Incremental Information Extraction Using Dependency Parser

A.Carolin Arockia Mary[1]
PG Scholar, ME-Software Engineering

S.Abirami[2]
PG Scholar, ME-Software Engineering

A.Ajitha[3]
Assistant Professor, Computer Science and Engineering

Abstract

Information extraction is a technique to extract particular kind of information from large volume of information using the pipeline approach. Failure of this approach is that whenever a new extraction goal is needed or a module is changed, extraction is applied from the initial to the whole text corpus. Small changes in corpus might also affect the entire process. In Information Extraction goals must be in the form of database queries. This has been evaluated and optimized by database system. Database queries are responsible to perform generic extraction and also reduce the reprocessing time by performing incremental information extraction by identifying the part of the data which is affected by the change. Incremental information extraction generates the queries automatically so that it reduces the user's time of learning the query language. Focus of information extraction i.e., efficiency and quality of extraction results is also achieved in incremental information extraction. If a new module is deployed then the incremental information extraction approach reduces the 89.64 percent processing time than the traditional pipeline approach.

Keywords- Dependency parser, information extraction, information retrieval, query language, relational database, Text mining.

1. Introduction

It is estimated that each year more than 6, 00,000 articles are published in the biomedical literature, with close to 20 million publication entries being stored in the Medline database. Extracting information from such a large corpus of documents is very difficult. So it is important to perform the extraction of information by automaticity. Information Extraction (IE) is the process of extracting structured information from the unstructured information. Entities and the relationship between the entities are the examples of the structured information. Unstructured information means it's just a random piece of information.

IE is a one-time process. It extracts the entities and a specific type of relationships from the collection of documents. IE is implemented as a pipeline of special-purpose modules.

Due to the demand of information extraction in several domains, the various frameworks such as UIMA [1] and GATE [2] has been developed. These kind extraction frameworks are usually file based and the data which is processed in this are used between components. In this Relational databases play a limited role of storing the extracted relationships.

File-based frameworks are only applicable for one-time extraction, because IE is performed continuously on the same document collection even though a small change in the extraction goal. Consider the scenario that the processing of web documents with modified content the availability of updated ontologies or improved components for named entity recognition, and the realization of new target relationships for extraction. If the existing extraction framework is used in any of the scenarios then it is necessary to reprocess the entire text collection, which can be a large process and also computationally expensive. If the extraction goal is changed then it needs the unnecessary reprocessing on the entire text collection. In another scenario if the extraction goal remains same but an updated ontology or an improved model based on statistical learning approach becomes available for named entity recognition. Changes in these scenarios only affect a portion of the text corpus. So a framework which has the capability of managing processed data and performing incremental extraction to identify which part of the data is affected by the change of components or goals.

Incremental information extraction framework uses database management system as an essential component. Database management system serves the

dynamic extraction needs over the file-based storage systems. Text processing components of named entity recognition and parsers are deployed for the entire text corpus. The intermediate output of each text processing component is stored in the relational databases called as Parse Tree Database (PTDB). Database query which is used to extract the information from the PTDB is in the form of Parse Tree Query Language (PTQL).

If the extraction goal is changed or a module is updated then the corresponded module is deployed and the processed data is populated into the PTDB with the previously processed data. Database queries are given for the extraction and also to identify the sentences with newly recognized mentions. If the changed sentences are identified then extraction is performed only on those sentences rather than the entire corpus. Unlike the file-based pipeline approach, incremental information extraction framework approach stores the intermediate processed data of each component; this avoids the need of reprocessing on the entire text corpus. Avoiding such reprocessing of data is most important for information extraction because it reduces the extraction time tremendously.

The contribution of this is that:

Novel Database-Centric Framework for Information Extraction: Unlike Traditional IE approaches, this new extraction approach stores the intermediate text processing data in the PTDB. Extraction is performed by the PTQL queries. So it is not necessary to write and run any special purpose programs for the extraction need. Also it minimizes the reprocessing time needed for the new extraction goal by the deployment of the improved processing component.

