Integrating Medical Text Extraction Tools

by

Silvia Baptista

Department of Electrical Engineering and Computer Science

Massachusetts Institute of Technology

January 31, 2008

Abstract

This report describes the steps taken to integrate the components of Category and Relationship Extractor (CaRE) into the General Architecture for Text Engineering (GATE). A Link Grammar Interface was developed and its output added as annotations into GATE's system. Then GATE's tokenizer, sentence splitter, part-of-speech tagger and noun phrase chunker tools were modified to resemble CaRE's preprocessing components. Next CaRE's Map and Mesh programs were adjusted to work with GATE. Finally, a Support Vector Machine (SVM) application was created using the features described above.

1. Introduction

The information found in medical discharge papers is important for doctors and researchers. Doctors providing care for their patients need to know what symptoms the patient has, allergies, drugs taken, test run, etc. Researchers trying to find statistics on patients with X and Y disease or symptom need a more efficient way to find all these records. The problem is that this information is not readily available. If the information is digitized and formatted appropriately it can then be easily queried and prevent medical errors caused by miscommunication.

The project described here takes another step forward towards accomplishing this task. Many programs have been developed to help with this problem. Tawanda Sibanda developed the Category and Relationship Extractor (CaRE) [1], which finds semantic categories and their relationships in medical discharge papers. The project integrates the tools developed by Sibanda so that they can work together seamlessly. These tools include preprocessing programs for Assertion Classification, De-identification, Concept Recognition, Relationship Extraction, along with their Support Vector Machine (SVM) training and predicting program. These tools were incorporated into the General Architecture for Text Engineering (GATE) [2], a free software framework available online.

The rest of this report is organized as follows: Section 2 discusses GATE and why it was chosen as a framework for this project. Section 3 discusses the Link Grammar parser and its pivotal role in identifying concepts. Section 4 describes the modified GATE plugins used to preprocess the documents. Section 5 explains the use of the UMLS based programs, Map and Mesh. Section 6 illustrates the SVM capabilities of GATE. Sections 7 and 8 cover the results and conclusions.

2. GATE

The General Architecture for Text Engineering (GATE) is a free software available online, widely used for creating text mining projects and for natural language processing (NLP). GATE is made up of many plugins that can be put together in a pipeline to form an application. These plugins can be used as is or modified to fit particular needs of a project. Since CaRE is made up of many different components, GATE makes it easy to mix and match plugins to find the application that works best. GATE also has a built-in "Annotation Diff" tool to measure precision and recall. GATE is versatile; it works on many different operating systems and can process several other languages besides English.

3. Link Grammar Parser Gate Interface

The Link Grammar Parser [3] is an open source tool that parses English text and extracts syntactic dependencies by labeling the relationship between pairs of words. The Link Grammar Parser is written in C code. In order to use it with the GATE environment, it had to be wrapped to permit its interoperation with Java code. Fortunately, there already exists a Java Native Code Link Grammar Interface available online, developed by Chris Jordan. The Link Grammar Interface uses Java Native Interface (JNI) to call the original programs written in C. There is a 32-bit and a 64-bit version of the Link Grammar Interface so that it can be run on multiple platforms and not be limited to Linux, as was CaRE.

CaRE's interface to the Link Grammar parser is found in a program called findlink2, also written in C code. Jordan's interface does not produce the same output as the findlink2 program so Jordan's work was modified to get the required results. The changes made included adding a "panic mode" and translating more of the code in the findlink2 program into Java. The "panic mode" is used to parse long sentences when a valid parse is not found within a certain time.

GATE's bootstrap wizard was used to create a plugin of the Link Grammar Interface, called FindLink. The FindLink plugin takes as input a tokenized file and produces a file with the link structure for each sentence. The input file is tokenized using GATE's tokenizer (described below) in order to bind the syntactic bigrams to each token. Tokenization is not necessary if the user is only parsing the file.

FindLink extracts the left and right syntactic bigrams for each token and adds them as features to GATE's token annotation set, to be used later by a support vector machine.

