Learning Embedded Discourse Mechanisms for Information Extraction

Andrew Kehler
SRI International
Artificial Intelligence Center
333 Ravenswood Avenue
Menlo Park, CA 94025
kehler@ai.sri.com

Abstract

We address the problem of learning discourse-level merging strategies within the context of a natural language information extraction system. While we report on work currently in progress, results of preliminary experiments employing classification tree learning, maximum entropy modeling, and clustering methods are described. We also discuss motivations for moving away from supervised methods and toward unsupervised or weakly supervised methods.

Introduction

In recent years, there have been two distinct but related trends in natural language processing (NLP) research. The first we call the *automation* trend, characterized by the movement from systems in which complicated data and rules are hand-crafted to those in which they are acquired automatically, through either symbolic or statistical learning. This movement is almost certainly a prerequisite to achieving broad deployment of NLP technology; nonetheless, it continues to be a work in progress, as NLP systems are still predominantly hand-coded (but see Cardie (1997) and references therein).

The second movement we call the *engineering* trend, characterized by the movement from approaches that are theoretically grounded in our understanding of human language processing to those that simply work well at some task, using whatever techniques are most appropriate. This movement is much closer to completion; *information extraction* (IE) systems,[1] for instance, typically eschew the once-standard detour through complex linguistic processing, relying instead on cruder representations and techniques that developers have found to be better suited for the task. While the limitations of these representations and techniques are well-known, the inaccuracies introduced in moving to more complex systems have generally outweighed the benefits.

In this paper we describe current research in learning discourse processing strategies embedded in a real-world IE system, particularly SRI's FASTUS (Hobbs *et al.* 1996), which is being pursued in the spirit of the trends outlined above. We report on work currently in progress, including negative results of preliminary experiments employing classification tree learning, maximum entropy modeling, and clustering methods. These results highlight several challenges in learning embedded discourse strategies that are less evident in studies of more separable NLP problems. In particular, we find that good performance by a learned model as trained and tested on human annotated data does not necessarily translate into increased performance of the system in which the model is embedded. Furthermore, which discourse strategy is optimal is highly dependent on the nature and accuracy of the annotated input it receives, which in any realistic development setting is constantly evolving, thus requiring too much time- and expertise-intensive data encoding to rely on the use of standard supervised methods. These facts motivate moving to certain types of weakly supervised methods, which is the subject of continuing work.

Information Extraction

IE systems process natural language and produce representations of the information relevant to a particular application, typically in the form of database templates. As an example, we consider the task specified for the Sixth Message Understanding Conference (MUC-6), which was, roughly speaking, to identify information in business news that describes executives moving in and out of high-level positions within companies. When a MUC-6 system encounters a passage such as example (1),

(1) John Smith, 47, was named president of ABC Corp. Smith replaces Mike Jones.

[1] See, for instance, the Proceedings of the Sixth Message Understanding Conference (1995).

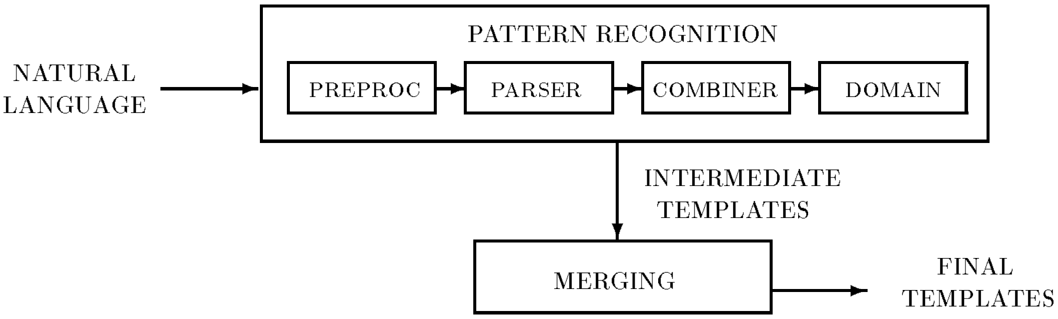

Figure 1: Architecture of the FASTUS Information Extraction System

it should extract the information that *Mike Jones* is 'out' and *John Smith* is 'in' at the position of *president* of company *ABC Corp.*

Figure exhibits the architecture of SRI's FASTUS system (Hobbs *et al.* 1996), which has two major components. The first component consists of a series of finite state transducers that recognize patterns in the text and create templates representing event and entity descriptions from them. The second component merges templates created from different phrases in the text that overlap in reference.

