

Information Extraction for MUC-3 Terrorism Domain Corpus

Siddharth Ramesh Aaron Lockhart
School of Computing, University of Utah,
Salt Lake City, UT 84112.
{sramesh,lockhart }@cs.utah.edu

1 Introduction

Our class was given the task of creating an information extraction program given the texts for the MUC-3 terrorism domain. Each news story that we got, described exactly one terrorist event. Our Information Extraction System had to go through the news article and produce a template which looked like the following:

ID: News story identifier (e.g., DEV-MUC3-0029)
INCIDENT: Type of event. One of arson, attack, kidnapping, bombing, robbery
WEAPON: Weapons used in the event
PERPINDIV: Individuals who perpetrated the event
PERPORG: Organization believed to be responsible for that event
TARGET: Physical targets of the event
VICTIM: Victim of the event

2 Organization and various Modules

Our system is organized into two parts. The first is a training program that produces the caseframe files. We developed this system by extracting the answers from the development set answer keys and looking up the corresponding sentences in the text files. Each sentence that was identified was then tagged and parsed. After parsing, the sentence was sent to a clause segmentation unit that identifies the subjects, direct objects and voice of verbs. We then applied a heuristics file to develop caseframes for that particular sentence. For example a given sentence might be:

```
MEANWHILE,    FMLN    FORCES    ATTACKED    A    POLICE    PATROL    CAR    IN    THE    CAPITAL,    ANNIHILATED    ALL  
<KEYWORDS>  
FMLN:PERP    ORG  
POLICE    PATROL    CAR:TARGET  
PASSENGERS:VICT    IM  
</KEYWORDS>
```

After parsing and applying heuristics one of the caseframes generated would be:

```

NAME :syntag-subj-P   ER P ORG-syntag-act   iv e- at tac ke d
SLOT
mark :syntag
cat :PERP ORG
val :subj
TRIGGER
mark :syntag
word :attacked
val :active
pp :

```

The pattern extracted by this caseframe is:
 <PERP ORG> attacked

and the heuristic used would be:
 <subject> active-verb

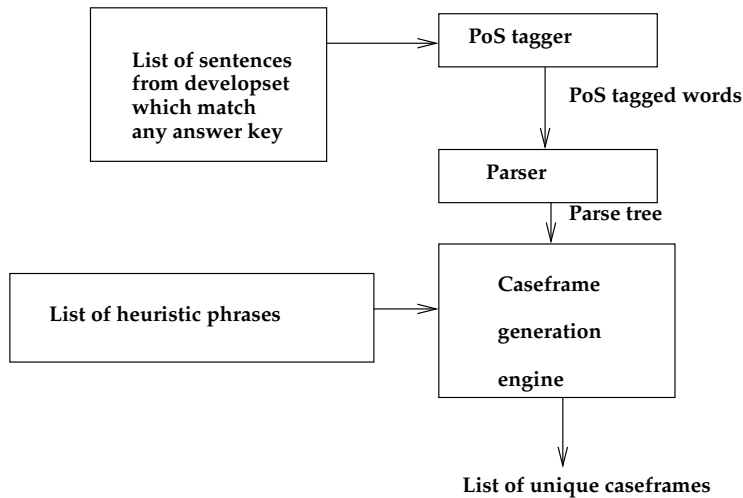


Figure 1: Flowchart of caseframe generation.

After creating caseframes for all sentences that contained an answer, we manually edited the results to caseframes that were most likely related to the terrorism domain. These caseframes were then fed into our information extraction unit.

The second program we created was our information extraction unit. This program uses the manually edited caseframes generated from the training program and a set of dictionaries to create an answer template for a given story. We first used the Perl module `Lingua::EN::Sentences` to break each story into individual sentences. We decided that the Weapons and Perp Org fields of the answer template would be best identified

using pattern matching augmented by the caseframe files. We used a weapon and perpetrator organization dictionary to search the story for keyword matches. If a matching perpetrator organization was found, the caseframes identifying a perpetrator organization were skipped. We found if an article identified a known perpetrator organization, then the entire article was likely based on that perpetrator organization.

