Knowledge Pre-Processing:  
A Generic Approach Based on Compiler Function Concepts

Tapati Bandopadhyay, ICFAI Business School, Bangalore, India; E-mail: chatterjee_tee@yahoo.com
Pradeep Kumar, ICFAI Business School, Gurgaon, India; E-mail: pkgarg@ibsdel.org
Anil Kumar Saini, USMS, GGSIP University, New Delhi, India; E-mail: aksaini@rediffmail.com

ABSTRACT
Knowledge extraction from semi-structured or unstructured documents and texts have become a significant research issue in today's context when knowledge is viewed as the crucial corporate asset and capturing tacit or implicit knowledge and converting them into some reusable form have therefore become necessary. In this paper, a concept called knowledge pre-processing is proposed, to adequately exploit certain latent structured-ness in specific areas of the knowledge sources. The concept uses the basic principles of compilers, namely the lexical and semantic analyzers, parsers and thesaurus.

Keywords: Knowledge, Clustering, Pre-processing, Context Free language

1. INTRODUCTION
Knowledge pre-processing can make knowledge extraction processes faster and more resource-efficient. The basic functions of a pre-compiler can be used as a pre-processing unit, as analogous to the Oracle-pro*C kind of combinations. In case of such pre-compilers like pro*C with oracle, we see that the pre-compiler primarily acts as a filter and sends the classified inputs to different processing units or modules like a separate c compiler for processing the C programming sections and an SQL compiler for processing the “exec SQL...” statements. Similarly, if this concept gets applied in pre-processing knowledge elements for creating re-usable knowledge repositories which can store integrated knowledge elements across various sources, types and structures, the knowledge extraction, capture, conversion/translation (to the format acceptable to the repository) etc. i.e. the later steps become easier and faster.

In fact, many of the knowledge elements which are generally viewed as “unstructured” or “free-flowing texts” have some degree of explicit structured information for example embedded in their labels. Unfortunately, these already embedded “semi-structured” information which can help any extraction module to do some level 0 or “pre-classification”, do not get adequately exploited if the whole document along with the semi-structured part also is input at the beginning itself to the extraction modules. For example, there can be limited amount of “pre-classification” information embedded or available in the document headers, message headings, subject lines of letters or emails and so on. These, if adequately processed by a knowledge preprocessor before entering into the actual extraction phase, some classification information can already be made available through this pre-processing to the extraction modules.

Therefore, the benefits of a knowledge pre-processing unit to be placed before the actual knowledge extraction and capture modules can be explained as follows:

• It can help the knowledge extraction modules, which are often extremely resource-hungry and slow (due to less availability of such computational resources), more efficient. The knowledge extraction modules tend to become slow because of their unavoidable and extreme logical and processing complexities. A pre-processed input can make the logic simpler to some extent.

• It also helps the knowledge extraction modules to exploit some amount of structured information that remained embedded in part of unstructured documents like headings etc.

In this paper, we propose a generic model for knowledge pre-processor using the concepts of compilers in programming languages. However, the main difference between the two contexts (i.e. the programming language executable code generation vs. knowledge pre-processing) is essentially the fact that the output of the knowledge pre-processor is not any executable code etc. but some structured information about the knowledge source that is being input to a knowledge extraction module. The other significant difference which is a basic one is the fact that input for a compiler is a source code file with a specific programming language as using regular expressions and regular grammar, whereas in case of a knowledge pre-processor the input will be free flowing text strings for example as constructs in CFI(Context Free language).

There have already been some applications of compiler-related techniques for discovering classification information from unstructured text, like topic searching using lexical analysis, lexical chains etc. Here, our main purpose is NOT to extend any of these techniques or even enter into the searching algorithms, pattern search or thesaurus-based pattern matching algorithms which get applied to the entire body of the messages/documents etc. i.e. the whole of the unstructured inputs.

