.

In this paper we focus on a third approach, which we call text mining via information extraction, 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 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 which any automated tagging system could handle.

While 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 etc.

In this paper we present ClearStudio, an integrated platform for developing the modules for information extraction in Text Mining applications. It provides an extensive language for defining the extraction process, by defining the types of information to be extracted, as well as how to how to identify this information. This is achieved by defining a set of rules for the information extraction engine. A set of rules for any given domain constitutes a rulebook. ClearStudio also includes a complete development environment for developing, compiling, testing, and debugging the extraction modules. This allows for easy creation and manipulation of information extraction rules. In this paper we will focus on the architecture that enables us to achieve a precision and recall that exceeds 90% for more than 120 different Event types.

2. RULEBOOK DEVELOPMENT

Developing a rulebook for a new domain can be very tedious and time consuming. To speed up the process we have created an environment with a range of productivity tools. It also provides tools for checking the quality of the rulebooks by examining its output on documents streams.

2.1 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 re-run the code to ensure that the problem has been fixed or the accuracy improved.

Central to all these operations is the Interpreter, which is able to act upon the code line by line without pre-compilation. 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.

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

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.

Match Features: 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 specific text in question. The success or failure of each component of the rule is then clearly shown in a report, featuring green checkmarks (success), red crosses (failure), and blue question marks (unchecked section). 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:* specifically, how long each predicate took to process a given document collection, and which in particular need tweaking, revising, or even complete removal in order to streamline the IE process. Its report is created as part of the compilation process, and thus the tool is typically applied at the end of the development process. Enabling the closest scrutiny of rulebooks is the low-level debugging tool integral to the IDE itself. It is similar to Match Features in that it tests the code against a sample text of the user's choosing, but more comprehensive as it examines the processing by the entire rule tile up to a breakpoint of their choosing, and all aspects of the process may be subsequently examined: from the full list of predicates and functions in the rule file , to the word classes and other resources actually loaded, the active windows and variables. *Stepinto, Stepover* and *Stepout* tools allow one to examine each rule call by call individually within the same stack frame or outside it.

3. DIAL (DECLARATIVE INFORMATION ANALYSIS LANGUAGE)

The rules are written in a language called DIAL (Declarative Information Analysis Language). DIAL is a language designed specifically for writing IE rules ([1,4]). The complete syntax of DIAL is beyond the scope of this paper. Here we describe the basic elements of the language.

3.1 Basic Elements

The basic elements of the language are syntactic and semantic elements of the text, and sequences and patterns thereof. Among these elements the language can identify:

- Predefined strings - e.g., "merger"
- Word class element: a phrase from a predefined set of phrases that share a common semantic meaning - e.g., WC-Countries, a list of countries.
- Scanner feature (basic characteristic of a token) e.g., @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 - e.g., noun or adjective
- Recursive Predicate Call - e.g., Company(C)

3.2 Constraints

Constraints carry out on-the-fly Boolean checks for specific attributes. These can be applied 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.

3.3 IE Rule Bases

The rule base is can be viewed as a logic program. Thus, a rule base, Γ, is a conjunction of definite clauses Ci: Hi ← Bi where Ci is a clause tag, Hi (called the head) is a literal and Bi = {Bil Bi2....} = Pi u Ni (called the body) is a set of literals, where Pi = {pij} is a set of Pattern Matching Elements and Ni = {nij} is a set of constraints operating on Pi. The clause Ci: Hit Bi represents the assertion that Hi is implied by the conjunction of the literals in Pi while satisfying all the constraints in Ni.

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

```
FMergerCCM(C 1, C2) :-
        Company(Compl)   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))
        @% @!
        {Cl = Comp1; C2 = Comp2} ;
```

The rule looks for a company name (carried out by the predicate Company, which returns the parameter Compl) followed by an optional phrase describing the company, and then the word "and". The system then skips up to ten 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 ten 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.

4. SUMMARY

Due to the abundance of available textual data, there is a growing need for efficient tools for Text Mining. Unlike structured data, where 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.

We presented the ClearStudio environment, which is an integrated environment to develop Information Extraction modules for efficient Text Mining. At the core, ClearStudio provides an extensive declarative rule language for defining the extraction process, and an execution module for executing these rules. In addition, ClearStudio provides a wide range of productivity tools, which facilitate efficient and rapid rule development, including: rule interpreter, debugger, profiler, and more. Also, ClearStudio provides a complete QA environment, with automatic feedback to the rule development process. ClearStudio was successfully used to develop a number of rulebooks, in diverse domains (e.g. financial news, patent analysis), several of which contain thousands of rules, and cover an entire rich domain. ClearStudio is used both within ClearForest, and by third party vendors developing rules for different domains. It has proved a robust, rich and diverse environment, providing a firm basis for text Mining.

5. ACKNOWLEDGMENTS

Our thanks to Michal Finkelstein, Yizhar Regev and Eyal Horovitz for helpful comments of drafts of this paper.

6. REFERENCES

[1] Appelt, Douglas E., Jerry R. Hobbs, John Bear, David Israel, Megumi Kameyama, and Mabry Tyson, 1993a. "The SRI MUC-5 JV-FASTUS Information Extraction System", Proceedings, Fifth Message Understanding Conference (MUC-5), Baltimore, Maryland, August 1993.

[2] Feldman R., and Hirsh H., 1996. Exploiting Background Information in Knowledge Discovery from Text. Journal of Intelligent Information Systems. 1996.

[3] Feldman R. and Dagan I., 1995. KDT - Knowledge Discovery in Texts. In Proceedings of the First International Conference on Knowledge Discovery, KDD-95.

[4] D. Fisher, S. Soderland, J. McCarthy, F. Feng and W. Lehnert, "Description of the UMass Systems as Used for MUC-6," in Proceedings of the 6th Message Understanding Conference, November, 1995, pp. 127-140.