As described in Sibanda's thesis, a syntactic bigram is "the right-hand links originating from the target; the words linked to the target through single right-hand links (call this set R1); the right-hand links originating from the words in R1; the words connected to the target through two right-hand links; the left-hand links originating from the target; the words linked to the target though single left-hand links (call this set L1); the left-hand links originating from the words in L1; and the words linked to the target through two left-hand links."

In the CaRE system, a separate program written in the Perl programming language extracts the syntactic bigrams. CaRE's FindLink component also creates intermediate files to be passed around to different programs, which can slow down the process of finding links. In the FindLink plugin created for GATE, the bigram extraction is accomplished within the same class with no intermediate files. This produces a cohesive program that is easier to decode and transform if changes need to be made in the future.

4. GATE's Preprocessing Resources

The first step in preprocessing the documents is tokenization. The CaRE system has a very simple tokenizer that tokenizes by white space. GATE's tokenizer is more complicated because not only does it split the document into tokens, it also extracts a multitude of information for each token. It places the tokens in different categories such as punctuation or word and it stores the length of the token and orthographical information such a capitalization. The tokenizer was modified to recognize decimal numbers, hyphenated-words, and dates as one token instead of multiple tokens. Having all this information already stored in each token eliminated the need to create a separate regular expression program to extract these features. All this information was later used to build the SVM model.

Many tools work on one sentence at a time, therefore it is important to split the input into individual sentences. It does not suffice to simply split the text after a period or line break as GATE's ANNIE Sentence Splitter does. For example, sentences that use a period in abbreviations, personal titles, or numbers can be incorrectly split. GATE's RegEx Sentence Splitter handles these cases correctly. RegEx Sentence Splitter's default configuration file splits sentence after two or more new line characters, and was modified to split after one or more new line characters.

One of the tools that require the sentence splitter is the part-of-speech (POS) tagger. The tagger behaves like the tagger used in CaRE once given the same lexicon and rule files. Like the other tools, the tagger stores its result as a feature of each token.

GATE's Gazetteer tool is used to replace CaRE's dictionary lookup. CaRE's name, hospital, and location dictionaries were added to GATE's already extensive list of dictionaries. When an entry in the input document is found in one of the dictionaries, an annotation type of "Lookup" is added to the document. Each Lookup annotation also holds a "majorType" feature, which equals the entry type such as name, hospital, or location.

Another step required to preprocess a document is to mark the noun phrases. The CaRE system uses the chunker produced by Ramshaw and Marcus [4] with a few modifications to fix the misclassification of dates and numerals such as phone numbers. GATE already has a Noun Phrase Chunker plugin, but it also had problems with dates and numerals. I used Java Annotation Patterns Engine (JAPE) rules to make the plugin handle such situations correctly. JAPE is a language developed to recognize patterns within the annotations of a document and produce new annotations out of the patterns.

Gate's Chunker was modified to output a file similar to the one produced by CaRE's chunker. This was necessary, because the chunker's output is used in the Map and Mesh programs, and these programs were written to process a file with very specific attributes. For example, below is a sentence Map and Mesh would use as input, where the words enclosed in brackets are noun phrases.

> [Mr. Smith] [' s] elevated [SGOT levels] indicate [an increased probability] of [myocardial infarction]

GATE's chunker, however, produces the following:

```
<NounChunk gate:gateId="50">Mr. Smith</NounChunk> <NounChunk
gate:gateId="51">' s</NounChunk> elevated <NounChunk
gate:gateId="52">SGOT levels</NounChunk> indicate <NounChunk
gate:gateId="53">an increased probability</NounChunk> of <NounChunk
gate:gateId="54">myocardial infarction</NounChunk>
```

Below is an example of the chunker's modified output, which is used in the Map and Mesh programs, which were also customized to look for the NounChunk XML tags instead of the brackets.

```
<NounChunk> Mr. Smith </NounChunk> <NounChunk> ' s </NounChunk>
elevated  <NounChunk> SGOT levels </NounChunk>  indicate  <NounChunk>
an increased probability </NounChunk>  of  <NounChunk> myocardial infarction
</NounChunk>
```

