A Novel Approach to Building Relationships Between Annotations in Pipeline Natural Language Processing Systems

Balaji R. Soundrarajan[†⋆] BALAJI@CS.UTAH.EDU
Thomas Ginter[†⋆] THOMAS.GINTER@UTAH.EDU
Scott DuVall[†⋆] SCOTT.DUVALL@HSC.UTAH.EDU
University of Utah[†]
Veterens Affairs Health Care System, Salt Lake City[⋆]

Abstract

This study demonstrates a novel approach to building relationships between existing annotations using functionality of the Annotation Librarian, a tool set that aids in the development of natural language processing application in the Apache Unstructured Information Management Architecture(UIMA). This feature allows a module to use annotations created in earlier steps in addition to document text to capture patterns, which may otherwise be difficult to using regular expressions or rule-based approaches. The syntax for the tool mirrors the Java Pattern and Matcher classes of the Regex package. This study compares the Annotation Librarian with contemporary methodologies that are used to perform similar functionsand addresses issues of performance, simplicity, extended capability and accuracy.

1. Introduction

Much work has been done to explore the various methods of extracting information from unstructured text. While machine learning methods are widely used in information extraction (IE) systems and have some marked advantages, many systems still rely, at least in part, on the simplicity and power of regular expressions, rules, and curated lexicons (Garvin et al., 2011; McCrae & Collier, 2008; Frenz; Chapman et al., 2001). Common among systems developed using frameworks such as the Apache Unstructured Information Management Architecture(UIMA)[1], system development is split into functional modules and text is processed in subsequent modules in a pipeline fashion. These pipeline IE systems allow the results from one module to be used as input for later modules. Downstream modules can then build upon annotations made in earlier modules and perform more complex tasks. Breaking an IE system into smallsteps that are coupled together helps organize tasks into reusable pieces and simplifies overall system development.

In true pipeline fashion, IE systems often include tasks for pre-processing, lexical and syntax parsing, and concept extraction or semantic class recognition that build on each other. Most projects require additional steps that help describe the context in which concepts were found or relationships between different concepts in the text. NegEx (Chapman et al., 2001) is an example of an algorithm that provides evidence for a concept being present or absent, such as whether a person has a disease that is mentioned in a clinical report, by identifying words before and after a concept that suggest the concept is not present. For example, "the patient has no fever" is an indication that the concept 'fever' is not present in the patient. Other methods for understanding the context around concepts include the use of an inclusion and exclusion list (Akbar et al., 2009), temporal locality search (Grouin et al.), window search (Li et al., 2009), and combinations of the above techniques (Hamon & Grabar, 2009). While these techniques typically use an existing annotation as an anchor point to assist in context classification, they also perform much of the task on the raw text itself rather than focusing on relational patterns between previously annotated concepts.

The Annotation Librarian allows patterns to be built

Appearing in *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

[1]Apache UIMA is available from http://uima.apache.org/

from a combination of text and annotation, simplifying individual tasks and building on previous annotations. The matching patterns are also saved as annotations and can therefore be used as input to downstream modules. Incorporating existing annotations into patterns increases capability and reduces complexity of search patterns, which can reduce errors.

In this study we describe our novel approach to building relationships between annotations with other algorithmic approaches. An applied project comparison was also performed in which the same functionality implemented using both regular expressions and with the Annotation Librarian.

2. Brief Overview of Previous Approaches

As mentioned above, the NegEx algorithm (Chapman et al., 2001) identifies which concepts mentioned in clinical text may not be present in a patient. The baseline algorithm negates all concepts from the occurrence of a negation word until the end of the sentence. NegEx is designed to differ between two basic negation types and catch double negatives. For example, "The case of cancer is not ruled out" is identified as a positive case by the NegEx algorithm, whereas "No Tobacco or Alcohol" will be identified as a negative case. This method works well on sentences that express assertions for a single concept but has difficulty with short phrases that are part of lists or template text and sentences with multiple concepts and assertions.

The Inclusion and Exclusion List (Akbar et al., 2009) method filters annotations previously made using explicit inclusion and exclusion lists. Terms on the exclusion list are removed after annotation and those on the inclusion list are added. Inclusion and exclusion lists can also be dynamically modified or generated using training and testing data sets within a supervised learning algorithm. Akbar used inclusion and exclusion lists for determining whether a medication was not taken by a patient and whether the patient was allergic to a medication. These lists may be iteratively created during training and can then be added to an exclusion list to filter out data in the testing set. One attractive feature of this method is that it uses follows a highly structured, iterative approach to the development of a grammar used in concept classification.

