DESCRIPTION OF THE TASC SYSTEM USED FOR MUC-7

Terry Patten, Beryl Hoffman, Martin Thurn

TASC
13605 Dulles Technology Drive
Herndon VA 20171

tapatten@tasc.com
(703) 793-3700 x2580

INTRODUCTION

TASC has recently developed a technology for learning scenario-template-extraction grammars from examples provided by end-users with no special computational or linguistic knowledge. For straightforward scenario-template problems, complete extraction systems can be learned from scratch in a few hours. The learned systems are fast, robust, and as accurate as carefully handcrafted systems. For more complex extraction problems, such as the MUC-7 task, the grammars can still be learned automatically, but some computational-linguistic work may be required for complex template merging and inference.

Although MUC evaluations do not test some of the strengths of the TASC extraction technology—in particular its speed and robustness—the evaluation of recall and precision was valuable. Early anecdotal experience with this grammar-learning technology suggested that the learned grammars performed practical extraction at least as well as manually constructed grammars. MUC-7 provided the opportunity for a more rigorous evaluation, and a comparison with other scenario-template extraction technologies.

SYSTEM OVERVIEW

TASC's research has focused on the problem of learning the grammatical knowledge required for scenario-template extraction. Only the ST task was performed—off-the-shelf products and well-known techniques were used to handle entity identification and anaphora, and only to the extent necessary for the ST task. IsoQuest's NetOwl™ was used to mark up entities as a pre-processing step. NetOwl does a good job of recognizing many categories such as dates, locations, people, and organizations. Although the names of rockets and satellites are not built into the product, they were easily added to its dictionary.

For straightforward scenario-template extraction problems, a single extractor can easily be trained to handle all the slots. For more complex problems it may be wise to conceptualize the problem as several relatively independent extraction problems. The MUC-7 task was divided into three simpler problems: filling a template for launches, filling a template for payloads, and filling a template for missile events. Each of these three extraction problems was addressed with a separate extractor that was independently trained. The results of the three extractors were merged as the final step in the extraction process. Reducing the problem in this fashion can be particularly beneficial for the learning component, as the information about missile events (for instance), involves very different language than that for rocket launches and payloads. A shared slot helped merge templates for launches and payloads. In cases where the shared slot was not clearly matched, merging decisions were made by a set of domain-specific heuristics.

Breaking up an extraction problem in this manner has proved to be a versatile and valuable technique. Even apparently overwhelming extraction problems can often be broken down into a number of manageable sub-problems that can successfully be trained independently.

The speed of the system as a whole is impressive despite having to run several extractors on the data. The total amount of time taken to process the formal-run data (.39MB) was two minutes on a single-processor machine. This measurement includes the time required to run NetOwl, to run each of the three extractors one after the other, and to do all the merging and inference.

LEARNING GRAMMARS

The grammars TASC uses to perform scenario-template extraction are a special variant of semantic grammars [1] that were developed for robustness in practical applications—they are highly tolerant of corrupted data, ungrammatical sentences, and other types of noise. These grammars are learned from scratch through a combination of statistical and search algorithms. There is no base grammar from which the learning starts—the system begins with no grammatical knowledge at all because it was designed to be able to learn extraction grammars for any natural language, or for technical sublanguages that may contain unusual constructions. Once a grammar is learned, a corresponding extractor is generated automatically.

The process of learning these extraction grammars begins by defining the template through a simple graphical user interface. That interface allows the names of the slots to be entered, and the category of the corresponding filler to be chosen from a simple menu. For instance, the interface can be used to specify that there is a *Payload Owner* slot that should be filled by an *Organization*.

Once the slots and fillers have been specified, an initial extractor is automatically constructed. Having no grammatical knowledge, it is willing to fill the slots with anything of the right category, producing poor results. But these poor results give the end-user a tangible focus for training (not to mention motivation for training).

A graphical display (see Figure 1) is used to show the results of the extraction run on a given set of data. Any mistakes that appear can easily be corrected by the end-user with two clicks of the mouse. The interface displays a text passage with color-coded annotations corresponding to the scenario-template slots. Clicking on an incorrect item will produce a menu of possible alternatives—other slots that could be filled by an item of that type. For instance, if an organization is mistakenly marked as the *Manufacturer* of a payload, the menu will contain other slots that can be filled by an organization—*Owner* will be in the menu, but *Site* will not. The menu always contains an *Other* alternative to indicate that the item does not fill any slot.

After several examples have been corrected, the user invokes the learning mechanism by clicking on the “Learn Rules” button. Once the learning is complete, the new extractor can be run on additional training examples. Initially, the system—with no grammatical knowledge—is merely guessing at how to interpret a passage, and many mistakes are made. With each learning iteration, fewer mistakes are made and the training becomes faster and easier. This iterative bootstrapping process is much more efficient than simply marking up a corpus, because the system quickly starts to extract much of the information correctly, and the annotations must only be done for the ones it gets wrong. In most cases, the system will end up doing the majority of the actual mark-up, with the user simply verifying the results.

Figure 1: Training the system to extract information about payloads

Additional training efficiency is gained by automatically sorting the training examples. Rather than simply presenting examples sequentially as they appear in the training corpus, the best training examples are displayed first. Choosing the right training examples can allow the system to improve faster in the early stages—and this reduces the human effort because of the bootstrapping process. Learning faster in the early stages also ensures the best results if the user runs out of time or patience and is not able to work through the entire training corpus.