We illustrate by walking through an analysis of passage (1). The input is initially processed by using the finite-state transducers to recognize relevant patterns and annotate the text accordingly. First, one or more preprocessing phases recognize low-level patterns such as person names, organization names, and parts of speech.

> [John Smith]$_{PERS-NAME}$ [47]$_{NUM}$ [was]$_{AUX}$ [named]$_{V}$ [president]$_{N}$ [of]$_{P}$ [ABC Corp]$_{ORG-NAME}$

The parsing phase identifies very local syntactic constituents, such as noun groups and verb groups; no attachment of ambiguous modifiers is attempted.

> [John Smith]$_{PERS-NAME}$ [47]$_{NUM}$ [was named]$_{VG}$ [president]$_{NG}$ [of]$_{P}$ [ABC Corp]$_{ORG-NAME}$

The combiner phase pieces together slightly larger constituents when it can be done reliably, here analyzing *47* as an age appositive to the proper name.

> [John Smith, 47]$_{PERS-NG}$ [was named]$_{VG}$ [president]$_{NG}$ [of]$_{P}$ [ABC Corp]$_{ORG-NAME}$

Finally, the domain phase applies domain-dependent patterns to the sentence to identify clause-level states and events. In this case, the entire sentence will match such a pattern.

> [John Smith, 47, was named president of ABC Corp]$_{DOMAIN-EVENT}$

Recognizing a pattern in the domain phase typically causes one or more template objects to be created. In light of the MUC-6 task specification, we defined *transition* templates that track movements in and out of positions at companies; a person's leaving a job is represented by the start state of a transition, whereas a person's taking a job is represented by the end state. Therefore, the person, company, and position in the first sentence are represented in an end state since Smith is taking the described position. To facilitate certain types of inferencing during discourse analysis, we also posit that someone, at this time unknown, is most likely leaving the position; this being represented in the transition's start state as shown in Figure 2.

$$\begin{bmatrix} \text{START} & \begin{bmatrix} \textsc{Person} & \text{---} \\ \textsc{Position} & \textsc{president} \\ \textsc{Organization} & \text{ABC } \textsc{Corp.} \end{bmatrix} \\ \text{END} & \begin{bmatrix} \textsc{Person} & \textsc{John Smith} \\ \textsc{Position} & \textsc{president} \\ \textsc{Organization} & \text{ABC } \textsc{Corp.} \end{bmatrix} \end{bmatrix}$$

Figure 2: Template Generated from *John Smith was named president of ABC Corp.*

The second sentence in the passage, *Smith replaces Mike Jones*, is then analyzed by the pattern matching phases, the details of which we omit. Having recognized a domain-level pattern, all that is known is that there is a start state involving the person Mike Jones and an end state involving a person John Smith, represented by the template shown in Figure 3.

As they stand, of course, these two templates do not appropriately summarize the information in the text; there is a discourse-level relationship between the two that must be captured. This is the job of the merging component. When a new template is created, the merger attempts to unify it with templates that pre-

$$\begin{bmatrix} \text{START} & \begin{bmatrix} \textsc{Person} & \textsc{Mike Jones} \\ \textsc{Position} & \text{---} \\ \textsc{Organization} & \text{---} \end{bmatrix} \\ \text{END} & \begin{bmatrix} \textsc{Person} & \textsc{John Smith} \\ \textsc{Position} & \text{---} \\ \textsc{Organization} & \text{---} \end{bmatrix} \end{bmatrix}$$