Another method implements a rule-based system, which works on the principle of Temporal Locality Search (Grouin et al.). Terms corresponding to different concept types are grouped into lexicons. A 'Restricted Lexicon' and a 'Larger Lexicon'. These term lists are then used to recognize patterns found in the text. Finally the algorithm looks for relationships based on lexical information in the document and a set of rules implemented by regular expressions.

The Window Search (Li et al., 2009) method uses a conditional random field model to identify concept terms and related information. This step did not take into account possible relationship information. A second model was then built that used the medication name as the anchor concept and considered the distance between the concepts (tokens and lines), the relative order, and the type of each concept.

Perhaps the most similar to our approach is the OpenDMAP (Hunter et al., 2008) system. OpenDMAP is a concept analysis system written primarily to extract microbiology assertions from MEDLINE extracts. The system is designed as a hierarchical rules engine with basic patterns forming the base rules such as the following:

VEHICLE := bicycle, car, bus, train;
TRANSPORT := ride, pedal, drive;

UIMA annotation types can also be assigned to a basic rule definition. Base rules can then be combined utilizing a rigid systematic syntax to form concept definitions such as:

COMMUTER-RELATION :=
[PERSON][TRANSPORT](the)?[VEHICLE]
(fromdet?[LOCATION])?(todet?[LOCATION])?;

There is also some support for variations in the order of the phrases that make up the rules pattern. This approach is similar to ours but not generally transferable outside of genetic domain extraction. This is because it is primarily a hierarchical representation of a ruleset specific to the genetic domain. For example, concepts annotated by other UIMA engines are only used if specialized rules can be produced to translate those annotation instances into a concept that matches the ruleset. The translation of which depends on custom code which must be modified to handle new instance types. Depending on how they have implemented the "pattern" engine however it might be modifiable to be more transferable but not in the current implementation.

Variations of each of these methods have been used in combination as well (Hamon & Grabar, 2009). Understanding the context in which a concept is used and the relationships to surrounding information is an important part of many IE projects.

3. Annotation Librarian

The simplest approach to building relationships between annotations is to do it all in one step. This involves creating regular expressions that find all concepts and relationships (Garvin et al., 2011). This approach generates very long and elaborate regular expressions and can be difficult to debug, maintain, and incorporate new patterns. This one-step process cannot take advantage of the building block process afforded by pipeline systems. While many closed lexical classes can be identified using regular expressions, the context algorithms defined above are preferred because they allow dictionary mapping, machine learning, and iterative extraction of concepts not possible with regular expressions. The context algorithms, in turn, are limited in the way they incorporate existing annotations, usually requiring one annotation to act as an anchor around which a window is searched for patterns or terms or requiring a strict limited syntax for the combination of rules from which previous annotations might be included. The Annotation Librarian provides tools that allow concepts to be discovered by any means and still be utilized in defining relationships between patterns.

Figure 1 below illustrates a sentence of clinical text that has been processed by several IE modules. The document text is shown in black and the annotations are displayed as layers associated with the text. In a pipeline IE system, each layer would represent a different functional module - sentence splitting, tokenization, part-of-speech tagging, phrase chunking, semantic class recognition, and concept mapping. In this visualization, the word cancer has been identified as a disease semantic class and has been mapped to the code CA001. The patient has been identified as a person semantic class and has been assigned the identifier P12340.

While the visualization divides the output of each functional module into disjoint layers, many applications also limit input to a single layer. Phrase chunking, for example, uses only assigned parts-of-speech and does not require any knowledge of tokenization, sentence splitting, or the actual text. Operating from a flat perspective does tend towards simple, modular algorithms but also limits the forms of contextual patterns that can be identified. Figure 2 illustrates a regular expression pattern that could be as simple as:

The\s+patient\s+had\s+cancer\s+\.

