VIII

Preprocessing Applications Using Probabilistic and Hybrid Approaches

The related fields of NLP, IE, text categorization, and probabilistic modeling have developed increasingly rapidly in the last few years. New approaches are tried constantly and new systems are reported numbering thousands a year. The fields largely remain experimental science – a new approach or improvement is conceived and a system is built, tested, and reported. However, comparatively little work is done in analyzing the results and in comparing systems and approaches with each other. Usually, it is the task of the authors of a particular system to compare it with other known approaches, and this presents difficulties – both psychological and methodological.

One reason for the dearth of analytical work, excluding the general lack of sound theoretical foundations, is that the comparison experiments require software, which is usually either impossible or very costly to obtain. Moreover, the software requires integration, adjustment, and possibly training for any new use, which is also extremely costly in terms of time and human labor.

Therefore, our description of the different possible solutions to the problems described in the first section is incomplete by necessity. There are just too many reported systems, and there is often no good reason to choose one approach against the other. Consequently, we have tried to describe in depth only a small number of systems. We have chosen as broad a selection as possible, encompassing many different approaches. And, of course, the results produced by the systems are state of the art or sufficiently close to it.

VIII.1 APPLICATIONS OF HMM TO TEXTUAL ANALYSIS

VIII.1.1 Using HMM to Extract Fields from Whole Documents

Freitag and McCallum (Freitag and McCallum 1999, 2000) implemented a fields extraction system utilizing no general-purpose NLP processing. The system is designed to solve a general problem that can be specified as follows: *find the best unbroken fragment of text from a document that answers some domain-specific*

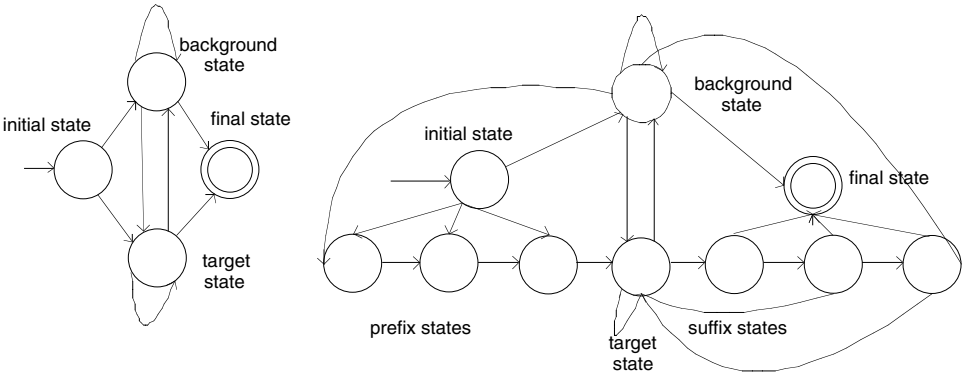

Figure VIII.1. Possible topologies of a simple HMM.

question. The question is stated implicitly in the form of a set of labeled training documents, each of them containing a single labeled field.

For example, if the domain consists of a collection of seminar announcements, we may be interested in the location of the seminar described in a given announcement. Then the training collection should contain the labeled locations. It is of course possible to extract several fields from the same document by using several separately trained models. Each model, however, is designed to extract exactly one field from one document.

The system does its task by modeling the generative process that could generate the document. The HMM model used for this purpose has the following characteristics:

- The observation symbols are the words and other tokens such as numbers.
- The HMM takes an entire document as one observation sequence.
- The HMM contains two classes of states: background states and target states. The background states emit words in which we are not interested, whereas the target states emit words that constitute the information to be extracted.
- The HMM topology is predefined and only a few transitions are allowed between the states.

The hand-built HMM topology is quite simple. One background state exists, which produces all irrelevant words. There are several prefix and suffix states, which are by themselves irrelevant but can provide the context for the target states. There are one or more parallel chains of target states – all of different lengths. And finally, there is an initial state and a final state. The topology has two variable parameters – the size of the context window, which is the number of prefix and suffix states, and the number of parallel paths of target states. Several examples of topologies are shown in Figures VIII.1 and VIII.2.