On the contrary, in this paper, we are proposing the concept of using a pre-processor based on similar concept like compilers, along with some explanations and examples of its possible use and benefits. Towards this end, we have first discussed some of the approaches for pattern discovery, subject identification, classification and clustering of unstructured/semi-structured documents. Then we take a clean-slate approach with zero assumptions about the concept of knowledge pre-processing, and develop a new generic model for doing the same. Therefore, the authors’ contribution starts from the section under heading “Generic model outline for knowledge pre-processor” which explains the basic framework of the knowledge pre-processor and its generic components, their roles and inter-relations.

2. UNSTRUCTURED TEXT HANDLING APPROACHES
There is various research issues related to unstructured/free-flowing text. The issues range from highly theoretical, mathematical, logical and analytical dimensions like discovering cohesions and relations between various sections of body texts (e.g. paragraphs), discovering topics, searching for topics. Further issues are related to the practical or implementations-specific side of the problem e.g. storing the discovered/searched information in a knowledge representation format which is more accessible, understandable, easy to implement, and easily retrievable to achieve the ultimate goal of re-usable knowledge repositories. These issues translate down to specific research questions like: text segmentation, topic tracking, topic detection, link detection, classification and clustering.

The background work for these issues have started since many years, starting from the machine readable dictionary-based approaches by McRoy(1992), Li(1995), then heuristics-based approached by McRoy(1992) etc. Topic segmentation issues have been worked upon by Hearst 1997 (topics boundaries discovered with slighting window-like systems), Kan 1998 (entity repetition-based concepts). Clustering techniques have also evolved over time, for example divisive clustering (Choi 2000), partitional and hierarchical clustering (He 2000). These works have culminated into further research work e.g. topic detection in unrestricted text using lexical cohesion (Chali 2001).
One of the methods for representing documents as networks using partial and hierarchical clustering techniques is further explained in this section, to compare its strength and applicability with the proposed knowledge pre-processing model here. This section is based on the work of him (2001) and Chen (2001). The original research was aimed at classifying hypertext documents, but the process logic is appealing for applications to any unstructured text domain. The basics of this process are as follows:

1. Any knowledge source/input is treated as unstructured documents.
2. Co-occurrence (He 2001) analysis is used to find the similarities and then consequently the dissimilarities between the documents. This is done as follows:

   \[ \text{Sim}_{ij} = \alpha \frac{A_{ij}}{|A|} + \beta \frac{S_{ij}}{|S|} + (1 - \alpha - \beta) C_{ij} / |C| \]  

   where, \( A \) is the asymmetric similarity score \( Ei \) and \( Ej \), and is calculated as follows:

   \[ S_{ij} = \text{sim}(Ei, Ej) = \left[ \frac{\sum d_k}{\sum d_k} \right] \]  

   where \( n \) is total number of terms in \( Ei \), \( m \) is total number of terms in \( Ej \), \( p \) is total number of terms that appear in both \( Ei \) and \( Ej \), \( dj \) is (Number of occurrence of term \( j \) in \( Ei \) X log \((N/dfj)X(wj/\text{Termtype factor})\)) and \( Cij \) is the number of terms that appear in both \( Ei \) and \( Ej \) (co citation/cross referencing matrix).

3. The generic model of knowledge pre-processor, as explained in the section above, is shown in Figure 1.

4. The sub-modules of the knowledge-preprocessing module:

   - Lexical information extractor: This is designed in line of lexical analyzer in compilers, the main differences being that in case of compilers, the output of a lexical analyzer is a symbol table with tokens, lexemes and patterns. But here the output of a lexical analyzer will be broken-down fragments of the subject sentence into nouns/verbs/adjectives/adverbs etc. (identification of a noun/verb and its subgroups e.g. names/objects/functions etc. can be done by using pattern matching and thesaurus). If we represent this analogy as in Figure 3, we get the symbol table equivalent in knowledge pre-processor as shown in table 1 inside Figure 1.