Query Language for Information Extraction: Goals are expressed as queries on parse tree database. XPath [3] and XQuery [4] languages are not suitable for extracting linguistic patterns because several important expressive features required for linguistic queries are missing or hard to express in this. So the query language Parse Tree Query Language (PTQL) is implemented.

Automated Query Generation: Learning and writing the extraction queries manually is a time consuming and labor-intensive process. This may achieve an unsatisfactory extraction performance. So to avoid this two algorithms are used to generate extraction queries automatically in the presence and absence of training data respectively. This reduces the user's effort on performing extraction.

2. Background

2.1. Stanford Dependency Parser

The Stanford typed dependencies representation was designed to provide a simple description of the grammatical relationships in a sentence that can easily be understood and effectively used by people without linguistic expertise who want to extract textual relations. Stanford dependencies (SD) are triplets: name of the relation, governor and dependent.

The Stanford Parser and the Link Grammar parser produce a forest of parse trees. Each syntactic possible interpretation of a sentence is called an analysis. The Link Grammar and the Stanford parsers share the same interfaces. They can return as many analyses as needed.

Differences between both parsers:

Both parsers share the same structure (i.e. constituents and dependencies). Nevertheless, there must be some subtle differences:

- If the constituents' labels are normalized ("NP", "VP", "PP"…), that is not the case for the **dependencies labels**; for instance, the 'subject' dependency between a noun and a verb is labelled 'S' in the Link Grammar, and 'nsubj' in the Stanford Parser.
- **Dependencies structure** is far from being identical in both systems.

Consider a sentence "John's arm is broken" as example. The Parse tree structure for both parsers is shown in the figure1.

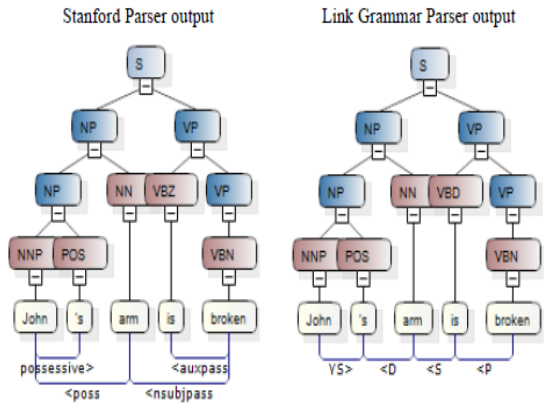

Figure 1. Parser output for the sentence "John's arm is broken"

3. Related Work

Information extraction is one of the major research areas over many past years. The objective of this is that

improving accuracy of the extraction systems. This section, describes how this incremental information extraction framework differs from traditional IE systems, rule-based IE systems.

3.1. Traditional IE Approaches

Popular file-based frameworks of UIMA [1] and GATE [2] IE frameworks have the ability to integrate various NLP components for IE. These frameworks do not store the intermediate processing data. QXtract [5], Snowball [6] systems uses the RDBMS to store and query the extracted results. Cimple [7], SystemT [8] systems use the joint operations in RDBMS. This extracts the results that are stored in various database tables. All these frameworks do not store any intermediate processed data, so if a component is improved or extraction goal is changed then all components have to be reprocessed from the initial. This consumes more time and also has the high computational cost. In order to reduce this high computational cost the most common approach called document filtering is used. Thus the filtered documents are called as promising documents which are used in [5], [9], and [10]. Extraction is performed on those promising documents and then the relevant documents are retrieved from this. This the filtering approach fully based on the sentences that are selected individually based on the lexical clues. These clues have been provided by the parse tree query language. Also this filtering process utilizes the efficiency of the IR engines.