Training such HMMs does not require using the Baum–Welsh formulas because there is only one way each training document can be generated. Therefore, the maximum likelihood training for each state is conducted simply by counting the number of times each transition or emission occurred in all training sequences and dividing by the total number of times the state was visited.

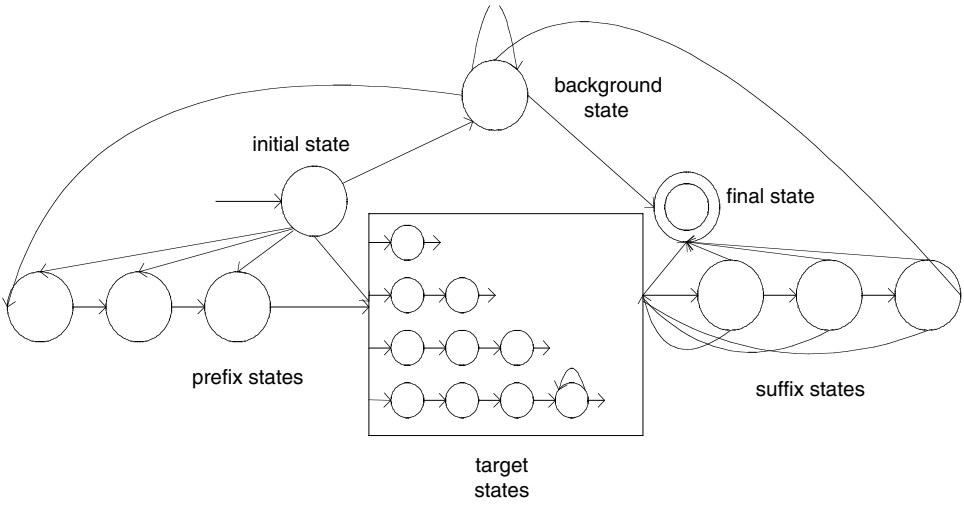

Figure VIII.2. A more general HMM topology.

The data sparseness problem, however, is severe – especially for the more complex topologies with bigger number of states. This problem is solved by utilizing the shrinkage [Crossref -> shrinkage] technique. Several possible shrinkage hierarchies were attempted. The best results were produced by shrinking straight to the simple topology shown in the left of Figure VIII.1. All prefix and suffix states are shrunk together with the background state, and all target states are also shrunk into a single target state.

This simple topology is further shrunk into a single-state HMM. The system also uses a *uniform* level, where the root single-state HMM is further shrunk into a single-state HMM with all emission probabilities equal to each other. This uniform level does the job of smoothing the probabilities by allowing previously nonencountered tokens to have a small nonzero probability. The interpolation weights for different levels were calculated by expectation maximization, using held-out data.

The system achieved some modest success in the task of extracting *speaker, location*, and *time* fields from the seminar announcements, achieving respectively 71-, 84- and 99-percent F1-measure in the best configuration, which included the window size of four as well as four parallel target paths of different sizes.

VIII.1.2 Learning HMM Structure from Data

The next work (Freitag and McCallum 2000) by the same authors explores the idea of automatically learning better HMM topologies. The HMM model works in the same way as the model described in the previous section. However, the HMM structure is not predefined and thus can be more complex. In particular, it is no longer true that every document can be generated by exactly one sequence of states. Therefore, Baum–Welsh formulas, adjusted for label constraints, are used for HMM parameter estimation.

The optimal HMM structure for a given task is built by hill climbing in the space of all possible structures. The initial simplest structure is shown in Figure VIII.3.

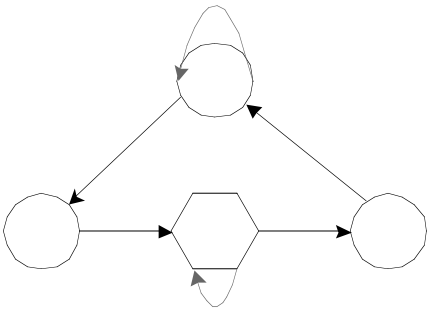

Figure VIII.3. Initial HMM topology.