5. Map and Mesh

CaRE uses concepts from the Unified Medical Language System (UMLS), provided by the National Library of Medicine. UMLS helps developers create computer systems that can “understand” the meaning of medical terms. It contains a Metathesauraus that links concepts to its synonym, identifies relationships between concepts, and assigns concepts to a semantic type. The Map program retrieves the UMLS semantic type of each word in a document. Mesh analyzes each noun phrase and retrieves the MeSH (Medical Subject Heading) ID, a descriptor for each medical term appearing in the input text.

Both programs parse the document produced by the Noun Phrase Chunker. To make the tools work seamlessly, GATE’s datastore system was used to store the chunked, mapped, and meshed files. Also, the UMLS type and MeSH ID were embedded in the document and set as a feature in the token annotation type. This way other plugins will be able to quickly find and use these annotations using gate’s existing framework.

6. Support Vector Machine

One of the most complicated components of the CaRE system is the Machine Learning component. It uses Support Vector Machines [5] to perform de-identification, concept recognition, and relationship extraction. The task of the de-identification SVM is to find and label the personal health information (PHI) in a document that belong to one of the following classes: doctor, patient, hospital, ID, location, date, and phone. The steps required to create an SVM in GATE are described below:

1. Manually annotate a document with the classes you want the SVM to learn. In the de-identification case, the classes are described above. This file will be used to train the SVM. Loading the document onto GATE, highlighting a text to be

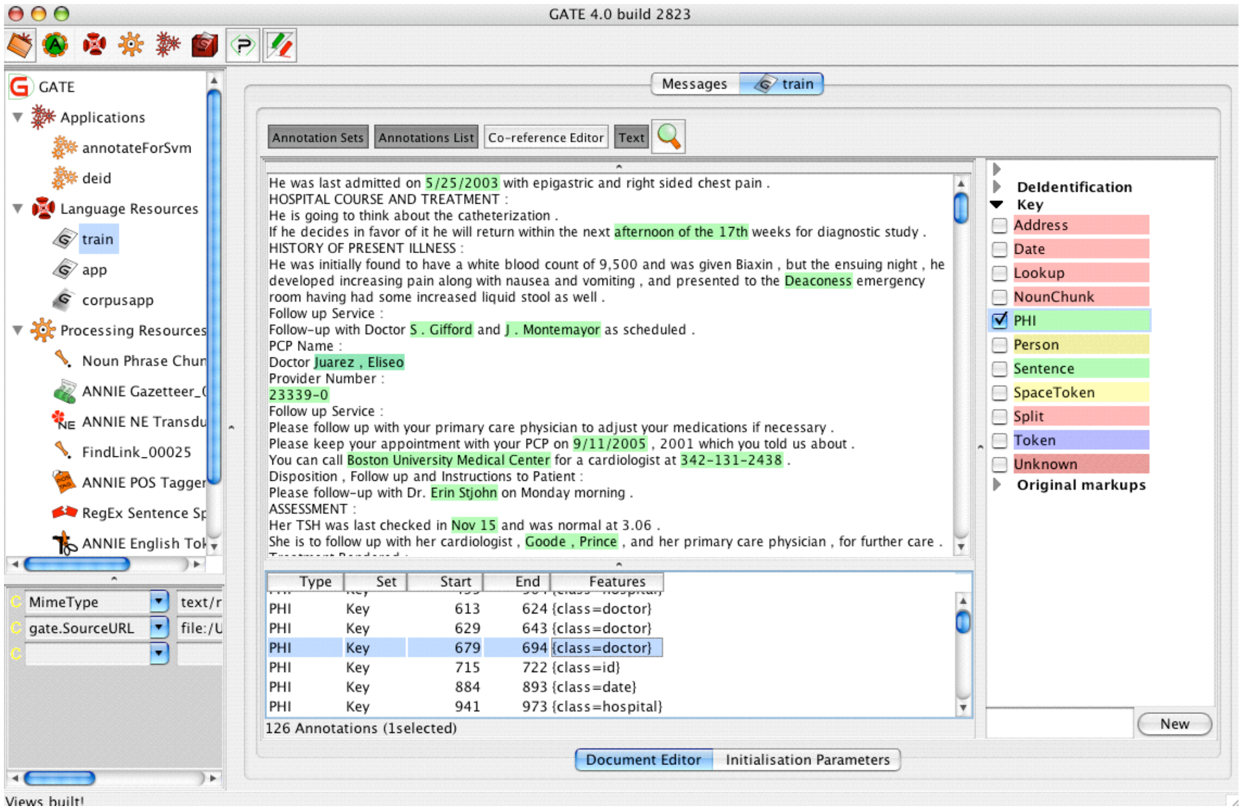
annotated and entering the annotation details in the popup box, can do this. (See Figure 1).

