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 CFL(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 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 partitional 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:

- Any knowledge source/ input is treated unstructured documents
- Co-occurrence (He 2001) analysis is used to find the similarities and then consequently the dissimilarities between the documents. This is done as follows:
  Co-occurrence analysis converts data indices and weights obtained from inputs of parameters and various document sources e.g. email/text message bodies, into a matrix that shows the similarity between every pair of such sources.(He et al 2001,He and Hui 2002, Shneidermann 1996)
  When measured between two documents, say Ei and Ej,

  $$Sim_{ij} = \alpha \{A\,ij \;/\; |A|2\} + \beta\, Sij \;/\; |S|2 + (1- \alpha - \beta)\, Cij \;/\; |C|2 \quad (1)$$

  $0< \alpha, \beta$ (parameters) <1, $0 <= \alpha + \beta <=1$,

  where A, S, and C are matrices for A_{ij}, S_{ij}, and C_{ij} respectively. Values for A_{ij} will be 1 if Ei has a direct link/ reference/ hyperlink to Ej, else 0. S is the asymmetric similarity score Ei and Ej, and is calculated as follows:

  $$S_{ij} = \text{sim}(Ei, Ej) = [[\sum_{k=1}^{p} d_{ki}\, d_{kj}] \,/\, [\sum_{k=1}^{n} d^2 d_{ij}]] \quad (2)$$

  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., dij = (Number of occurrence of term j in Ei) X log((N/dfj)Xwj)X(Termtype factor); dfj is number of documents containing term j; wj is number of words in term j; Termtype factor = 1 + ((10-2 X typej / 10), where typej = min 1 if term j appears in subject, 2 if it appears in body, 3 if it appears in 'note' etc.) and Cij is number of Es pointing to both Ei and Ej (co citation/ cross-referencing matrix).
- Document bodies which are very similar in terms of their contents i.e. many of the identified key-terms (i.e. Terms excluding the general terms like pro-nouns, prepositions, conjunctions etc.)are same, can be clubbed up together to form a cluster. Dissimilar document bodies can be created as other clusters.
- These clusters can then form a network using hierarchical and partitional clustering method to form a graph with the nodes as representative knowledge maps for a particular group of documents with high-similarity in their body text.
- Partitioning of a graph, say G, can be done in various ways, for example, by using similarity measures as below: (Rich and Knight 2001,Shi and Malik 2000)

Normalized Cut (x) = {cut between (A, B)/ assoc(A, V)} + {cut between (A, B)/ assoc (B,V) (3)

where, Cut between (A,B) = ∑i€A, j€B Sim_{ij} , Simij is similarity between nodes i and j of the graph. Assoc(A,V) and assoc(B,V) shows how on average nodes within a group are connected to each other. A cut on a graph G = (V, E) is defined as removal of a set of edges such that the graph is split into disconnected sub-graphs. (Chen et al 1998,Chen et al 2001)

Now, this approach can work fine when the whole document has no element of structure in it at all i.e. any headers / titles / subject lines etc., or these also are combined together along with the body text and are processed together as well, not separately. This property is the main strength as well as weakness of this approach in specific and these kind of clustering-based approaches in general. The strength is that it can handle the whole document as a whole. The weakness is, in doing so, 1) It fails to exploit whatever little structure-related information that is embedded in some part of the document structure itself e.g. label, headings etc., 2) the complex and repetitive nature of the algorithm makes it extremely resource-intensive and in absence of such intensive or dedicated resources, extremely slow.

Other approaches like lexical chains suffer from similar constraints. Lexical chains arise from concepts of lexical cohesion that may arise from semantic connections between words (Chali 2005). Deriving the cohesion structure of a text is equivalent to retrieving lexical chains like LC = {w_1, w_2, …, w_n}. These approaches while working fine with entire text as inputs, as is the case of topic discovery, searching or matching, do not again exploit certain default structured properties of text documents.

The concept of LCs however, can be used appropriately within the context of this paper as well, i.e. we can create the first level of document identifiers or classifiers by applying these LC-discovery concepts to the document label information itself e.g. the heading/ subject lines etc. We have actually used the concept similar to that of Roget's thesaurus as explained by Chali 2005, in the lexical analysis equivalence part of our model.

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:

- 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. (the 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 3.