Figure 3: Template Generated from *Smith replaces Mike Jones.*

cede it. In this case, the template shown in Figure 3 should be unified with the template shown in Figure 2 to produce the template shown in Figure 4, which will lead to the correct output when the intermediate templates are converted to the final templates in the MUC-6 output format. If this merge had not taken place, FASTUS would have generated output describing who took the position from the end state of the first transition, but would have missed the information concerning who left the position.

$$\begin{bmatrix} \text{START} & \begin{bmatrix} \textsc{Person} & \textsc{Mike Jones} \\ \textsc{Position} & \textsc{president} \\ \textsc{Organization} & \text{ABC }\textsc{Corp.} \end{bmatrix} \\ \text{END} & \begin{bmatrix} \textsc{Person} & \textsc{John Smith} \\ \textsc{Position} & \textsc{president} \\ \textsc{Organization} & \text{ABC }\textsc{Corp.} \end{bmatrix} \end{bmatrix}$$

Figure 4: A Successful Merge

The development of FASTUS is a product of the engineering trend described in the introduction. In bypassing much of the complex linguistic processing characteristic of previous systems, we focus on approximations to such processing using a more coarse-grained style of analysis that is closer to the needs of the IE application. For instance, the use of cascaded finite-state transducers approximates the computation of syntax, and the construction of intermediate template representations approximates semantics. Likewise, the merging component serves as an approximation for a variety of discourse and inference processes, which include resolving various sorts of anaphoric reference (pronouns, definite NP reference, definite event reference), non-anaphoric coreference (e.g., coreference between proper names), implicit arguments, and other types of unmarked coreference that are resolved by drawing inferences required to understand the discourse as coherent.[2] In the example, merging recovers the information about what position and organization are 'missing' in the *replacing* event described in the second sentence.

To allow for end-to-end evaluation of an IE system developed for a given application, typically a set of keys are encoded for a moderate-size set of sample messages. The keys are then divided into a *development* set, used by system developers to gauge their progress, and an unseen *test* set, used to evaluate the completed system. For the MUC evaluations, two sets of 100 keys, one for each purpose, have been provided by the organizers. Using a test key, IE systems are typically evaluated by calculating the metrics *recall*, *precision*, and *F-score*. Recall is the percentage of the template slot fills in the key that the system produced. Precision is the percentage of the system's template fills that are in the key. Therefore, if there are 100 correct fills in the key, and the system produces 80 fills of which 60 are correct, then the system's recall is $\frac{60}{100}$=0.60, and its precision is $\frac{60}{80}$=0.75. The F-score averages these two numbers, but penalizes disparities between them.

Learning Merging Strategies

In accordance with the automation trend described in the introduction, our goal is to move toward IE systems that automatically acquire their data and rules. For systems with an architecture similar to that shown in Figure , this requires two efforts: learning patterns and pattern matching strategies, and learning merging strategies. A discussion of current work in pattern acquisition can be found in Cardie (1997) and references cited therein. Here we address the acquisition of merging strategies within FASTUS, with the goal of improving performance on the end-to-end extraction task while simultaneously reducing the need to develop these strategies by hand.

Supervised Methods

The first pair of approaches we describe are supervised methods for learning. These methods require a set of training data that the learner can consult in constructing its model. For our initial experiments, we ran the 100 MUC-6 training messages through FASTUS and wrote out feature signatures for the 534 merges that the system performed. The feature signatures were created by asking a set of fifty questions about the context in which the proposed merge is taking place, referencing the content of the two templates and/or the

[2] This is not to suggest that one cannot also build special-purpose modules for certain of these processes, for instance, FASTUS currently has a module for pronoun and other types of anaphoric reference. In fact, the optimal merging strategy will depend on the quality of these mechanisms, a point to which we return later.

distance between the phrases from which each template was created. Some example questions are:

- SUBSUMED?: true if the contents of one template completely subsumes the contents of the other.
- UNNAMED-REFERENCES?: true if either transition has a slot fill with an object lacking a proper name, e.g., "an employee" in the person slot. While these objects can merge with other (perhaps named) entities of the same type, in general they should not.
- LESS-THAN-700-CHARS?: true if the phrases from which the templates are created are less than 700 characters apart in the text.