The Annotation Librarian allows an IE system developer to create patterns that span all layers of annotations. This capability makes it possible for the system to utilize specialized algorithms for classification of unique data types without having to build those algorithms into regular expressions or other complicated algorithms for contextual classification. Instead relatively simple patterns can be created increasing the accuracy of the desired pattern. The ability to capture patterns like:

<token/> \s+<NOUN/> \s+<VERB/>
\s+<disease/>

is illustrated in Figure 3

Immediately, the advantage of separating the pattern away from the text becomes clear. If nothing else, this pattern allows any disease condition to be substituted for the word cancer - opening up phrases like:

The patient had diabetes

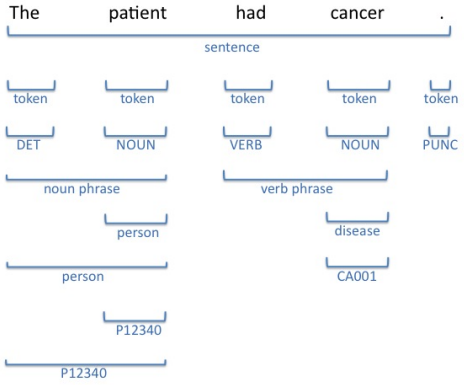

Figure 1.

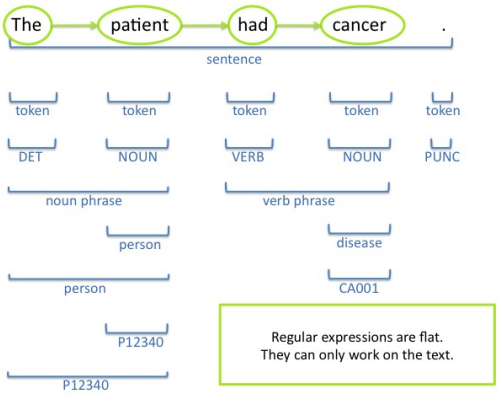

Figure 2.

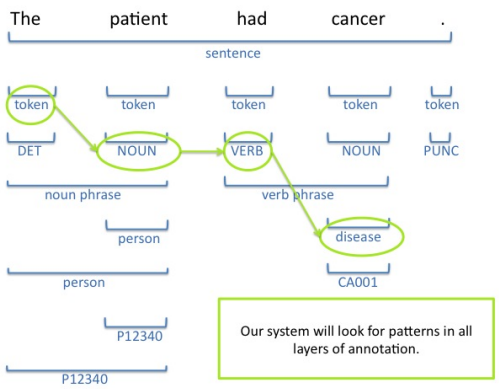

Figure 3.

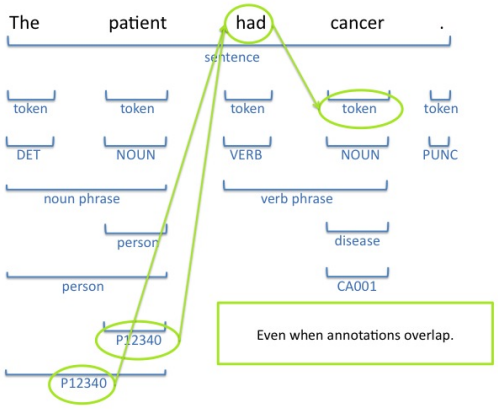

Figure 4.

and

The patient had arthritis

The full pattern can also catch phrases like:

His brother had diabetes

and

The doctor diagnosed arthritis

The method utilized by the Annotation Librarian also accounts for the overlap of annotations as illustrated by Figure 4.

The Annotation Librarian was built to take advantage of the annotation structure in UIMA. It interacts with the Common Analysis Structure using the UIMA API to view annotation information. All annotations are associated with spans in the text, so the text covered by an annotation or the raw document text are also available.

The Annotation Librarian methods mirror the functionality available in Java. In place of the Pattern class, an AnnotationPattern class is implemented that parses regular expressions containing annotation classes. Annotation classes are identified using XML syntax (<disease />) and can contain information about the type of content the annotation may contain (<disease getID="CA001" />).

4. Comparison

A health services use case wherein positive instances of Ejection Fraction data was extracted was used for comparison of the performance and accuracy of an annotator implemented using regular expressions and one implemented using the AnnotationPattern feature of the Annotation Librarian.

Figure 5 below illustrates a pattern wherein a Concept annotation is followed by a preposition and then by a Value annotation.