2. Pick the features you want to add to the SVM and create a configuration file based on these features. (See Figure 2). For complete instructions on how to create the configuration file refer to the GATE manual.
3. Run the necessary tools on the manually annotated document to extract the features from step 2.
4. Train the SVM with the resulting file from step 3.

To apply the learned SVM model to a new document simply run the same tools from step 3 above on the new document and run the SVM on the resulting document, with the learning mode set to application.

The SVM uses features of each word to try and figure out how to categorize the word. The features used for de-identification are the target word, the two strings before and after the target word, length, left and right syntactic bigrams, MeSH ID, part-of-speech tag, whether or not the word is capitalized, contains punctuations or numbers, and whether or not it is found in one of the dictionaries. This information is found by examining the annotation features stored in each token through the preprocessing tools described above.

Manually annotating a document for training the SVM is very time consuming. Given more training files for training, the SVM will be more accurate. Due to lack of time, only the de-identification SVM was created. The assertion classifier and relation extraction SVMs can be created similarly. If the user is not satisfied with the results of the SVM, one can manually delete the incorrect annotations and insert new annotations. Then the corrected output can be saved and added to the training corpus.



He was last admitted on 5/25/2003 with epigastric and right sided chest pain .
HOSPITAL COURSE AND TREATMENT :
He is going to think about the catheterization .
If he decides in favor of it he will return within the next afternoon of the 17th weeks for diagnostic study .
HISTORY OF PRESENT ILLNESS :
He was initially found to have a white blood count of 9,500 and was given Biaxin , but the ensuing night , he developed increasing pain along with nausea and vomiting , and presented to the Deaconess emergency room having had some increased liquid stool as well .
Follow up Service :
Follow-up with Doctor S . Gifford and J . Montemayor as scheduled .
PCP Name :
Doctor Juarez , Eliseo
Provider Number :
23339-0
Follow up Service :
Please follow up with your primary care physician to adjust your medications if necessary .
Please keep your appointment with your PCP on 9/11/2005 , 2001 which you told us about .
You can call Boston University Medical Center for a cardiologist at 342-131-2438 .
Disposition , Follow up and Instructions to Patient :
Please follow-up with Dr. Erin Stjohn on Monday morning .
ASSESSMENT :
Her TSH was last checked in Nov 15 and was normal at 3.06 .
She is to follow up with her cardiologist , Goode , Prince , and her primary care physician , for further care .

Type	Set	Start	End	Features
PHI	Key	613	624	{class=doctor}
PHI	Key	629	643	{class=doctor}
PHI	Key	679	694	{class=doctor}
PHI	Key	715	722	{class=id}
PHI	Key	884	893	{class=date}
PHI	Key	941	973	{class=hospital}