The next step was to search each sentence for a caseframe trigger word. If a trigger word is found, the sentence is tagged and parsed. The sentence is then sent through clause segmentation and identification of subject, direct object, and verb voice. The caseframe is then applied to the sentence and the proper slot of the template is filled in.

Once an entire story was checked against caseframes, we then generated an answer for the Incident field of the answer template. This field was generated by doing a keyword match for words related to each category. As the last step, we cleaned up the output of the templates by removing duplicates and removing stop words in order to conform to the answer templates.

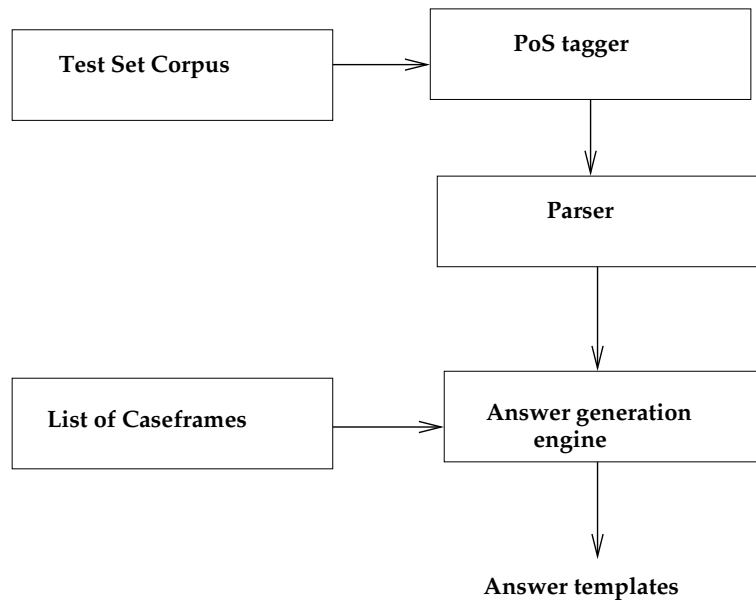


Figure 2: Flowchart of answer generation.

3 Dictionaries and other resources used

From our class discussions on the topic of information extraction, we determined that we would need a part of speech tagger, a parser, a dictionary of domain specific extraction patterns, and dictionaries of domain specific words. Our first task as a group was to determine the problem of parsing and part of speech tagging. In the CPAN Perl module repository (<http://cpan.org>), we discovered a wide array of natural language programs. We discovered a specific module called `Lingua::EN::Tagger` that seemed to fit our needs for part-of-speech tagging. The tagger is based on the Penn Treebank corpus and produced output using the Penn Treebank tag set. The parser we decided to use is called `Cass` which was created by Steven Abney of the

University of Michigan. The last module we used was `Lingua::EN::Sentences` which provided a way of separating a story into individual sentences. We used these three external resources in our project

We created two dictionaries that our information extraction system uses to generate answers for the templates. The dictionaries were created using the development answer keys to extract weapons and perpetrator organizations.

The weapons dictionary contained 54 entries and was organized manually into like categories of weapons. The information extraction program searches through each weapon category until a match is found. If a match is found, the result is then put into the answer key template and the rest of that weapon category is skipped. Each category is organized from specific to more general answers. For example the bomb category looks like:

```
¡BOMBS¡ terrorist bombs incendiary bomb car bomb car-bomb carbomb bomb blast bombs bomb
```

The perpetrator organization dictionary was a simple list of 139 known perpetrator organizations. This list was used to identify any organization that is mentioned in an article. If a keyword matched that the answer was put into the answer template.

4 Contribution of each team member

Sid created the main program for segmenting the Cass parse into clauses and performing the sentence analysis for identification of subject, verb tense, and direct objects.

Aaron helped create the incident, and weapon matching functions as well as the support programs for creating the dictionaries, and editing the caseframes.

Both team members were a part of the review process for the case frames and we pair programmed the training program and the information extraction unit in `dryrun 1`. This was a hack job that Sid cleaned up by implementing his clause segmentor into a Perl module and rewriting the training and information extraction unit to use this module.

