WAVE: An Incremental Algorithm for Information Extraction

Jonathan H. Aseltine
Department of Computer Science
University of Massachusetts, Amherst, MA 01003-4610
aseltine@cs.umass.edu

Abstract

This paper describes WAVE, a fully automatic, incremental induction algorithm for learning information extraction rules. Unlike traditional batch learners, WAVE learns from a stream of training instances, not a set. WAVE overcomes the inherent problems of incremental operation by maintaining a generalization hierarchy of rules. Use of a hierarchy allows similar rules to be found efficiently, provides a natural bound on generalization, enables recall/precision trade-offs without retraining, and speeds extraction since all rules need not be applied to an instance. Finally, because the reliability of rule predictions are continually updated throughout storage, the hierarchy can be used for extraction at any time. Experiments show that WAVE performs as well as CRYSTAL, a related batch algorithm, in two very different extraction domains. WAVE is significantly faster in a simulated incremental application setting.

Introduction

As the Internet continues to grow at an astonishing rate, the amount of online text available and the need for automatic methods to extract information from it have also grown. An information extraction (IE) system extracts specific, well-defined types of information (concepts) from text from a restricted domain. Handcrafting of IE systems is often time-consuming and expensive, hence trainable IE systems have come to the forefront as a way to meet that need.

AutoSlog (Riloff 1993) was an early attempt at automating the construction of an IE rule lexicon. AutoSlog proposes extraction rules for each annotated concept in a training corpus. A human decides which proposed rules should be kept.

CRYSTAL (Soderland 1995) removed the human from the loop. Extraction rules are induced from annotated training, and evaluated without human intervention. CRYSTAL processes training instances in batches: documents are collected, annotated, then fed *en masse* to the induction algorithm. If a new set of training instances becomes available, it is added to the previous batch and the new batch is reprocessed from scratch. Reprocessing large training sets can take hours. In static application environments, the additional time needed to reprocess batches is merely an inconvenience. In interactive, dynamic application environments, a few hours to retrain can be quite problematic: while the system is being retrained, deadlines are missed and user interaction stops.

WAVE is an incremental induction algorithm for learning information extraction rules. Like CRYSTAL, WAVE learns and evaluates its rules automatically. Unlike CRYSTAL, WAVE incrementally constructs and evaluates extraction rules from a *stream* of training instances, not a set. The rules can be used to extract information from other instances at any time. These properties make WAVE a better choice for interactive, dynamic application settings.

Representing Training Instances and Extraction Rules

Before rule learning can begin, the text must be parsed into a series of training instances. The syntactic analyzer MARMOT (Fisher et al. 1996) identifies syntactic buffers (subject, verb, object, and so on) and a few other simple features, such as the verb root and voice (active or passive). Semantic analysis is trivial, limited to a simple table lookup to assign a class to each term. For example, consider the clause "MEMBERS OF THE FARABUNDO MARTI NATIONAL LIBERATION FRONT ATTACKED AN ELECTRIC POWER SUBSTATION" from the Fourth Message Understanding Conference (MUC-4) domain of Latin American terrorism. Parsing this clause results in the following frame:

Subject:	MEMBERS OF THE FARABUNDO MARTI NATIONAL LIBERATION FRONT {Human, Terrorist-Organization}
Verb:	ATTACKED (*attack, active*) {Attack}
Object:	AN ELECTRIC POWER SUBSTATION {Building Energy}

To train the system with this clause, annotations are added to each buffer signifying the concepts it contains. For example, the "Object:" buffer above contains the extraction concept *physical-target*. Training instances thus constructed are used as initial rules (sometimes called "instance rules") to seed inductive generalization: the features for each buffer are *constraints* to be satisfied; any annotations on a buffer are *predictions* that will be made if all the constraints in the frame are satisfied.

The WAVE Algorithm

WAVE automatically learns extraction rules through generalization of training instances. Batch algorithms like CRYSTAL have the considerable advantage of having the entire training set at hand when making control decisions. Because WAVE is incremental, the algorithm must make decisions about the current instance in the input stream without knowing anything about instances further upstream. Moreover, the algorithm must be able to find good generalizations efficiently and update its error estimates for existing rules as new evidence is received.