The feature signatures having been written, we examined the texts and manually encoded a key for each.

We attempted two approaches to classifying merges using this corpus as training data. The first approach was to grow a classification tree in the style of Breiman et al. (1984). At each node, the algorithm asks each question and selects the one resulting in the purest split of the data. Node splitting was halted when the impurity of the best split was worse than the impurity of the parent node, or if the number of examples within the node was less than a preset threshold.[3] Entropy was used as the measure of node impurity.

In the second set of experiments, we used the approach to maximum entropy modeling described by Berger et al. (1996). The two possible values for each of the same fifty questions (i.e., yes or no) were paired with each of the two possible outcomes for merging (i.e., correct merge or not) to create a set of *feature functions*, or *features* for short. These features were used to define *constraints* on a probabilistic model in which we take the expected value of each feature in the model to be the values found in the distribution of the training data. After choosing a set of such constraints, we identify the model meeting these constraints that has the maximum entropy – this is the model that assumes the least information beyond those constraints. Berger et al. (1996) show that this model is a member of an exponential family with one parameter for each constraint. They demonstrate that the optimal values for these parameters can be obtained by maximizing the likelihood of the training data with respect to the model, which can be performed using their *improved iterative scaling* algorithm. We used the learned maximum entropy model as a classifier by considering any merge with a probability strictly greater than 0.5 to be correct, and otherwise incorrect.[4]

Out of the available set of questions, each approach selects only those that are most informative for the classifier being developed. In the case of the decision tree, questions are selected based on how well they split the data. In the case of maximum entropy, the algorithm approximates the gain in the model's predictiveness that would result from imposing the constraints corresponding to each of the existing inactive features, and selects the one with the highest anticipated payoff. One potential advantage of maximum entropy is that it does not split data like a decision tree does, which may prove important as training sets will necessarily be limited in their size.

In our preliminary evaluations, we used two-thirds of our annotated corpus as a training set (356 examples), and the remaining one-third as a test set (178 examples). We ran experiments using three different such divisions, using each example twice in a training set and once in a test set. In each case the maximum entropy classifier chose features corresponding to either six or seven of the available questions, whereas the decision tree classifier asked anywhere from seven to fourteen questions to get to the deepest leaf node. In each case there was considerable, but not total, overlap in the questions utilized. Adding the errors from the three evaluations together, the decision tree made 34 errors (out of a possible 534), in which 13 correct merges were classified as incorrect and 21 incorrect merges were classified as correct. The maximum entropy classifier made a total of 31 errors, in which 14 correct merges were classified as incorrect and 17 incorrect merges were classified as correct. This is compared to a total of 139 errors out of the 534 merges that the current merger made.

These results may appear to be positive, as it would seem that both methods found some reliable information on which to make classifications. However, the reader should not find these numbers to be particularly meaningful. For one, merging is highly dependent on the system in which it is embedded, and is therefore not something that is particularly worth evaluating in itself. Furthermore, the number of correct merges is dependent on a variety of idiosyncrasies related to the number and type of templates that the MUC-6 system creates, most of the details of which we have omitted. What we are really interested in is how much of an impact these improvements have on the performance of the system on the end-to-end template-filling task.

[3] We also experimented with growing trees beyond these thresholds, which yielded no significant differences in accuracy.

[4] Kehler (1997) discusses approaches utilizing the maximum entropy method for template merging applied to entity coreference.

Thus, we replaced the existing FASTUS merging algorithm, which merges any two templates as long as they unify and meet any *a priori* constraints defined on them, with two more discriminating mergers, each directed by one of our learned classifiers. The first version consulted the decision tree and merged only when the example was classified as correct. The second version did the same using the maximum entropy classifier. For these experiments, the two models were trained using the entire set of 534 examples.