5 Emphasis and Originality

Our emphasis of this project was to create a system that automatically created caseframes which could be reviewed quickly. We wanted our system to be able to run on any domain with a minimal amount of changing the code. In order to build such a system we needed a good parser and a good sentence analyzer.

By using the well known Cass parser and `Lingua::EN::Tagger` for part of speech tagging, we were able to utilize the work done on the Penn Treebank corpus. This gave us a good start on the sentence analyzer. Since the CASS did not perform very well on clause segmentation and syntactic tagging, we used the basic CASS output and built our own heuristic based clause segmentation and syntactic tagging components. The sentence analyzer is at the core of our training and information extraction programs.

Once the sentence analyzer was performing well, we were able to utilize the power of heuristics to generate caseframes. This made our system unique in that our system is one of the few in class that can quickly build

caseframes for other domains. After a manual review of the caseframe files our system could be run with little to no change of the source code.

6 NLP Techniques used

The basis of our algorithm was the AutoSlog algorithm which would generate a list caseframes from a set of heuristics and a text corpus with answers known. As mentioned previously, the basic parser used by us was the CASS parser. But since the CASS parser fared pretty badly in clause segmentation and syntactic tagging of phrases, we developed our own heuristics-based clause segmentation, relative pronoun resolution and syntactic tagging which was a wrap-around over the basic CASS parser output.

Below, we explain each of the components of our system, explaining in detail things which were original, and in brief, things which were not new

6.1 PoS Tagging

We used, as mentioned earlier, the PERL package `Lingua::EN::Tagger` for part-of-speech tagging of the input text. This package was a Hidden Markov Model based tagger using bigram probabilities obtained from the Penn TreeBank corpus. But it must be added that this PoS tagger wasn't very accurate on lots of occasions.

6.2 Clause Segmentation

The output of the CASS parser which we used consisted of just the marked up Noun phrases, Verb phrases and Prepositional phrases in a sentence. Fig shows an example parse.

The clause segmentation heuristics which we used were pretty straight forward, but it was pretty extensive all the same. They were

1. If a conjunction word (like 'and', 'or') was found, and there was a verb occurrence before and after it, it usually is a separate clause.

For example in the sentence "*John ate and cried*", "*John ate*" and "*cried*" are two separate clauses. Here a verb after the conjunction is a necessary condition as sentences like "*John drank milk and Coke*" should **not** get separated into two clauses like "*John drank milk*" and "*Coke*".

2. If there was a comma with a verb occurrence before it, it was split as a new clause.

For example in "*The king who was crowned in January, agreed to implement changes*", "*The king who was crowned in January*" and "*agreed to implement changes*" were separated as two different clauses.

3. If there was any relative pronoun (like 'who', 'which' etc.) with a verb occurrence before it, it usually indicated a new clause.

For example, in *"The man killed the boy who picked his purse"*, *"The man killed the boy"* and *"who picked his purse"* were split as different clauses.

4. If there was a second verb phrase in a sentence, it indicated the start of a new clause.

For example, in *"He hit the ball thrown at him"*, *"He hit the ball"* and *"thrown at him"* would be the two clauses generated.

6.3 Relative pronoun resolution

The heuristics for relative pronoun resolution were pretty neat and differentiated between clauses generated because of a comma and those which weren't generated by an occurrence of a comma.

1. If a clause started with a relative pronoun and it was a comma-generated clause, the pronoun usually referred to the Subject of the previous clause.

For example, in *"Explosives loaded in a small truck, which was going to be fired at the presidential house was captured"*, the second clause *"which was going to be fired at the presidential house"* starts with a relative pronoun (*which* in this case). Here *which* refers to *Explosives* which is the Subject of the previous clause, and does not refer to *small truck*. Note here that the second clause is a comma-generated clause

2. If a clause started with a relative pronoun but it was not a comma-generated clause, then the relative pronoun usually referred to the last Noun phrase of the previous clause.

For example, in *"Explosives loaded in a small truck which was driven away ..."*, the second clause *"which was driven away"* is not a comma generated clause but starts with a relative pronoun *which*. Here, *which* refers to *a small truck* and not to *Explosives*.