WAVE addresses these issues by maintaining a generalization hierarchy of extraction rules. Organization of the hierarchy is based on the *covers* relation: rule A covers rule B if and only if all constraints in A are also in B *and* any predictions made by A are made by B as well. Each training instance is stored under the most specific rules in the hierarchy that cover it. During storage, the reliability of each rule is updated based on the instance's annotations. Once stored, the training instance is generalized with its siblings to form new rules. This strategy has several benefits:

- Rules most similar to the current instance rule in the stream can be found quickly. By finding similar rules, the bottom-up generalization strategy is less likely to miss good rules due to generalization steps that are too large.
- The hierarchy naturally bounds the generalization process. A new instance rule is unified only with rules close to it in the hierarchy.
- The hierarchy allows WAVE to manage large numbers of generalizations efficiently. Since so many different generalizations with differing levels of reliability can exist, WAVE can trade-off precision for breadth of coverage *without* retraining.
- The hierarchy benefits extraction. When an instance is presented for extraction, there is no need to test all rules to see if they extract from it: if a rule near the root does not extract from the instance, none of its descendants will either. Hence large sections of the hierarchy can be pruned away.
- Because the reliability of rule predictions are continually updated throughout storage, the hierarchy can be used for extraction at any time.

```
store(A, B)
1. If $A$ does not cover $B$, return failure.
2. Update reliability of each prediction in $A$ based on
   annotations in $B$.
3. Recursively store $B$ under the children of $A$.
4. If $B$ is not successfully stored under a child of $A$:
   (a) Unify $B$ with each child of $A$.
   (b) Add the valid unifications to the children of $A$.
   (c) Add $B$ to the children of $A$.
   (d) Reorganize the children of $A$ to preserve hierarchy
       invariants.
5. Return success.
```

Figure 1: A recursive algorithm to store a new instance B under an existing node A. The initial call is *store(Root, B)*.

```
extract(R, I)
1. If $R$ does not cover $I$, return failure.
2. For each prediction $p$ in $R$:
   (a) Calculate $E_p$, the error estimate for $p$.
   (b) If $E_p \leq error\ tolerance$, predict $p$ for $I$.
3. For each child $C$ of $R$, call $extract(C, I)$.
4. Return success.
```

Figure 2: A recursive algorithm to apply the hierarchy's extraction rules to a new instance, I. The initial call is $extract(Root, I)$.

The algorithm for storing instance rules in the hierarchy is given in Figure 1. Instance rules are stored in three phases: *siting*, *growing*, and *mending*. *Siting* (steps 1 through 3) finds the correct locations in the hierarchy for a new instance and updates rule reliability along the way. Valid sites are found under the most specific generalizations that cover the instance. Once sited, *growing* (steps 4a through 4c) unifies the instance with all of its siblings, creating new generalizations. Unification combines two rules by relaxing compatible constraints just enough so that the resulting rule covers the original two. Incompatible constraints are dropped. Any new generalized rules are added at the site as siblings. *Mending* (step 4d) fixes any violations by rearranging the new group of siblings.

The algorithm for extracting from instances is given in Figure 2. Error estimates for predictions are calculated using data gathered during storage.

WAVE's induction algorithm is similar to the CRYSTAL algorithm developed by Soderland. Both algorithms rely on unification to build more general extraction rules. CRYSTAL generalizes an instance rule by repeatedly unifying it with the most similar rule found in the pool of training instances. Generalization stops when either no more similar rules are found or the generalized rule exceeds a set error tolerance when tested on the training set. WAVE does not rely on error tolerance to control generalization. Instead, WAVE bounds generalization to unification with siblings in the hierarchy. CRYSTAL's reliance on an error tolerance parameter forces retraining if a different recall/precision trade-off is desired. Moreover, CRYSTAL is a batch algorithm: it requires the entire set of training instances to accurately measure the error rates of its generalized rules.