As we were still experimenting at this point, we were not ready to perform an evaluation using our set of blind test messages. As an information gathering experiment, we applied FASTUS using the new mergers to the corpus of messages that produced the training data. We would of course expect these experiments to yield better results than when applied to unseen messages. Nonetheless, the results were humbling – both experiments failed to improve the performance of the overall system, and in fact degraded it slightly. Generally, a point of precision was gained at the expense of a point or two of recall.

Clearly, there is a rift between what one might consider to be good performance at discriminating correct and incorrect merges based on human judgments, and the effect these decisions have on overall performance. Because the baseline FASTUS algorithm merges too liberally, using the classifiers causes many of the incorrect merges that were previously performed to be blocked, at the expense of blocking a smaller handful of correct merges. Thus, one conclusion we might draw is that the correct merges the system performs help its end-to-end performance much more than incorrect merges hurt it.[5] For instance, it may be that correct merges often result in well-populated templates that have a marked impact on performance, whereas incorrect merges may often add only one incorrect slot to an otherwise correct template, or even result in templates that do not pass the threshold for extractability at all. In fact, in certain circumstances incorrect merges can actually help performance, if in fact two incorrect templates that would produce incorrect end results are unified to become one such template.

It could also be that the pattern matching phases in FASTUS are currently not well suited for this type of learning strategy, perhaps in being too conservative in passing fragmental analyses onto merging. Thus, we could adapt the pattern matching phases and attempt learning again. This would require that we encode a new set of training data by hand, as the previous data set will no longer be relevant. And as is the case in any realistic IE system development environment, FASTUS is constantly undergoing change, and thus training data will continually need to be recoded.

This, of course, is unappealing since data encoding requires a lot of time and expertise, not particularly unlike hand-coding system mechanisms and heuristics. In fact, the need for these resources is exactly what we have been trying to avoid; we have not addressed the motivations for the automation trend if we are merely replacing an expert's time building systems with an expert's time encoding data. Thus, we need to look to other methods for learning merging mechanisms.

Unsupervised Methods

Naturally, the alternatives to supervised methods are unsupervised methods. We consider replacing our merging algorithm with one that performs an unsupervised clustering of the templates and merges the templates in each cluster. Of course, we will not know *a priori* how many clusters there are, that is, how many templates we should be left with when we are finished. A method that does not require such knowledge is Hierarchical Agglomerative Clustering (HAC) (Duda & Hart 1973; Everitt 1980, inter alia).

The HAC algorithm is conceptually straightforward. Given a set of examples, the algorithm begins by assigning each to its own cluster. A predetermined similarity metric is then applied to each pairwise combination of clusters, and the most similar pair combined. The process is iterated until no pair of clusters have a similarity that exceeds a preset threshold.

Our application of clustering is a bit different from many problems to which clustering has been applied. For one, our clusters will always have only one member, since templates are merged upon clustering. Issues with how to compute similarity between two nonsingleton sets of data points are therefore avoided. Furthermore, our notion of similarity is non-standard. Usually, similar examples are distinct, but have properties that are "close" to each other. Here, similarity is meant to measure the likelihood that the two templates are incomplete descriptions of the same complex of eventualities (i.e., the same transition), but at the same time the templates themselves may look very different.

We performed some informal experiments in which we intuited a similarity metric, assigning weights to a subset of the questions that we had defined for the supervised learning experiments. For instance, templates that were created from phrases close to each other in

[5] In the case of learning classification trees, we could incorporate *misclassification costs* in which the cost of misclassifying a correct merge would be set significantly higher than the cost of misclassifying an incorrect merge. No such experiments have been tried.

the text and that overlapped in content received high similarity, whereas those that were far apart and did not overlap received low similarity. Instead of merging incrementally as in the supervised learning experiments, pattern matching was first applied to the entire text, and the resulting templates were then clustered and merged until no pair of templates passed a preset similarity threshold.