At each step, each step of the following set of operations is applied to the current model:

- ***Lengthen a prefix***. A single state is added to the end of a prefix. The penultimate state now undergoes transition only to the new state; the new state changes to any target states to which the penultimate state previously changed.
- ***Split a prefix***. A duplicate is made of some prefix. Transitions are duplicated so that the first and last states of the new prefix have the same connectivity to the rest of the network as the old prefix.
- ***Lengthen a suffix***. The dual of the prefix-lengthening operation.
- ***Split a suffix***. Identical to the prefix-splitting operation except that it is applied to a suffix.
- ***Lengthen a target string***. Similar to the prefix lengthening operation, except that all target states, in contrast to prefix and suffix states, have self-transitions. The single target state in the simple model in Figure VIII.1 is a target string of length one.
- ***Split a target string***. Identical to the prefix-splitting operation except that it is applied to a target string.
- ***Add a background state***. Add a new background state to the model, with the same connectivity, with respect to the nonbackground states, as all other background states: the new state has outgoing transitions only to prefix states and incoming transitions only from suffix states.

The model performing best on a separate validation set is selected for the next iteration. After 25 iterations, the best-performing (scored by three-fold cross-validation) model is selected from the set of all intermediate models as the final model.

The experiments show that the models learned in this way usually outperform the simple hand-made models described in the previous section. For instance, in the domain of seminar announcements, the learned model achieves 77- and 87.5-percent F1-measure for the tasks of extracting *speaker* and *location* fields, respectively.

VIII.1.3 Nymble: An HMM with Context-Dependent Probabilities

A different approach was taken by BBN (Bikel et al. 1997) in the named entity extraction system *Nymble* (later called IdentiFinder). Instead of utilizing complex

HMM structures to model the complexity of the problem, Nymble uses a simple, fully connected (*ergodic*) HMM with a single-state-per-target concept and a single state for the background. However, the emission and transition probabilities of the states are not permanently fixed but depend on the context. The system achieved a very good accuracy, outperforming the handcoded rule-based systems.

Nymble contains a handcrafted tokenizer, which splits the text into sentences and the sentences into tokens. Nymble represents tokens as pairs $<w,f>$, where w is the lowercase version of the token and f is the token feature – a number from 1 to 14 according to the first matching description of the token in the following list:

1. digit number (01)
2. digit number (1996)
3. alphanumeric string (A34–24)
4. digits and dashes (12–16–02)
5. digits and slashes (12/16/02)
6. digits and comma (1,000)
7. digits and period (2.34)
8. any other number (100)
9. all capital letters (CLF)
10. capital letter and a period (M.)
11. first word of a sentence (The)
12. initial letter of the word is capitalized (Albert)
13. word in lower case (country)
14. all other words and tokens (;)

The features of the tokens are choosen in such a way as to maximize the similarities in the usage of tokens having the same feature. The Nymble model is designed to exploit those similarities. Note that the list of features depends on the problem domain and on the language. The list of features for different problems, different languages, or both, would be significantly different.

The named entity extraction task, as in MUC evaluation (Chinchor et al. 1994: MUC), is to identify all named locations, named persons, named organizations, dates, times, monetary amounts, and percentages in text. The task can be formulated as a classification problem: given a body of text, to label every word with one of the name class tags such as Person, Organization, Location, Date, Time, Money, Percent, or Not-A-Name.

Nymble utilizes an HMM model, which contains a state per each name class. There are two additional states for the beginning and the end of sentence. The HMM is fully connected (ergodic), and thus there is a nonzero probability of transition from any state to any other state. The HMM topology of Nymble is shown in Figure VIII.4.

Unlike the classical formulation, however, the transition and emission probabilities of the states in Nymble HMM depend on their context. The probability of emitting a first token in a name class is conditioned on the previous name class. The probability of emitting any other token inside a name class is conditioned on the previous token, and the probability of transition to a new name class is conditioned on the last word in the previous name class.