3.2. Rule-Based IE Approaches

Rule-based IE approaches used in [11], [12], [13], and [14]. Avatar System [14] uses the AQL query language; this has the capability of performing IE task by matching it with regular expressions. But this query language does not support the traversal on parser tree. DIAL [12], TLM [13], KnowItNow [15] systems are fully based on the relationship extraction. They use their own query languages. But all these query languages only supports querying of data from the shallow parsing they do not provide the capability of performing extraction using the rich grammatical structures. Cimple [7], SystemT [8], Xlog [16] used the declarative languages. Joins operations are performed on the RDBMS and then rules are applied for the integration of the different extracted results. However, these rules are not capable of querying parse trees. MEDIE [11] stores the parse tree in the database and it allows the query language for the extraction over this parse trees. The XML-like query languages such as XPath [3], XQuery [4] are based on one kind of dependency grammar called head driven phrase structure (HPSG). Link types cannot be expressed in this query language as in PTQL.

4. Problem Statement

File-based approaches to data storage are based on relatively simple data structures, such as the Indexed Sequential Access Method (ISAM), and are usually implemented for a single application. Files are generally created on an as needed basis to service the data needs of an application. The files are associated with an application.
The disadvantages founded in the file-based frame wok are:

- File-based approaches do not recognize relationships between entities until such information is needed by an application.
- File-based frameworks are suitable for one-time extraction, because IE has to be performed repeatedly even on the same document collection.

5. System Design

This new extraction frame work consists of two phases. They are: Initial Phase used to processing the text, Extraction Phase used to perform the extraction.

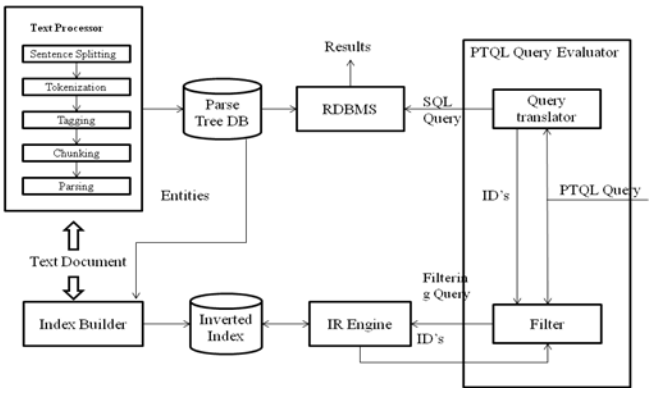

Figure 2. System Architecture

Initial Phase: Text processor is responsible to perform a one-time parse, entity recognition, and tagging on the whole corpus based on the current knowledge. This processed text is stored in a relational database, called parse tree database (PTDB).

Extraction Phase: Extraction is then achieved by PTQL. PTQL Query Evaluator transforms the PTQL query into keyword-based queries and SQL queries. These are evaluated by using the RDBMS and IR engine. Inverted index is extended from the index builder to speed up the query evaluation. This has been done by indexing the sentences according to the words and the corresponding entities. PTQL queries are generated using two modes of operation. They are: training set driven query generation and pseudo-relevance feedback driven query generation.

5.1. Parse Tree Database and Inverted Index

Each document is represented as a hierarchical representation called the parse tree of a document, and collection of the parse trees of all documents forms the parse tree database.