Running the system over the MUC-6 development set yielded results similar to our experiments using the supervised mergers. We did not find this to be particularly surprising; the mediocre results could be attributable to the similarity metrics not being very good, or because of the conservativity of the input the merger was receiving.

We did not push this approach any further, however, because it is still lacking with respect to one of our goals for pursuing learning strategies. While it addresses the problem of requiring training data, it does not address the fact that the optimal merging strategy is inherently dependent on its input. If we encode a similarity metric for clustering and keep it fixed, we are left with only a single degree of freedom – the similarity threshold at which to halt the clustering process. While this may yield some leverage (for instance, good input to the merger may call for a high threshold, whereas bad input may call for a lower threshold), it will certainly be too inflexible in the general case. There might be any of a number of factors which could influence the likelihood of a potential merge within a particular application, and thus it seems that something tied to the application needs to guide the learning process.

Weakly Supervised Methods

Recall that when developing an IE system, one typically encodes (or is given) a moderate-size set of development keys for a set of sample messages. These keys only need to be encoded once, and are already necessary to support manual or automatic pattern acquisition. We have not used these keys thus far; they are not easily utilized for the types of supervised learning we pursued because of the difficulties in aligning the inaccurate and incomplete intermediate templates produced by the system with the (normalized) targeted end results. However, we can use the keys to evaluate the end results of the system, and attempt to tune a merging strategy based on these evaluations. This view is appealing considering that it is improving end-to-end performance that we are seeking in the first place.

Thus, we consider a form of what we are calling 'weakly' supervised learning. We use the HAC mechanism described in the previous section, but attempt to learn the similarity metric instead of stating it explicitly. The search through the space of possible similarity metrics will be driven by end-to-end performance on a set of training messages.

We start by defining a space of similarity metrics. In a preliminary experiment, we used seven of the questions that were used in the supervised experiments, coupled with their negations, for a total of fourteen questions. These questions are assigned weights, either positive or negative, that get incorporated into a similarity metric when the question is true of a potential merge. Let w_i be the weights assigned to corresponding questions q_i, and let the function $f_{q_i}(t_1, t_2)$ be 1 if the question q_i is true of the templates t_1 and t_2, and 0 if not. Then the similarity $S(t_1, t_2)$ is given by

$$S(t_1, t_2) = \frac{e^{\sum_i f_{q_i}(t_1,t_2)*w_i}}{e^{\sum_i f_{q_i}(t_1,t_2)*w_i} + 1}$$

This function allows for any value w_i, always yields a value between zero and one, and has a value of 0.5 when all the applicable weights are zero.

We used a simple hill-climbing procedure to tune the weights w_i. The weights are initialized to zero and the similarity threshold set to 0.49. The algorithm begins by processing the training set, in this case a 10-message subset of the MUC-6 development corpus, with the initial configuration to establish a baseline F-score. In this pass, all compatible templates have similarity 0.5 and therefore merge; ties in similarity are broken by preferring the pair created from phrases that are closest together in the text. The algorithm then iterates, selecting one of the questions at random and permuting its weight by a random amount, in this case between -0.33 and 0.33. The system is then rerun over the training set and the F-score measured. Any permutation resulting in an F-score that is strictly greater than the current baseline is adopted as the new baseline, otherwise the old baseline is kept.

In our initial run, we allowed the system to iterate 4007 times, but the search got stuck in a maximum after only the 147th trial. In fact, it adopted a permutation to the similarity metric only six times. There was not much dynamic range in the F-score, moving from an initial score of 57.71 to 59.05, perhaps suggesting that the 10 messages were not a large enough set, or perhaps that the questions selected were not particularly informative. The resulting weights for the six questions with non-zero weights were:

not-less-than-700-chars?	-0.2958
subsumed?	0.1607
not-less-than-7-nulls?	-0.1558
sentence-internal?	0.0878
some-overlap-in-content?	-0.0862
unnamed-references?	0.0188

The system learned larger weights in places in which one might expect, including a weight against templates that were evoked 'far apart' in the text (here, 700 characters), a preference for cases in which one template subsumed the other, and a weight against merging very sparsely populated templates. The preference for sentence-internal merges is likewise expected. The final two weights are more surprising; one would expect a positive weight for templates that have overlapping content, and a negative weight for templates containing unnamed references.