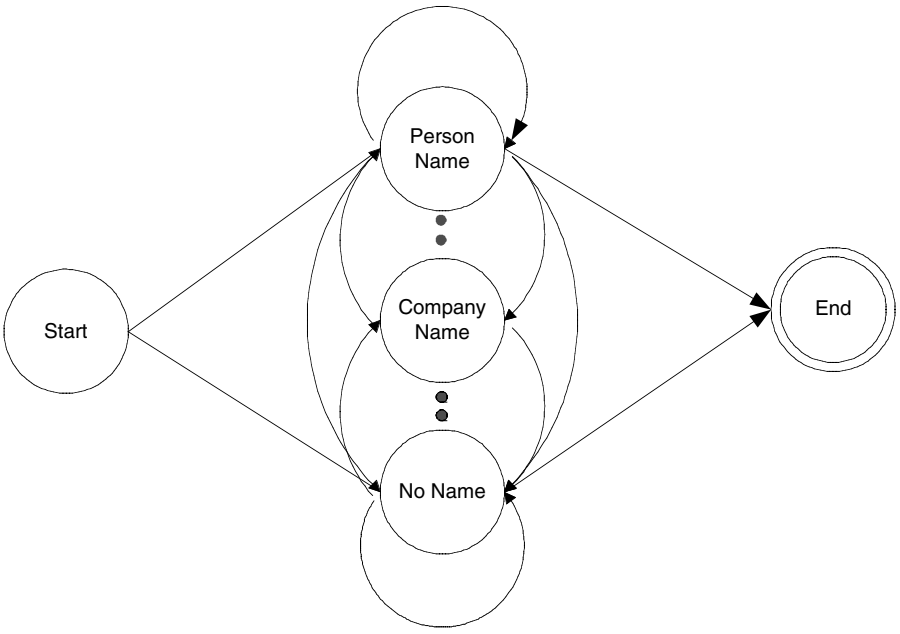

Figure VIII.4. Nymble HMM topology.

Formally, such a model can be described as a classical HMM by substituting |V| new states for each nameclass state, where V is the vocabulary of the system. Each new state will emit the token it corresponds to with the probability one, and the fixed transition probabilities between the states would then be conditioned as required. The nonstandard formulation, however, allows enormously more efficient processing, cleaner formulation of the back-off models below, and the possibility of improving the system by conditioning the probabilities on additional context clues.

As described earlier, there are three different classes of probabilities that the model must be able to estimate:

- The probability $P(<w, f> \mid \mathrm{NC}, \mathrm{NC}^{-1})$ of generating the first token in a name class (NC) conditioned on the previous name class,
- The probability $P(<w, f> \mid \mathrm{NC}, <w^{-1}, f^{-1}>)$ of generating the subsequent tokens inside a name class with each token conditioned on the previous one, and
- The probability $P(\mathrm{NC} \mid \mathrm{NC}^{-1}, w^{-1})$ of transition to a new name class conditioned on the previous word.

The model is trained by maximum likelihood. There is no need for Baum–Welsh reestimation because for each sentence there is only one way it can be generated. Thus, the probabilities above are calculated using events/sample-size. For instance,

$$P(< w,\ f > \mid \mathrm{NC}, \mathrm{NC}^{-1}) = c(< w,\ f >, \mathrm{NC}, \mathrm{NC}^{-1})/c(\mathrm{NC}, \mathrm{NC}^{-1}),$$

where the $c(\ldots)$ represents the number of occurrences of a particular event in the training data.

The training data sparseness problem manifests itself here especially as the probabilities are conditioned on context. There are two separate cases: tokens that do not appear in the training data (the unknown tokens) and other events for which the training data are insufficiently representative.

We deal with unknown token $<w, f>$ robustly by substituting for it a pair $<$_UNK_, $f>$ having the same feature and a new _UNK_ word. Statistics for the unknown tokens are gathered in a separate model built specifically for dealing with them. The model is trained in the following way: The whole training set is divided into two halves. Then the tokens in the first half that do not appear in the second and the tokens in the second half that do not appear in the first are substituted by the _UNK_ tokens. The unknown words model is trained on the resulting dataset. In this way, all of the training data participate.