6.4 Assigning subjects to subject-less clauses

The heuristics which we used to assign Subjects to subject-less clauses was also pretty intelligent in that it was able to use the information whether the clause was generated with a comma or conjunction or neither.

1. If a clause generated either by a comma or by a conjunction does not contain a subject, the subject of the previous clause is made the subject of this clause.

For example, *"He hit the ball and drank milk"* would get clause-segmented as *"He hit the ball"* and *"drank milk"*. The second clause does not have a subject. And since it is a conjunction-generated clause, the subject of this clause would be the subject of the previous clause which is *He*.

Another example would be, in *"He hit the ball, threw the bat and ran"*, the second clause *"threw the bat"* is a comma-generated clause and does not have a subject. And the subject of this clause is again *He*.

2. If a clause is not generated by a comma or conjunction (i.e., it is generated by a second verb phrase) and it does not have a subject, then the last Noun phrase of the previous clause is usually the subject of this clause.

For example, "He threatened to murder individuals involved in the murder" would get clause-segmented as "He threatened to murder individuals" and "involved in the murder". Here, the second clause is neither generated because of a comma nor a conjunction. And as we can see, *involved in the murder* refers to *individuals* and **not He**.

6.5 Combining Adjacent Noun Phrases

Noun phrases which are either adjacent or separated with a conjunction within a clause were combined into a single noun phrase. So in "John killed Mary and Lucy", *Mary and Lucy* would become a single noun phrase.

6.6 Syntactic Tagging

We used heuristics to tag noun phrases as **subjects**, **direct objects**, *indirect objects* and *direct objects of infinitives*. But these were pretty straight-forward and did not involve a lot of cases.

6.7 Tagging Verb Phrases

Verb phrases were also tagged as being either *active*, *passive* or *auxiliary*. The heuristics are well known and straight-forward.

1. If there is a noun phrase before a verb phrase, it was tagged as the subject (SUBJ tag). For example, in "John killed Mary", *John* would be tagged as the SUBJ.
2. If there was a single noun phrase after a verb phrase, it was tagged as the direct object (DOBJ tag). For example, in "John also killed Lucy", *Lucy* would be tagged as DOBJ
3. If there were two noun phrases after a verb phrase, the first one is the indirect object (IOBJ) and the second one is the direct object (DOBJ). In "John gave Mary a gift", *Mary* would be the IOBJ and *a gift* the DOBJ
4. The noun phrase following an infinitive verb was tagged as INF_DOBJ. In "John worked to get money", *money* would be tagged as INF_DOBJ which indicates it is the direct object of the infinitive verb phrase.

6.8 Sample Parser Output

The man who hit the ball and ran , threatened to kill individuals involved in the attack

CLAUSE : :the man who hit the ball and
NP :SUBJ :the man

DT :the
 NN :man
 WP :who
 VP :ACTIVE :hit
 VED :hit
 NP :DOBJ :the ball
 DT :the
 NN :ball
 CC :and
 CLAUSE : :the man ran ,
 NP :SUBJ :the man
 DT :the
 NN :man
 VP :ACTIVE :ran
 VED :ran
 PPC : ,
 CLAUSE : :the man threatened to kill individuals
 NP :SUBJ :the man
 DT :the
 NN :man
 VP :ACTIVE :threatened
 VED :threatened
 INF : :to kill
 TO :to
 VB :kill
 NP :INF_DOBJ :individuals
 NNS :individuals
 CLAUSE : :individuals involved in the attack
 NP :SUBJ :individuals
 NNS :individuals
 VP :ACTIVE :involved
 VEN :involved
 PP : :in the attack
 IN :in
 NP : :the attack
 DT :the
 NN :attack

6.9 Caseframe generation

The caseframe generation part, as already mentioned, is similar to the AutoSlog caseframe generation techniques. We used a list of heuristics on the sentences in the development set which contained at least one answer. This module then generated a list of unique caseframes. The caseframes contained a *trigger* and *slot* information. The *trigger* information would include the trigger word and more details which could

either be its syntactic tag (like ACTIVE, PASSIVE, SUBJ, DOBJ etc) or just its type (like NP, VP etc). The *slot* information would give details of which word(s) should be grabbed if the trigger word matched. It would also contain which category (among VICTIM, TARGET, PERP_INDIV, PERP_ORG, WEAPON) to tag it as.