126 Annotations (1selected)

Figure 1. A document with PHI highlighted in GATE

Each feature is enclosed inside an attribute or attributelist xml tags. Name holds the feature name chosen by the user. Semtype holds the type of the attribute value (only the Nominal is supported for now). Type is the annotation type to be analyzed. Feature is a sub-element of the annotation type where the actual SVM feature is held. Position and range correspond to the position relative to the given word.
The features described below are the target word and the words within a +/-2 context window of the target word, the left syntactic link of the target word (retrieved from the link parser and saved inside the token annotation), the part of speech tag of the target word and the words within a +/-2 context window of the target word (saved in the token annotation under category), the MeSH ID of the target word (saved inside the token annotation under mesh from the Mesh program), and the length of the target word.

```
<ATTRIBUTELIST>
        <NAME>LexicalBigrams</NAME>
        <SEMTYPE>NOMINAL</SEMTYPE>
        <TYPE>Token</TYPE>
        <FEATURE>string</FEATURE>
        <RANGE from="-2" to="2"/>
</ATTRIBUTELIST>

<ATTRIBUTE>
        <NAME>LeftLink1</NAME>
        <SEMTYPE>NOMINAL</SEMTYPE>
        <TYPE>Token</TYPE>
        <FEATURE>leftlink1</FEATURE>
        <POSITION>0</POSITION>
</ATTRIBUTE>

        <ATTRIBUTELIST>
        <NAME>POS</NAME>
        <SEMTYPE>NOMINAL</SEMTYPE>
        <TYPE>Token</TYPE>
        <FEATURE>category</FEATURE>
        <RANGE from="-2" to="2"/>
</ATTRIBUTELIST>

        <ATTRIBUTE>
        <NAME>MeshID</NAME>
        <SEMTYPE>NOMINAL</SEMTYPE>
        <TYPE>Token</TYPE>
        <FEATURE>mesh</FEATURE>
        <POSITION>0</POSITION>
</ATTRIBUTE>

<ATTRIBUTE>
        <NAME>Length</NAME>
        <SEMTYPE>NOMINAL</SEMTYPE>
        <TYPE>Token</TYPE>
        <FEATURE>length</FEATURE>
        <POSITION>0</POSITION>
</ATTRIBUTE>
```

Figure 2. Explanation of some of the features in the de-identification SVM

7. Results

The goal of this project was to reproduce CaRE's functionality within GATE. We were successful in reaching this goal. All the preprocessing components present in CaRE as well as the de-identification SVM are now available in GATE. The modularized design of the application also allows the user to customize the program with the interchangeable components (See Figure 3).

We tried to run parallel tests to compare the SVM results of the GATE program to CaRE's results. The GATE's SVM was trained with a subset of the medical discharge papers found with the CaRE system. The sizes of test files were 10KB (test1), 8KB (test2), and 8KB(test3). With respect to running time, CaRE took about 6 minutes, 5 minutes, and 30 seconds for test1, test2, and test3, while GATE took 8 minutes, 3 minutes, and 2.5 minutes. On a 100KB file CaRE ran for about 10 minutes, but GATE took almost 1.5 hours. CaRE retrieved 475 PHI, and GATE retrieved 332. We were unable to draw a conclusion based on the running time.

On test1, CaRE found 32 PHI, while GATE found 60. GATE missed 1 PHI found by CaRE, but also found 25 patients and 4 dates that CaRE missed. On test2 CaRE found 13 PHI, while GATE found 43. GATE missed 3 PHI found by CaRE, but also found the following PHI that CaRE missed: 30 patients, 2 dates, and 1 doctor. On test3 CaRE found 16 PHI, while GATE found 55. Gate missed 1 PHI found by CaRE, but also found 1 hospital, 1 date, and 39 patients that CaRE missed.

In the tests mentioned above, CaRE did not find any patient PHI. The reason for such a disparity in the results is due to the use of different training files in the two applications. GATE's SVM was trained with different files because we were unable to determine the training files used for CaRE's SVM. The two programs should have similar results once given the same training files.