For dealing with the general data sparseness problem, several layers of backoff are employed:

- The probability of generating the first word in a name class $P(<w, f> \mid \mathrm{NC}, \mathrm{NC}^{-1})$ is interpolated with $P(<w, f> \mid \mathrm{NC} <\mathrm{any}>)$ and further with $P(<w, f> \mid \mathrm{NC})$, with $P(w \mid \mathrm{NC}) \cdot P(f \mid \mathrm{NC})$, and with $|\mathrm{V}|^{-1}|\mathrm{F}|^{-1}$.
- The probability of generating subsequent tokens $P(<w, f> \mid \mathrm{NC}, <w^{-1}, f^{-1}>)$ is interpolated with $P(<w, f> \mid \mathrm{NC})$, with $P(w \mid \mathrm{NC}) \cdot P(f \mid \mathrm{NC})$, and with $|\mathrm{V}|^{-1}|\mathrm{F}|^{-1}$.
- The transition probability $P(\mathrm{NC} \mid \mathrm{NC}^{-1}, w^{-1})$ is interpolated with $P(\mathrm{NC} \mid \mathrm{NC}^{-1})$, with $P(\mathrm{NC})$, and with 1/(number of name classes).

The weights for each back-off model are computed on the fly, using the following formula:

$$\lambda = \left(1 - \frac{c(Y)}{bc(Y)}\right) \frac{1}{1 + \frac{\#(Y)}{bc(Y)}},$$

where $c(Y)$ the count of event Y according to the full model, $bc(Y)$ is the count of event Y according to the backoff model, and $\#(Y)$ is the number of unique outcomes of Y. This λ has two desirable properties. If the full model and the backoff have similar levels of support for an event Y, then the λ will be close to zero and the full model will be used.

The number of unique outcomes is a crude measure of uniformity, or uncertainty, of the model. The more uncertainty the model has, the lower is the confidence in the backoff model, the lower λ is then used.

The experimental evaluation of the Nymble system showed that, given sufficient training, it performs comparably to the best hand-crafted systems (94.9% versus 96.4% F1-measure) for the mixed-case English *Wall Street Journal* documents and significantly outperforms them for the more difficult all-uppercase and speech-form (93.6% and 90.7% versus 89% and 74%, respectively).

VIII.2 USING MEMM FOR INFORMATION EXTRACTION

Very recently the conditional models trained using the maximal entropy approach received much attention. The reason for preferring them over the more traditional generative models lies in their ability to make use of arbitrary features of the

observations, possibly overlapping and interdependent, in a consistent and mathematically clean way.

The MEMM is one formalism developed in McCallum et al. (2000) that allows the power of the ME approach to be used. They tested their implementation of MEMMs on the problem of labeling the lines in a long multipart FAQ file according to their function as a *head*, a *question*, an *answer*, and a *tail*.

The problem is especially well suited for a conditional model because such a model can consider each line a single observation unit described by its features. In contrast, a generative model like HMM would have to generate the whole line (i.e., to estimate its probability), which is clearly infeasible.

The 24 binary features (trigger constraint functions) used for classifying lines in the particular problem are shown below:

begins-with-number	contains-question-mark
begins-with-ordinal	contains-question-word
begins-with-punctuation	ends-with-question-mark
begins-with-question-word	first-alpha-is-capitalized
begins-with-subject	indented
blank	indented-1-to-4
contains-alphanum	indented-5-to-10
contains-bracketed-number	more-than-one-third-space
contains-http	only-punctuation
contains-non-space	prev-is-blank
contains-number	prev-begins-with-ordinal
contains-pipe	shorter-than-30

As can be seen, the features of a line do not define the line completely, nor are they independent.

The MEMM was compared with three other learners:

- Stateless ME classifier, which used the 24 features to classify each line separately.
- Traditional, fully connected HMM with four states emitting individual tokens. Similar four-state HMM emitting individual features.
- Each line was converted to a sequence of features before training and testing.

It was found that MEMM performed best of all four, and Feature HMM was second but had significantly worse performance. The other two models functioned poorly.

VIII.3 APPLICATIONS OF CRFs TO TEXTUAL ANALYSIS

VIII.3.1 POS-Tagging with Conditional Random Fields

CRFs were developed in Lafferty et al. (2001) as a conditional ME–based version of HMM, which does not suffer from label bias problems. Lafferty et al. applied the CRF formalism to POS tagging in Penn treebank style and compared its performance with that of HMM and MEMM.

In the first set of experiments, the two types of features were introduced – *tag–word* pairs, and *tag–tag* pairs corresponding to HMM observation and transition features. The results are consistent with the expectations: HMM outperforms MEMM as

a consequence of the label bias problem, whereas CRF and HMM perform similarly with CRF slightly better overall but slightly worse for out-of-vocabulary words.