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

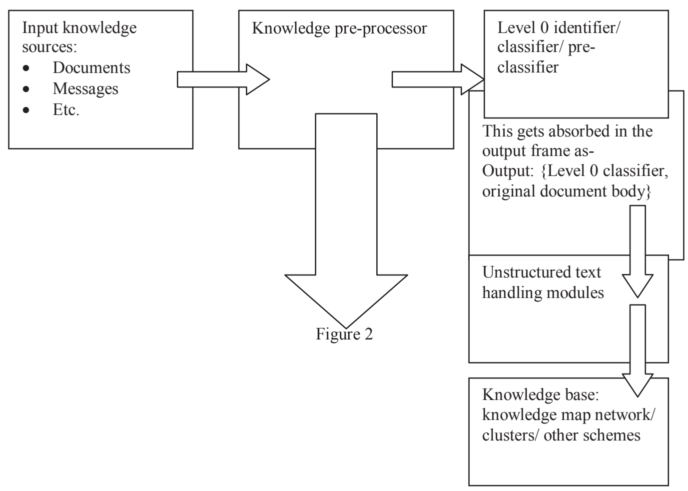


Figure 2. Knowledge pre-processor – basic building blocks and their outputs

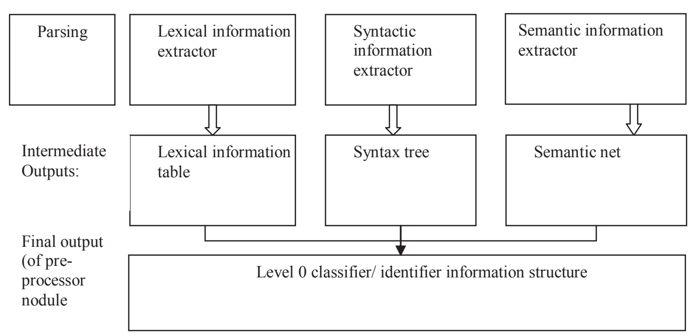
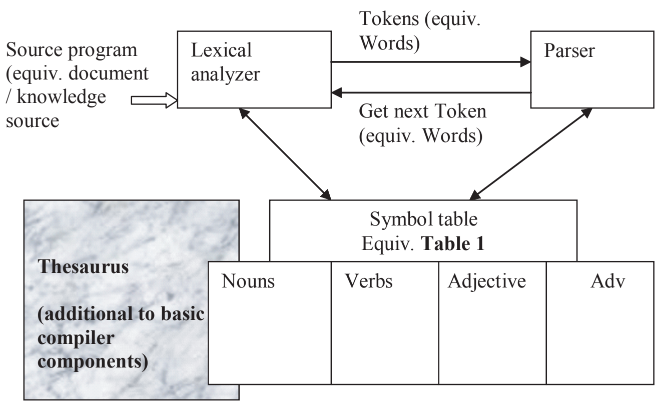
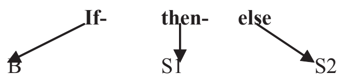


Figure 3. Lexical information extractor as an analogous equivalent of a lexical analyzer in a compiler



- Syntactic information extractor: The syntax extractor can draw it's equivalence to the syntax directed definitions including the annotated parse trees, dependency graphs, evaluation order-based graphs and syntax trees. A syntax tree can be thought as a condensed form of parse tree useful for representing language constructs. For example a production rule-type knowledge presentation scheme can appear as a syntax tree in the following form:



For a production rule: If B then S1 else S2.

- Semantic information extractor: Can be designed with an equivalence of semantic analyzer. The output can take form of a semantic network.

In the following section, we use a simple example drawn from a practical application situation and take this example through the initial steps in the knowledge pre-processor, basically onto up to the lexical analyzer equivalent part. This example can be further worked upon for generating the syntax trees as explained briefly above. And then it can be taken further to form it's semantic net equivalent.

AN EXAMPLE

Suppose there are two customer e-mail messages about rouble-shooting, to be input to a CRM knowledge base. The email messages have their subject lines in a fairly structured fashion, as they have used the pre-defined form fields of customer feedback forms on the company websites. These subject fields are considered as two inputs strings in this example. They are as follows:

InputString1: {Microwave model no. 2021 purchased in year 2002 not functioning: the table is not rotating}

InputString2: {Microwave model no. 4576 purchased in year 2005 not functioning: heating is not proper}

First level of lexical analysis on these two strings may generate output as follows:

InputString1: [Microwave model no. 2021 purchased in year 2002 not functioning]: (considered as connector) [the table is not rotating]

InputString2:[Microwave model no. 4576 purchased in year 2005 not functioning]: [heating is not proper]

2nd level: nouns (match from dictionary of nouns: can be made restricted to contexts: e.g. names (e.g. in case of customers complaining about service etc. by names), objects (as microwave in this example), place-names, function-names(e.g. 'heating' in the 2nd input string and so on)