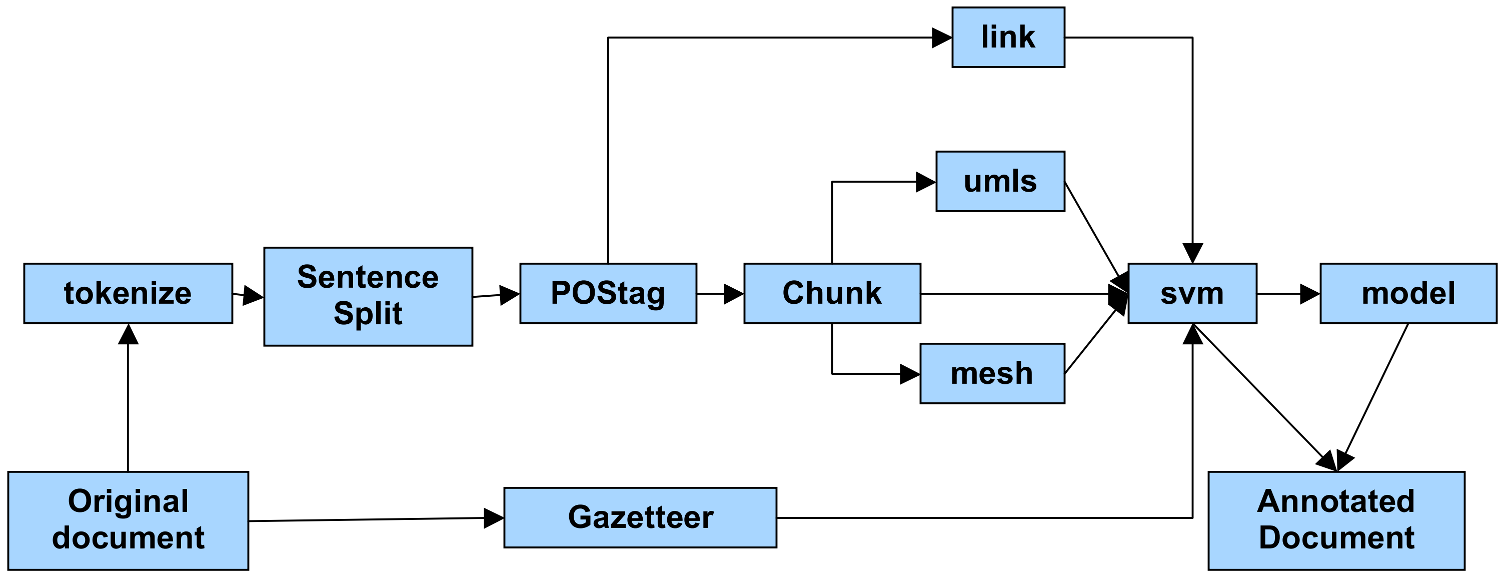


Figure 3. Flow diagram of GATE's version of CaRE

8. Conclusion

This report described the steps taken to integrate the CaRE system into GATE. First, a JAVA interface to the Link Parser Grammar was created. Then some of GATE's preprocessing tools, such as the tokenizer and noun phrase chunker, were modified to resemble CaRE's preprocessing components. The next step was to integrate CaRE's Map and Mesh programs to interoperate with GATE's annotations. Finally, the annotations created by these tools were used as features for a SVM to find personal health information in medical discharge papers.

When compared to CaRE, SVM feature extraction is easier in GATE. CaRE uses custom Perl files to extract the features. GATE stores the features in different annotation types, and the user simply tells the SVM where to look. However, the GATE application's running time was poor for large files. It was found that the slowest components of this application were the FindLink, Map, and Mesh programs, and it is suggested that future work be done to make them faster.

References

1. Sibanda, Tawanda. Was the Patient Cured? Understanding Semantic Categories and Their Relationships in Patient Records. Massachusetts Institute of Technology, June 2006.
2. H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). Philadelphia, July 2002.
3. D. Sleator and D. Temperley. Parsing English with a link grammar. Technical Report CMU-CS-91-196, Carnegie Mellon University, 1991.
4. L. Ramshaw and M. Marcus. Text chunking using transformation-based learning. Third ACL Workshop on Very Large Corpora, 1995.
5. Chih-Chung Chang and Chih-Jen Lin, LIBSVM: a library for support vector machines, 2001.