In the second set of experiments, a set of simple morphological features was added: whether a word begins with a digit or uppercase letter, whether it contains a hyphen, and whether it ends in one of the following suffixes: -ing -ogy -ed -s -ly -ion -tion -ity -ies. Here the results also confirm the expectations: Both CRF and MEMM benefit significantly from the use of these features – especially for out-of-vocabulary words.

VIII.3.2 Shallow Parsing with Conditional Random Fields

Shallow parsing is another sequence labeling problem. The task is to identify the nonrecursive cores of various types of phrases. The paradigmatic shallow parsing problem is *NP chunking*, finding the nonrecursive cores of noun phrases, the *base NPs*. Sha and Pereira (2003) adapt CRFs to this problem and show that it beats all known single-model NP chunkers, performing at the level of the best known chunker – voting arrangement of 24 forward- and backward-looking SVM classifiers.

The input to an NP chunker consists of a sentence labeled with POS tags. The chunker's task is to further label each word indicating whether the word is (O)utside the chunk, (B)egins a chunk, or (C)ontinues a chunk.

The chunking CRF in Sha and Pereira (2003) has a second-order Markov dependency between chunk tags. This is encoded by making the labels of CRF pairs of consecutive chunk tags. That is, the label at position i is $y_i = c_{i-1}c_i$ where c_i is the chunk tag of word i, one of O, B, or C. Because B must be used to start a chunk, the label OC is impossible. In addition, successive labels are constrained. These constraints on the model topology are enforced by giving appropriate features a weight of $-\infty$, forcing all the forbidden labelings to have zero probability.

The features of the chunker CRF are represented as

$$f(\boldsymbol{x}, \boldsymbol{y}, i) = g(\boldsymbol{x}, i)h(y_i, y_{i+1}),$$

where $g(x, i)$ is a predicate on the input sequence and position, and $h(y_i\ y_{i+1})$ is a predicate on pairs of labels. The possibilities for the predicates are as follows:

g(x, i)	true	
	w_i=w	
	w_{i-1}=w	w_{i+1}=w
	w_{i-2}=w	w_{i+2}=w
	(w_i=w) and (w_{i-1}=w′)	(w_i=w) and (w_{i+1}=w′)
	t_i=t	
	t_{i-1}=t	t_{i+1}=t
	t_{i-2}=t	t_{i+2}=t
	(t_i=t) and (t_{i-1}=t′)	(t_i=t) and (t_{i+1}=t′)
	(t_{i-1}=t) and (t_{i-2}=t′)	(t_{i+1}=t) and (t_{i+2}=t′)
	(t_i=t) and (t_{i-1}=t′) and (t_{i-2}=t″)	
	(t_i=t) and (t_{i-1}=t′) and (t_{i+1}=t″)	
	(t_i=t) and (t_{i+1}=t′) and (t_{i+2}=t″)	
h(y_i, y_{i+1})	y_i=y	
	(y_i=y) and (y_{i+1}=y′)	
	c(y_i)=c.	

The w_i, t_i, y_i mean, respectively, the word, the POS tag, and the label at position i; $c(y_i)$ means the chunk tag, and thus $c(\text{OB}) = \text{B}$. The $w, w', t, t', t'', y, y', c$ are specific words, tags, labels, and chunk tags chosen from the vocabulary generated by the training data.

A Gaussian weight prior was used to reduce overfitting, and thus the log-likelihood of the training data was taken as

$$L(\boldsymbol{\lambda}) = \Sigma_k[\boldsymbol{\lambda} \cdot \mathbf{F}(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) - \log Z_{\boldsymbol{\lambda}}(\mathbf{x}^{(k)})] - ||\boldsymbol{\lambda}||^2/2\sigma^2.$$

The experimental evaluation demonstrates the state-of-the-art performance of the CRF chunk tagger. Interestingly, the GIS training method was shown to perform less well than some other general-purpose convex optimization algorithms – especially when many correlated features are involved. The convergence rate of GIS turns out to be much slower.