Experimental Results

Experiments were done to determine WAVE's performance relative to CRYSTAL. Comparisons are based on the metrics *recall* (R) and *precision* (P). Let *poss* be the number of concept instances available for extraction (possibles), let *corr* be the number of correct extractions, and let *errs* be the number of incorrect extractions. Then $R = \frac{corr}{poss}$ and $P = \frac{corr}{corr+errs}$. A third metric, the *F-measure* (Chinchor 1992), provides a way to measure overall performance on a concept. This metric is a weighted average of recall and precision:

$$F = \frac{(\beta^2 + 1.0) \times P \times R}{\beta^2 \times P + R}$$

where β is the relative importance of recall to precision (for all experiments here, $\beta = 1.0$, giving equal weight to recall and precision).

Experiments were conducted in two extraction domains. The first is the familiar MUC-4 Latin American terrorism (LAT) domain. The second is *hospital discharge summaries* (HDS). Summaries are dictated by a physician at the end of a patient's hospital stay, and include the cause of the hospitalization, lab data, diagnoses, symptoms reported by the patient, past medical history, and the course of treatment. Target concepts for extraction include symptoms (both present and absent) and diagnoses that were ruled-out by the physician. Results for all concepts were averaged over 10 runs. Training sets from increasing percentages of the corpus were used to determine learning curves.

Figures 3 (LAT) and 4 (HDS) compare F-measure performance at an error tolerance of 0.1 (at most 10% of extractions made by a rule on the training are errors). On LAT, performance is nearly identical except at 10% training, where CRYSTAL is slightly better. At very low training levels, CRYSTAL tends to outperform WAVE on recall: use of error tolerance to bound generalization allows it to create rules that are more general. On HDS, again performance is nearly identical. Performance at higher error tolerances is similar.

The final experiment measured the execution speed of both CRYSTAL and WAVE in a simulated incremental, dynamic application setting. Both algorithms were given 4,000 instances from the LAT domain in increments of 100 instances. To run CRYSTAL incrementally, each batch had to be reprocessed along with each new increment. The total time to process all 40 increments in this fashion was approximately 40 minutes;

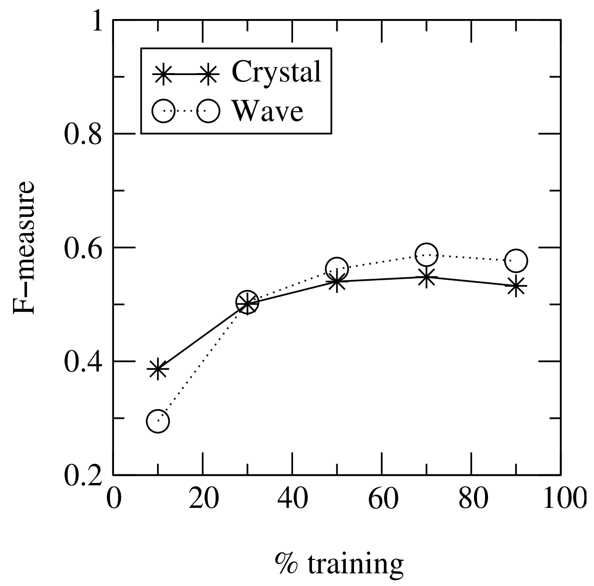

Figure 3: LAT F-measure performance.

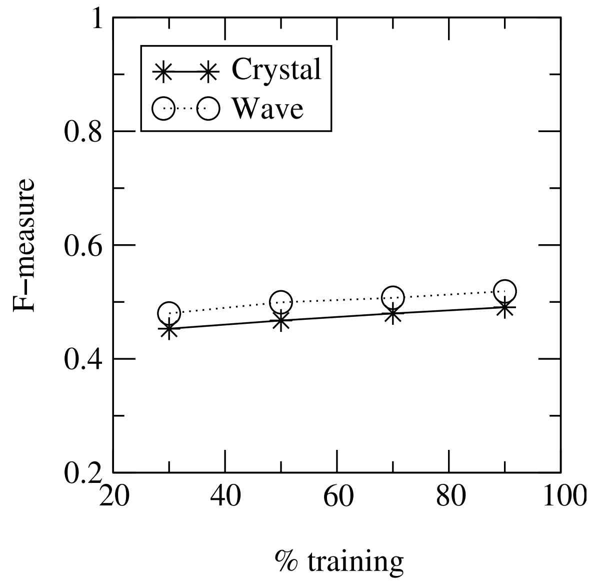

Figure 4: HDS F-measure performance.