One technique for ordering training examples is to display the examples containing the most ambiguity first. This allows the teacher to provide the most information (in the information-theoretic sense) in the least amount of time. For instance, an organization in the payload extractor could be the owner, the manufacturer, the recipient, or a spurious organization such as the parent of the manufacturer. Obtaining feedback on examples that contain several organizations allows the system to quickly learn how to put organizations in the right slots.

Another ordering technique is to avoid displaying examples if several similar examples have already been presented. There is no point displaying hundreds of similar examples—the system will learn the relevant grammatical knowledge after the first few. This technique can significantly reduce the required training time and allows the system to benefit from a larger variety of constructions earlier in the training process.

In summary, being able to learn extraction grammars from training examples is only useful if the required training examples can be obtained cost-effectively in practice. Annotating examples can be prohibitively expensive, but the graphical user interface, bootstrapped training, and ordered examples described above allow an end-user to perform the training quickly and easily. The total amount of human effort involved in creating the three extraction grammars for the MUC-7 scenario-template task was roughly one day.

DISCUSSION

Performance vs. Other Systems

Using grammars that were learned automatically from scratch, the TASC system received the second-highest score on the scenario-template formal run and the highest score on the dry run. This clearly suggests that the grammar-learning technology is able to compete with the very best methods—both manual and automatic—for building grammars for extraction systems. It is important to understand, however, that there were several issues that made these MUC-7 tasks particularly ill-suited for the TASC technology. This section will outline the reasons why this technology performs significantly better on many other scenario-template extraction problems.

First, TASC did not participate in the NE evaluation, and used an off-the-shelf named-entity tool with a limited customization effort. Given the importance of MUC-7 categories such as rockets, satellites, and military vehicles, none of which are built into NetOwl, the score was likely lowered by the limited effort in this area. The MUC experiment is really not designed for participants who want to evaluate scenario-template technology independently of named-entity issues. The combination of grammar learning and off-the-shelf name tagging should perform better on tasks that were not specifically chosen to be challenging for entity-identification tools.

Second, the discussion-oriented nature of the *New York Times* articles poses significant difficulties for approaches based on grammar learning. At first glance, the *New York Times* articles would seem to be ideal for learning—they are free from spelling errors and contain only highly grammatical text. But discussion articles provide little consistency in how information is presented. Indeed, repeated use of similar constructions would be considered poor style. This, together with the small amount of *New York Times* data provided, made finding generalizations extremely difficult. The grammar-learning techniques embodied in the TASC system produce better results when several examples of each construction can be provided during the training—as is normally the case in *Wall Street Journal* business briefs, technical documents, news briefs, and many other sources.

Learning extraction grammars automatically has enormous benefits in terms of the time and effort required to build a system, but there has always been a recall/precision penalty. The recall and precision scores of the TASC system demonstrate that automatically learned grammars can compete with manually constructed grammars, even for complex extraction problems, and even when the learning takes place under difficult conditions.

Finally, it is important to note that while the MUC-7 task required customized template merging by computational linguists, this new learning technology allows simpler template-extraction problems to be successfully tackled by an end-user with no additional assistance. For many practical problems, a financial or intelligence analyst can complete a scenario-template extractor for a completely new problem in a few hours. Applying extraction to spur-of-the-moment, time-critical applications has not been possible until now.

Performance vs. Humans

The TASC extraction system, like all other MUC ST extraction systems to date, has recall and precision scores that are much lower than those published for the human annotators. It should be noted, however, that MUC evaluations are heavily biased against fast extraction systems and in favor of the human annotators because of the small amount of test data and the practically unlimited amount of time allowed for the task. In practice, time limitations have a significant impact on recall—information cannot be extracted from material that is not read. Many applications require the monitoring of a large number of online sources, and recall for humans can be very low because of time constraints. In this type of situation, TASC's fast extraction system will achieve recall scores that are several times better than the human scores, and hence combined recall and precision scores that are significantly better than those for humans.

CONCLUSIONS

Being able to learn extraction grammars from examples provided by an end-user greatly increases the practicality of scenario-template-extraction technology. Manually constructing grammars is a slow, laborious, and ultimately expensive process. Largely eliminating this cost produces a huge swing in the cost/benefit ratio—even if some effort by computational linguists is still required for other aspects of the problem.

The benefits of trainable systems can be lost, however, if the training itself is too expensive or requires special skills. TASC has addressed this issue by developing a training scheme that requires no special computational or linguistic knowledge of the teacher, involves an extremely simple graphical interface, uses bootstrapping to automate the creation of training examples, and automatically selects the best training examples. An end-user can train this technology quickly and easily, ensuring the benefits of learning grammars are maximized.

The most significant result here is that automatically learned grammars do not entail a recall/precision sacrifice. The scores achieved using the automatically learned grammars clearly indicate that this new technology can successfully compete with handcrafted grammars, even when faced with complex problems and data ill-suited to learning.

REFERENCES

[1] Allen, J., *Natural Language Understanding*, Bejamin/Cummings, 1987, pp. 250-253.