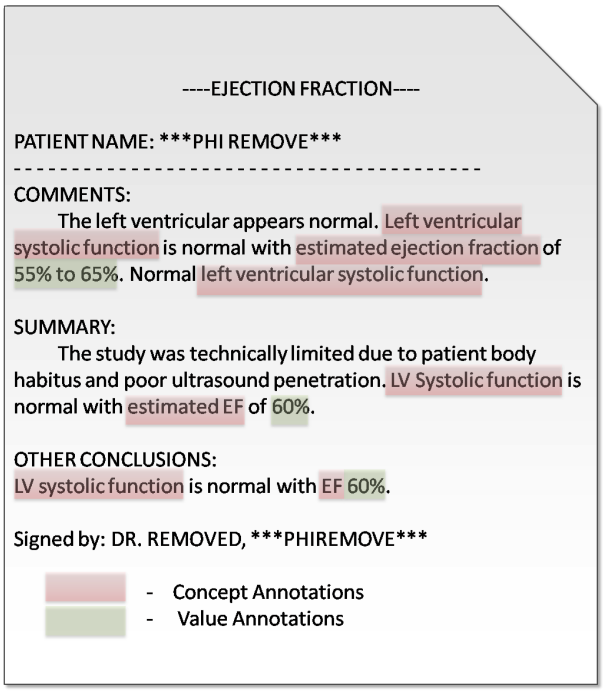

Figure 5.

The original project implementation utilized complex regular expressions to identify Concept and Value annotations individually within the report text. It then created a temporary comparison String that replaced each character within each Concept annotation with

a unique character (α). Then each character in each Value annotation was replaced with a different unique character (δ). Such characters were then included in much larger patterns used to search for a Concept code character followed by a very large regular expression pattern used to identify prepositions, then followed by Value code characters. In this way the regular expression syntax provided functionality similar to the AnnotationPattern Feature. The resulting patterns were used to create Ejection Fraction annotations as shown in Figure 6 below.

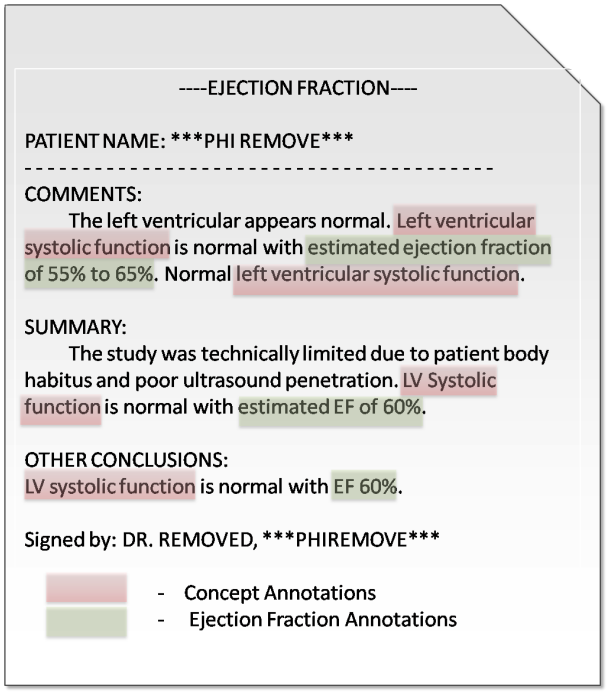

Figure 6.

The Ejection Fraction project was then implemented using the AnnotationPattern feature. Individual Concept and Value annotations were created in separate annotators. The AnnotationPattern feature was then utilized to match a pattern such as the following:

<Concept getConceptName="Ejection Fraction" />string regex representing propositional phrases<Value />

The pattern above matches a Concept class annotation whose ConceptName feature is "Ejection Fraction". This pattern then includes the regular expression strings representing the variety of prepositional phrases included in the text and finally followed by a Value annotation class object.

5. Results

The table 1 summarizes the relative performance of the annotators based on execution time on a corpus of 1,185 documents as well as lines of code required for implementation.

Table 1. Performance

EF Documents	Time Taken (ms)	Lines of Code
String Regex	80941	150
AnnotationPattern	136552	25

Performance From the statistics above, we can see the AnnotationPattern takes more execution time than performing the same operation utilizing the Java Pattern[2] class for regular expressions. To be precise, the AnnotationPattern feature is 41% slower. However the lines of code (LOC)were reduced by 83.3%, which illustrates the increased simplicity of the AnnotationPattern.