3. GENERIC MODEL OUTLINE FOR KNOWLEDGE PRE-PROCESSOR

The generic model of knowledge pre-processor, as explained in the section above, is shown in Figure 1.

Explanation of the sub-modules of the knowledge-preprocessing module:

- Input knowledge sources: Documents, Messages, Etc.
- Preprocessors:
  - Lexical information extractor:
  - Syntactic information extractor:
  - Semantic information extractor:
- Outputs:
  - Level 0 classifier/identifier information structure
  - Knowledge base: knowledge map network/clusters/other schemes
  - Unstructured text handling modules
  - Final output of preprocessing module

Figure 1. Positioning the knowledge pre-processor in the context of creating a re-usable knowledge base/repository with unstructured sources

Figure 2. Knowledge pre-processor – basic building blocks and their outputs

Figure 3. Knowledge pre-processor with unstructured text handling modules

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
Table 1. A minimal view of the lexical extractor output with two example input strings

<table>
<thead>
<tr>
<th>Type of string component</th>
<th>Noun</th>
<th>Verb</th>
<th>Adjectives</th>
<th>Adverbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputString1: substring1</td>
<td>[Microwave] [model no. 2021]</td>
<td>[purchased]</td>
<td>[not functioning]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[year 2002]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InputString1: substring2</td>
<td>[table]</td>
<td>[not rotating]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InputString2: substring1</td>
<td>[Microwave] [model no. 4576]</td>
<td>[purchased]</td>
<td>[not functioning]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[year 2005]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InputString2: substring2</td>
<td>[heating]</td>
<td>[not proper]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
This whole string can be stored as an identifier with the numbers as indices for specific values as 13-21-4-3, just to remember the structure of the phrase. This information can be further added as the syntactic information for the phrases which would help in easy reconstruction of the phrases and subsequently easy and highly understandable retrieval. Also, the connectors may give valuable information, e.g. in this example case the symbol ‘::’ depicts a further explanation of the problem, whereas in other cases the same symbol might mean different things e.g. cause-and-effect link between the two constructs. So, the connector along with its semantic role as a connector (e.g. a further explanatory/ a cause-and-effect link) will also have to be stored as part of the semantic extractor’s job.

The rest of the example can be worked upon using further concepts on syntax and semantic analysis, as has already been mentioned before. Also, we can combine this model with the LC or co-occurrence analysis models as explained in earlier sections and can make the process more efficient.

4. CONCLUSION

This paper presents a fresh approach for knowledge extraction from unstructured sources using the concept of a pre-processor and the tried and tested concepts of traditional compiler construction in theoretical as well as applied computer sciences domain. The primary advantage of having a knowledge pre-processor, as has been explained in the first section of this paper, is the fact that a pre-processor can perform a level 0 analyzing and discover or present a basic identifier or classifier for an unstructured knowledge source by exploiting some amount of structured string-type information that are usually present in the source headers or document labels or message subjects/headers. This way it can reduce the workload of a knowledge extraction module which can then take the entire body-text of the document/ message/ knowledge source and apply the well-researched approaches of unstructured text handling on them. This way the entire process of knowledge extraction becomes faster and more resource-efficient. Further research possibilities include detailed design and implementation of the sub-modules under the knowledge pre-processor and exploiting the opportunities there again to use the tried and tested concepts of compilers, theory of computer science, theory of languages like regular grammar and CFL etc. With reference to the model presented in this paper, there are research issues in terms of scalability of the model e.g. the volume of unstructured data as well as heterogeneous source support-systems that can be handled by the model. Also there are issues related to the implementation, performance, resource utilization and tuning of any system based on this model which includes questions like which algorithms to choose for unstructured information handling, topic detection, preliminary information extraction, clustering etc., how to optimize the resource utilization for these algorithms, how to improve performance of an actual knowledge preprocessor and so on. Therefore, the model presented in this paper can be extended in multiple dimensions including theoretical aspects like algorithms design and analysis to implementation aspects including scalability and performance issues.

REFERENCES