InputString1: [[Microwave] [model no. 2021] purchased in [year 2002] not functioning] : [the [table] is not rotating]

InputString2:[[Microwave] [model no. 4576] purchased in [year 2005] not functioning]: [heating is not proper]

3rd level: verbs

InputString1: [[Microwave] [model no. 2021] [purchased] in [year 2002][not functioning]] : [the [table] is [not rotating]]

InputString2:[[Microwave] [model no. 4576] [purchased] in [year 2005] not functioning]: [heating] is not proper]

4th level: qualifiers/ adjectives

InputString1: [[Microwave] [model no. 2021] [purchased] in [year 2002][not functioning]] : [the [table] is [not rotating]]

InputString2:[[Microwave] [model no. 4576] [purchased] in [year 2005] not functioning]: [heating] is [not proper]]

Now, suppose we construct a table to store these strings as analyzed by the lexical extractor we get Table 1.

This table can be further fine-tuned, for example, by using a look-up table with index values for all these word-types and their sequential combinations e.g. 1 for nouns, and then 11 for names, 12 for objects, 13 for verb-type nouns e.g. function-names (like 'heating'), 2 for verbs(21 for auxiliary verbs, 22 for continuous tense …), 3 for adjectives, 4 for binary(yes/ no-not) response and so on. So, a phrase like 'Heating is not proper' can be expressed using this preliminary look-up table would be

Table 1. A minimal view of the lexical extractor output with two example input strings

Type of string component =➔	Noun	Verb	Adjectives	Adverbs
InputString1: substring1	[Microwave] [model no. 2021] [year 2002]	[purchased] [not functioning]		
InputString1: substring2	[table]	[not rotating]]		
InputString2: substring1	[Microwave] [model no. 4576] [year 2005]	[purchased] [not functioning]		
InputString2: substring2	[heating]		[not proper]	

<13>	<21>	<4>	<3>
[Heating]	[is]	[not]	[proper]

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/headings. 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 tired 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

Chen, H.; Fan. H.; Chau. M.; and Zeng, D.(2001) *Meta Spider: Meta Searching And Categorization On The Web*. Journal of the American Society for Information Science and Technology. 52(13). 1134-1147.

Chen. H.; Chung, Y.; Ramsey. M.; and Yang. C.(1998) A smart itsy bitsy spider for the Web. *Journal of the American Society far Information Science. 49*. 7, 604-618.

Chali, Y.(2001). *Topic Detection Using Lexical Chains*. Proceedings of the Fourteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Budapest, Hungary, Lecture Notes in Computer Science 2070, Springer-Verlag. 552–558.

Chali, Y. (2005) Topic Detection Of Unrestricted Texts: Approaches And Evaluations, Applied Artificial Intelligence 19(2): 119-135.

Choi, F. Y.(2000). *Advances In Domain Independent Linear Text Segmentation*. Proceedings of the 1st North American Chapter of the Association for Computational Linguistics, Seattle, Washington, USA , 26–33.

Hearst, M. A.(1997). *Texttiling: Segmenting Text Into Multi-Paragraph Subtopic Passages*. Computational Linguistics 23(1):33–64.

He, Y, and Hui. S.C. (2002) *Mining A Web Citation Database For Author Co-Citation Analysis.* Information Processing and Management. 38(4). 491-508.

He. X.; Ding. C; Zha. H.; Simon, H. (2001) *Automatic Topic Identification Using Webpage Clustering*. Proceedings of the 2001 IEEE International Conference on Data Mining. Los Alamitos. CA: IEEE Computer Society Press. 2(X)I. 195-202.

Kan, M.-Y., K. R. McKeown, J. L. Klavans. (1998). Linear segmentation and segment relevance. In Proceedings of 6th International Workshop of Very Large Corpora (WVLC-6), , Montre´al, Canada, 197–205.

Li, X., S. Szpakowicz, S. Matwin. (1995). *A Wordnet Based Algorithm For Word Semantic Sense Disambiguation*. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Montre´al, Canada, 1368–1374.

McRoy, S. 1992. *Using Multiple Knowledge Sources For Word Sense Disambiguation*. Computational Linguistics 18(1):1–30.

Rich E., Knight K.(2001), *Artificial Intelligence*, Tata McGrawHill Publishing Company Ltd, N. Delhi.

Shi. J., and Malik. J.(2000) Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence. 22*. S (2(X)0), 8S8-905.

Shneidermann, B.(1996) *The Eyes Have It: A Task By Data Type Taxonomy For Information Visualizations*. Proceedings of IEEE Symposium on Visual languages. Los Alamitos, CA: IEEE Computer Society Press, 336-343.