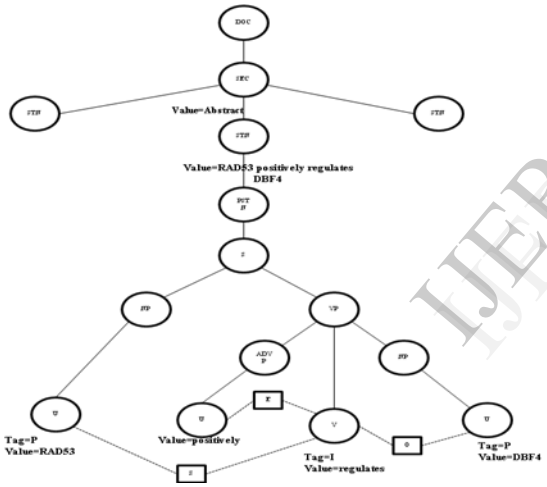

Figure 3. Parse tree for a document

Figure 3 shows an example of a parse tree. The parse tree contains the root node labelled as DOC and each node represents an element in the document which can be a section (SEC), a sentence (STN), or a parse tree for a sentence (PSTN). A node labelled as STN may have more than one child labelled with PSTN to allow the storage of multiple parse trees. The node below the PSTN node indicates the start of the parse tree. A solid line represents a parent child relationship between two nodes in the constituent tree, whereas a dotted line represents a link between two words of the sentence. In the constituent tree, nodes S, NP, VP, and ADVP stand for a sentence, a noun phrase, a verb phrase, and an adverb phrase, respectively. The linkage contains three different links: the S link connects the subject noun to the transitive verb, the O link connects the transitive verb to the direct object and the E link connects the verb-modifying adverb to the verb. The square box on a dotted line indicates the link type between two words. Each leaf node in a parse tree has value and tag attributes. The value attribute stores the text representation of a node, while the tag attribute indicates the entity type of a leaf node.

Inverted index is an essential component which is extended from the index builder also maintained by an IR engine. This inverted index enables the efficient processing of PTQL queries. Fig. 4, the index builder relies on the text pre-processor to recognize entities and replace the entities with identifiers in the sentences. The index builder relies on the text processor identifies the entities and replace those entities with identifiers in the sentences. Each sentence in the documents is indexed on its own so that each keyword-based filtering query retrieves a sentence rather than the entire document.

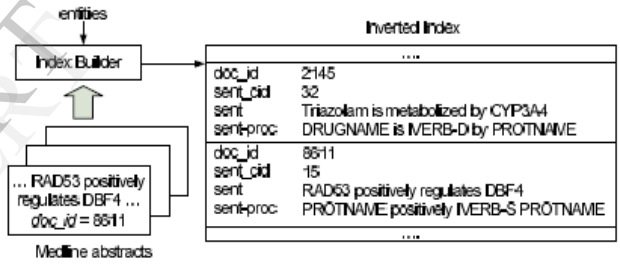

Figure 4. An extended inverted index

5.2. Parse Tree Query Language

Standard XML query languages of XPath [3] and XQuery [4] does not have the ability of express immediate following siblings and immediate-preceding siblings. This issue leads to the development of LPath [17], [18] as a query language for linguistic queries on constituent trees. However, XQuery and LPath can only express ancestor descendant and sibling relations between nodes. PTQL has the ability to express links and link types between pairs of nodes, so that PTQL can be used to express linguistic patterns based on constituent trees and links, as well as link types. PTQL is an extension of the linguistic query language LPath that allows queries to be performed not only on the constituent trees but also the syntactic links between words on linkages. A PTQL query is made up of four components:

1. Tree patterns: describes the hierarchical structure and the horizontal order between the nodes of the parse tree.
2. Link conditions: describes the linking requirements between nodes.
3. Proximity conditions: to find words that is within a specified number of words.
4. Return expression: defines what to return**.**

5.3. Query evaluation

Evaluation of PTQL queries uses IR engine as well as RDBMS. IR engine selects the sentences based on the tokens, which are defined in PTQL queries, and only the subset of sentences retrieved by the IR engine. These sentences are considered for the evaluation of the conditions specified in the PTQL queries by RDBMS. The process of the evaluation of PTQL queries as follows.

1. Translate the PTQL query into a filtering query.
2. Use the filtering query to retrieve relevant documents D and the corresponding sentences S from the inverted index.
3. Translate the PTQL query into an SQL query and instantiate the query with document id d Є D and sentence id s Є S.
4. Query PTDB using the SQL query generated in Step 3.
5. Return the results of the SQL query as the results of the PTQL query.

In step 2, the process of finding relevant sentences with respect to the given PTQL query requires the translation of the PTQL query into the corresponding filtering query. Here, the syntax of the keyword-based filtering queries.