Given below is sample caseframe:

```

NAME :syntag-subj-T   AR GET -s yn ta g- pas si ve -s ab ota ge d
SLOT
mark :syntag
cat :TARGET
val :subj
TRIGGER
mark :syntag
word :sabotaged
val :passive
pp :
SAMPLE :in addition , a steel factory in zacatecoluca was sabotaged after

```

The above caseframe means that if the trigger word *sabotaged* appears in the passive form (like in *was sabotaged*) in a clause, then the SUBJECT of the clause should be grabbed and tagged as the TARGET

7 Results and Analysis

The overall results of our system on TestSet1 and TestSet2 are given below.

TestSet1

	RECALL	PRECISION	F-MEASURE
Incident	0.74 (74/100)	0.74 (74/100)	0.74
Weapons	0.66 (29/44)	0.49 (29/59)	0.56
Perp_Ind	0.23 (16/69)	0.33 (16/48)	0.27
Perp_Org	0.43 (20/47)	0.47 (20/43)	0.44
Targets	0.27 (17/62)	0.22 (17/78)	0.24
Victims	0.15 (22/144)	0.27 (22/83)	0.19
-----	-----	-----	----
TOTAL	0.38 (178/466)	0.43 (178/411)	0.41

TestSet2

	RECALL	PRECISION	F-MEASURE
Incident	0.78 (78/100)	0.78 (78/100)	0.78
Weapons	0.72 (28/39)	0.48 (28/58)	0.58
Perp_Ind	0.11 (11/101)	0.24 (11/45)	0.15

Perp_Org	0.29 (12/42)	0.27 (12/45)	0.28
Targets	0.10 (6/59)	0.09 (6/64)	0.10
Victims	0.15 (22/149)	0.31 (22/71)	0.20
-----	-----	-----	----
TOTAL	0.32 (157/490)	0.41 (157/383)	0.36

Overall we are happy that there was not too much variation between the results of TestSet1 and TestSet2. Another positive with our system is that it performed decently well on all categories, i.e., it did not totally fail on any one particular category eventhough comparitively it did not fare extremely well on any category also. This, we realize is because of the generality of the system in using caseframes. There were a lot of very good caseframes which were generated which picked up most of the obvious answers in the test corpus. There were also a few over-correct answers such as in the text DEV-MUC3-0016 where *NO ONE* was picked as the VICTIM from the sentence *THE SPOKESMAN ADDED THAT NO ONE WAS INJURED*.

8 Conclusion and Future Work

We realize as a result of this project that Natural Language Processing problems need not always produce best results when a logical algorithm is applied to it. As in, a neat and logical solution might have even produced better results on an open system. But when dealing with a specific domain, lots of things which look clumsy might churn out the best results. We learnt that any NLP solution should first explore things which can be taken advantage off in that particular domain.

We were very happy with the parser and caseframe generation side of our system. It was a pretty robust parser which could, to some extent parse decently, all the complex sentences of the corpus. And we also had a wide range of heuristics which allowed to produce a range of very good caseframes. In this regard we were satisfied with our work.

We could have made this system better in many aspects. For one, we did not pay enough attention to formatting of the output templates. A lot of our answers were right but varied slightly with the answer key. This could have been made better. But this would have been an improvement only from the score point of view. From the overall algorithmic point of view, one thing which we wanted to do but didn't, due to time constraints was to incorporate semantic information to the answers and use that as a filter. In the end, we just made a quick hack by building dictionaries of various categories from the answer keys. Obviously dictionaries are insufficient to take into account all possible new words. We could have also focussed more on the INCIDENT category recognition, because compared with other systems, ours performed poorly on this ; the problem of INCIDENT recognition was much easier than other categories, being a general text categorization problem.

On the whole working on this project was a very challenging and enriching experience.