Simplicity The syntax of the feature is designed in such a way that it can handle combinations of string regular expressions and annotation metadata easily. The syntax also resembles the Java Matcher[3] and Pattern regular expression syntax facilitating adoption by Java developers. Creating contextual classifications of regex and annotation utilizing traditional regular expressions requires a developer to create very complex patterns for all but the simplest of cases. Such complex patterns make the code more difficult to maintain and increases the likelihood of an error within the pattern. The AnnotationPattern feature allows for much simpler and smaller patterns that are easier to maintain and less likely to contain errors.

6. Discussion

In the NegEx algorithm or the Inclusion/ Exclusion List method, the initial preprocessing steps of the pattern search are performed on the raw unstructured information by omitting the text which of least or no interest. Though these methods employee intense scrutiny, these methods are effectively made to perform two tasks. First task is to prune the document of all patterns, which cannot fall into the candidate list.

[2]Documented at http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html

[3]Documented at http://download.oracle.com/javase/6/docs/api/java/util/regex/Matcher.html

Second task is to select the matching patterns from the candidate patterns. Annotation Pattern bypasses the above overhead by designing search patterns that encompasses both the core search entity and its environment.

The Annotation Pattern, since it allows a combination of both annotations and regular expressions, it becomes intuitively easy to create patterns which establish a relationship among Medical terms, Dosage, Medication, the environment of the discussion and other related entities. This approach over comes the difficulty in the Window Search algorithm where we first use a machine to find the different entities, such as Medical terms, Medication followed by another system, which is used to form relationship between medication fields. Finally,a third system to determine the matching patterns.

Apart from the above advantages, the lines of code to form the engine is also small, which is a property resulting from the ability to create complex patterns using simple syntax, where the complexity is handled by the previous pipelines during the metadata creation.

Acknowledgments

This work was supported using resources and facilities at the VA Salt Lake City Health Care System with funding support from the VA Informatics and Computing Infrastructure (VINCI), VA HSR HIR 08-204; and the Consortium for Healthcare Informatics Research (CHIR), VA HSR HIR 08-374. Views expressed are those of authors and not necessarily those of the Department of Veterans Affairs or affiliated institutions.

References

Akbar, Saiful, BroxRst, Thomas, Slaughter, Laura, and Nytr, ystein. Extracting medication information from patient discharge summaries. 2009.

Chapman, Wendy Webber, Bridewell, Will, Hanbury, Paul, Cooper, Gregory F., and Buchanan, Bruce G. A simple algorithm for identifying negated findings and diseases in discharge summaries. *Journal of Biomedical Informatics*, 34:301–310, 2001. doi: 10.1006/jbin.2001.1029.

Frenz, Christopher M. Deafness mutation mining using regular expression based pattern matching.

Garvin, Jennifer H., South, Brett R., Bolton, Dan, Shen, Shuying, DuVall, Scott L., Bray, Bruce, Heidenreich, Paul, Samore, Matthew H., and Goldstein, Mary K. Automated extraction of ejection fraction (ef) for heart failure (hf) from va echocardiogram reports. 2011.

Grouin, Cyril, Delger, Louise, and Zweigenbaum, Pierre. A simple rule-based medication extraction system.

Hamon, Thierry and Grabar, Natalia. Linguistic approach for identification of medication names and related information in clinical narratives. 2009.

Hunter, Lawrence, Lu, Zhiyong, Firby, James, Baumgartner, William, Johnson, Helen, Ogren, Philip, and Cohen, K Bretonnel. Opendmap: An open source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression. *BMC Bioinformatics*, 9(1): 78, 2008. doi: 10.1186/1471-2105-9-78. URL http://www.biomedcentral.com/1471-2105/9/78.

Li, Zuofeng, Cao, Yonggang, Antieau, Lamont, Agarwal, Shashank, Zhang, Qing, and Yu, Hong. Extracting medication information from patient discharge summaries. 2009.

McCrae, John and Collier, Nigel. Synonym set extraction from the biomedical literature by lexical pattern discovery. *BMC Bioinformatics*, 9(1):159, 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-159. URL http://www.biomedcentral.com/1471-2105/9/159.