A query term t for a filtering query is a string that can be preceded by the required operator +, as well as the term <field>:, where <field> is the name of a field. A phrase p is in the form "$t_1 \ldots t_n$," where $t_1,\ldots,t_n$ are query terms. P can be followed by a proximity operator in the form of p~<number>. A parenthesis expression is composed of query terms and phrases, enclosed by parentheses, and it can be preceded by the required operator. A keyword-based filtering query is a list of query terms, phrases, and parenthesis expressions. A PTQL query q is translated into a keyword-based filtering query using the following steps:

1. Generate query terms for each of the node expressions that are in the tree pattern of q.
2. Form phrases if consecutive node expressions are connected by "immediate following" horizontal axes.
3. Form phrases followed by the proximity operator if the corresponding nodes are defined in the proximity condition of q.

6. Query Generation

IE system must have the ability to extract high-quality results. In incremental information extraction approach PTQL queries automatically generated using two methods. They are: training set driven query generation and pseudo-relevance feedback driven query generation.

6.1. Training Set Driven Query Generation

The unlabeled document collections under a particular problem-specific database are annotated using the annotator component. This is very necessary step for precise recognition and normalization. Pattern generator identifies the phrases from the labelled data which refers the interaction to generate the patterns. These initial patterns are used to compute the conesus patterns through the pattern generator component. PTQL queries are then formed by the query generator to perform extraction from the parse tree database.

6.2. Pseudo-Relevance Feedback Driven Query Generation

Training data are not always available. At this situation the Pseudo-Relevance Feedback Driven Query Generation approach identifies the linguistic structures to generate the PTQL queries. The basic idea behind this is that, it generates the PTQL queries by considering the constituent trees of the top-k sentences retrieved with the Boolean keyword based query. This constituent tree for the retrieved relevant sentence's generated pattern is identified and interaction extraction is performed by using the PTQL queries translated from the generated extraction patterns. A boolean keyword-based query q is composed of query terms $t_1 \ldots t_n$, where a query term t_i can be a keyword, or an identifier for an entity type. With q, a ranked list of sentences S is retrieved and the constituent trees of the top-k sentences of S (denoted as S_k) are retrieved from PTDB. To find common grammatical patterns among the constituent trees of S_k, string encodings are generated for each of the sentence in S_k. A 0th level string encoding records the labels of the lowest common ancestor lca of the query terms and the query terms themselves in a pre-order tree traverse order. A

mth level string encoding is defined as the string encoding that includes at most m descendants of lca on each of the paths connecting lca and a query term t_i. If two sentences are grammatically similar also they have the same mth level string encoding. Grammatically similar sentences are grouped together to form a cluster. A PTQL query is then generated for each of the clusters of string encodings. The steps of generating PTQL queries can be outlined as follows. Let C_m be a set of clusters with mth level string encodings. Given a boolean keyword-based query q and parameter k,

1. Retrieve sentences using q from the inverted index and retrieve the constituent trees of the top-k sentences S_k from PTDB.
2. For each sentence in S_k extract the subtree that is rooted at the lca of all the query terms $t_1, \ldots, t_n$ with the query terms as leaf nodes from the constituent tree.
3. Generate mth level string encodings for each of the subtrees.
4. Sentences that are grammatically similar based on their mth level string encodings are grouped together to form clusters of common grammatical patterns C_m.
5. A PTQL query is generated for each common grammatical pattern C_m.

Interactions are extracted through the evaluation of the generated PTQL queries.

7. Conclusion

Existing extraction frameworks do not provide the capability to manage the intermediate processed data. This leads to the unnecessary reprocessing of the entire text collection when the extraction goal is modified or improved, which can be computationally expensive and time consuming one. To reduce this reprocessing time, the intermediate processed data is stored in the database as in novel framework. The database is in the form of parse tree. To extract information from this parse tree the extraction goal written by the user in natural language text is converted into PTQL and then extraction is performed on text corpus. This increment extraction approach saves much more time compared to performing extraction by first processing each sentence one-at-a time with linguistic parsers and then other components.

8. Future Enhancement

PTQL also does not provide the ability to compute statistics across multiple extractions. For future work, this has been extended to the support of other parsers by providing wrappers of other dependency parsers and scheme, such as Pro3Gres, so that they can be stored in PTDB and queried using PTQL. Also the capability of PTQL is expanded, such as the support of regular expression and the utilization of redundancy to compute confidence of the extracted information.