Running the system over the entire set of training messages using this similarity metric resulted in a negligible increase in F-score (0.11). This was only a first-cut experiment, however, and there is a lot of room for exploration with more comprehensive sets of questions, larger training sets, and more sophisticated search strategies. Further experimentation along these lines is underway.

Future Directions

The lack of success of our experiments so far could be attributed to several facts. For one, as we indicated earlier, it could be that the pattern matching components are currently not well suited for learning of this sort. In particular, using learning strategies to constrain merging might allow for a fairly liberal pattern matching strategy, where in fact we have been using a more constrained strategy given our relatively unconstrained merging mechanisms. In the future, we will be experimenting with different strategies and gauging their effect on the learning process.

What we suspect is the biggest problem, however, is the lack of reliable, informative indicators for merging decisions. Unlike problems in which local information appears to bear highly on the outcome, including, for instance, part-of-speech tagging (Church 1988; Brill 1992, inter alia) and sense disambiguation (Yarowsky 1994; 1995, inter alia), none of the questions we have formulated appear to be particularly indicative of what effect a potential merge will have on system performance. Much of our future work will be focused on encoding more sophisticated types of contextual information, including features based on the strings and patterns from which templates are created, and on discourse structure.

Conclusions

Learning merging strategies that are embedded within an information extraction system poses several challenges that are less evident in studies of more separable natural language processing problems. First, dealing with a constantly evolving system requires too much time- and expertise-intensive data encoding to rely on the use of supervised methods. In this context, the same motivations for moving from hand-coded systems to ones that automatically acquire their rules and data also motivate a movement from supervised methods to unsupervised or weakly supervised methods. Second, good performance by a learned model with respect to human annotated data does not always translate into increased performance of the system in which the model is embedded, when in fact we are primarily concerned with the latter.

There are also other, more general reasons for moving away from supervised learning. Perhaps more so in discourse processing than in other areas, the proper representations to be reasoning with and producing, whether it be symbolically or statistically, remain poorly understood. Besides being time-intensive to build, annotated data sources by their nature assume many of these properties. An undue focus on supervised learning jeopardizes the usefulness of such work to researchers building their own systems using their own representations, and risks driving theoretical work in discourse processing to a premature close.

Acknowledgments

The author thanks John Bear and two anonymous reviewers for helpful comments and criticisms, and the SRI FASTUS team for their contributions to the system in which this work is embedded.

References

Berger, A.; Della Pietra, S. A.; and Della Pietra, V. J. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics* 22(1):39–71.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees.* Belmont, CA: Wadsworth.

Brill, E. 1992. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing.*

Cardie, C. 1997. Empirical methods in information extraction. *AI Magazine* 18(4):65–79.

Church, K. W. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Pro-*

ceedings of the Second Conference on Applied Natural Language Processing, 136–143.

Duda, R. O., and Hart, P. E. 1973. *Pattern Classification and Scene Analysis.* New York: John Wiley and Sons.

Everitt, B. 1980. *Clustering Analysis.* New York: John Wiley and Sons.

Hobbs, J. R.; Appelt, D. E.; Bear, J.; Israel, D.; Kameyama, M.; Stickel, M.; and Tyson, M. 1996. FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In *Finite State Devices for Natural Language Processing.* Cambridge, MA: MIT Press.

Kehler, A. 1997. Probabilistic coreference in information extraction. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP-97)*, 163–173.

Proceedings of the Sixth Message Understanding Conference. 1995.

Yarowsky, D. 1994. Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL-94)*, 89–95.

Yarowsky, D. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*, 189–196.