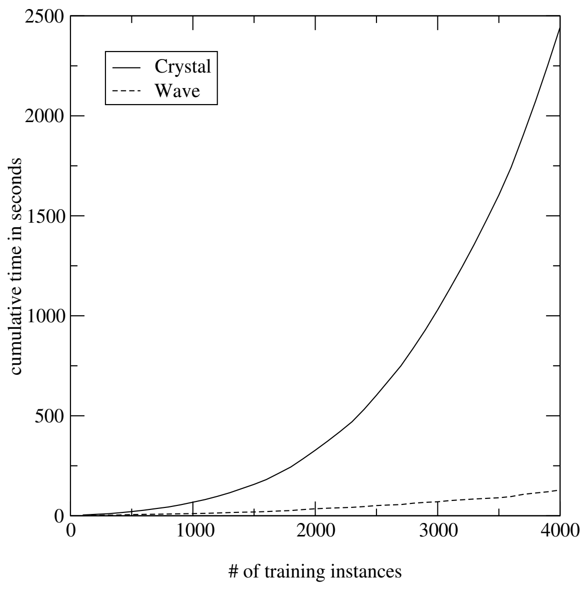

Figure 5: A comparison of cumulative execution time on a single partition.

WAVE took just over 2 minutes. Figure 5 shows the cumulative execution time at each increment. WAVE processed the instances in a fraction of the time taken by CRYSTAL, yet performed as well on F-measure.

Conclusions and Future Work

WAVE is a fully automatic, incremental induction algorithm for learning information extraction rules. Unlike traditional batch learners, WAVE learns from a stream of training instances, not a set. WAVE overcomes the inherent problems of incremental operation by maintaining a generalization hierarchy of rules. Use of a hierarchy allows similar rules to be found efficiently, provides a natural bound on generalization, enables recall/precision trade-offs without retraining, and speeds extraction since all rules need not be applied to an instance. Finally, because the reliability of rule predictions are continually updated throughout storage, the hierarchy can be used for extraction at any time.

WAVE performs as well as CRYSTAL, a related batch algorithm, in two very different domains. In a simulated incremental application setting, WAVE is significantly faster than CRYSTAL because new instances are simply added to the hierarchy; CRYSTAL needs to retrain from scratch using the old batch and the new instances.

Future work will focus on strategies for bounding the number of rules in a hierarchy. For some training partitions, the generalization hierarchy had 90,000 nodes. The question needing research is when and where to prune. One model under consideration puts an absolute bound on the number of rules that can be stored in the hierarchy. If the bound is exceeded, rules are sorted based on heuristic ratings of their utility and a sufficient number are pruned. One interesting facet of this approach is that rule reliability does *not* enter into the rating process: rules with low extraction reliability may still be useful for organization because they partition the rule space well. Rating schemes that keep the hierarchy balanced and eliminate chains of rules in which parents differ little from their descendants will be explored.

Acknowledgments

This material is based on work supported by the National Science Foundation, Library of Congress, and Department of Commerce under cooperative agreement number EEC-9209623. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the National Science Foundation, Library of Congress, or Department of Commerce.

References

Chinchor, Nancy. 1992. MUC-4 Evaluation Metrics. In *Proceedings of the Fourth Message Understanding Conference,* 22–29. Morgan Kaufmann.

Fisher, D.; Soderland, S.; McCarthy, J.; Feng, F.; and Lehnert, W. 1996. Description of the Umass System as Used for MUC-6. In *Proceedings of the Sixth Message Understand Conference,* 127-140.

Riloff, E. 1993. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence,* 811–816.

Soderland, S.; Fisher, D.; Aseltine, J.; and Lehnert, W. 1995. CRYSTAL: Inducing a Conceptual Dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* 1314–1319.