REFERENCES

[1] D. Ferrucci and A. Lally, "UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment," Natural Language Eng., vol. 10, nos. 3/4, pp. 327- 348, 2004.

[2] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications," Proc. 40th Ann. Meeting of the ACL, 2002.

[3] J. Clark and S. DeRose, "XML Path Language (XPath)," http://www.w3.org/TR/xpath, Nov. 1999.

[4] "XQuery 1.0: An XML Query Language," http://www.w3.org/ XML/Query, June 2001.

[5] E. Agichtein and L. Gravano, "Querying Text Databases for Efficient Information Extraction," Proc. Int'l Conf. Data Eng. (ICDE), pp. 113-124, 2003.

[6] E. Agichtein and L. Gravano, "Snowball: Extracting Relations from Large Plain-Text Collections," Proc. Fifth ACM Conf. Digital Libraries, pp. 85-94, 2000.

[7] A. Doan, J.F. Naughton, R. Ramakrishnan, A. Baid, X. Chai, F. Chen, T. Chen, E. Chu, P. DeRose, B. Gao, C. Gokhale, J. Huang, W. Shen, and B.-Q. Vuong, "Information Extraction Challenges in Managing Unstructured Data," ACM SIGMOD Record, vol. 37, no. 4, pp. 14-20, 2008.

[8] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu, "SystemT: A System for Declarative Information Extraction," ACM SIGMOD Record, vol. 37, no. 4, pp. 7-13, 2009.

[9] P.G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano, "Towards a Query Optimizer for Text-Centric Tasks," ACM Trans. Database Systems, vol. 32, no. 4, p. 21, 2007.

[10] A. Jain, A. Doan, and L. Gravano, "Optimizing SQL Queries over Text Databases," Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE '08), pp. 636-645, 2008.

[11] Y. Miyao, T. Ohta, K. Masuda, Y. Tsuruoka, K. Yoshida, T. Ninomiya, and J. Tsujii, "Semantic Retrieval for the Accurate Identification of Relational Concepts in Massive Textbases," Proc. 21st Int'l Conf. Computational Linguistics and the 44th Ann. Meeting of the Assoc. for Computational Linguistics (ACL '06), pp. 1017-1024, 2006.

[12] R. Feldman, Y. Regev, E. Hurvitz, and M. Finkelstein-Landau, "Mining the Biomedical Literature Using Semantic Analysis and Natural Language Processing

Techniques," Information Technology in Drug Discovery Today, vol. 1, no. 2, pp. 69-80, 2003.

[13] J.D. Martin, "Fast and Furious Text Mining," IEEE Data Eng. Bull., vol. 28, no. 4, pp. 11-20, 2005.

[14] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, and S. Vaithyanathan, "An Algebraic Approach to Rule-Based Information Extraction," Proc IEEE 24th Int'l Conf. Data Eng. (ICDE '08), 2008.

[15] M. Cafarella, D. Downey, S. Soderland, and O. Etzioni, "Knowitnow: Fast, Scalable Information Extraction from the Web," Proc. Conf. Human Language Technology and Empirical Methods in Natural Language Processing (HLT '05), pp. 563-570, 2005.

[16] W. Shen, A. Doan, J.F. Naughton, and R. Raghu, "Declarative Information Extraction Using Datalog with Embedded Extraction Predicates," Proc 33rd Int'l Conf. Very Large Data Bases (VLDB '07), pp. 1033-1044, 2007.

[17] S. Bird, Y. Chen, S.B. Davidson, H. Lee, and Y. Zheng, "Extending XPath to Support Linguistic Queries," Proc. Workshop Programming Language Technologies for XML (PLAN-X), 2005.

[18] S. Bird et al., "Designing and Evaluating an XPath Dialect for Linguistic Queries," Proc 22nd Int'l Conf. Data Eng. (ICDE '06), 2006.