VIII.4 TEG: USING SCFG RULES FOR HYBRID STATISTICAL–KNOWLEDGE-BASED IE

Another approach has been described that employs a hybrid statistical and knowledge-based information extraction model able to extract entities and relations at the sentence level. The model attempts to retain and improve the high accuracy levels of knowledge-based systems while drastically reducing the amount of manual labor by relying on statistics drawn from a training corpus. The implementation of the model, called trainable extraction grammar (TEG), can be adapted to any IE domain by writing a suitable set of rules in a SCFG-based extraction language and training them using an annotated corpus.

The system does not contain any purely linguistic components such as a POS tagger or parser. We demonstrate the performance of the system on several named entity extraction and relation extraction tasks. The experiments show that our hybrid approach outperforms both purely statistical and purely knowledge-based systems and require orders-of-magnitude less manual rule writing and smaller amounts of training data. The improvement in accuracy is slight for named entity extraction tasks and more pronounced for relation extraction.

By devoting some attention to the details of TEG, we can provide a concrete sense of how hybrid-type systems can be employed for text mining preprocessing operations.

VIII.4.1 Introduction to a Hybrid System

The knowledge engineering (mostly rule-based) systems traditionally were the top performers in most IE benchmarks such as MUC (Chinchor, Hirschman, and Lewis 1994), ACE (ACE 2004), and the KDD CUP (Yeh and Hirschman 2002). Recently, though, the machine learning systems became state of the art – especially for simpler tagging problems such as named entity recognition (Bikel, Schwartz, and Weischedel 1999) or field extraction (McCallum et al. 2000).

Still, the knowledge engineering approach retains some of its advantages. It is focused around manually writing patterns to extract the entities and relations. The patterns are naturally accessible to human understanding and can be improved in a controllable way, but improving the results of a pure machine learning system would

require providing it with additional training data. However, the impact of adding more data soon becomes infinitesimal, whereas the cost of manually annotating the data grows linearly.

TEG is a hybrid entities and relations extraction system, which combines the power of knowledge-based and statistical machine learning approaches. The system is based on SCFGs. The rules for the extraction grammar are written manually, and the probabilities are trained from an annotated corpus. The powerful disambiguation ability of PCFGs allows the knowledge engineer to write very simple and naive rules while retaining their power, thus greatly reducing the required labor.

In addition, the size of the needed training data is considerably smaller than that of the training data needed for a pure machine learning system (for achieving comparable accuracy results). Furthermore, the tasks of rule writing and corpus annotation can be balanced against each other.

VIII.4.2 TEG: Bridging the Gap between Statistical and Rule-Based IE Systems

Although the formalisms based on probabilistic finite-state automata are quite successful for entity extraction, they have shortcomings that make them harder to use for the more difficult task of extracting relationships.

One problem is that a finite-state automaton model is flat, and so its natural task is assignment of a tag (state label) to each token in a sequence. This is suitable for the tasks in which the tagged sequences do not nest and there are no explicit relations between the sequences. Part-of-speech tagging and entity extraction tasks belong to this category, and indeed the HMM-based POS taggers and entity extractors are state of the art.

Extracting relationships is different because the tagged sequences can and must nest and there are relations between them, which must be explicitly recognized. Although it is possible to use nested automata to cope with this problem, we felt that using a more general context-free grammar formalism would allow for greater generality and extendibility without incurring any significant performance loss.

VIII.4.3 Syntax of a TEG Rulebook

A TEG rulebook consists of declarations and rules. Rules basically follow the classical grammar rule syntax with a special construction for assigning concept attributes. Notation shortcuts like [ ] and | can be used for easier writing. The nonterminals referred by the rules must be declared before usage. Some of them can be declared as *output concepts*, which are the entities, events, and facts that the system is designed to extract. Additionally, two classes of terminal symbols also require declaration: *termlists* and *ngrams*.

A termlist is a collection of terms from a single semantic category written either explicitly or loaded from external source. Examples of termlists are countries, cities, states, genes, proteins, people's first names, and job titles. Some linguistic concepts such as lists of propositions can also be considered termlists. Theoretically, a termlist is equivalent to a nonterminal symbol that has a rule for every term.

An ngram is a more complex construction. When used in a rule, it can expand to any single token. The probability of generating a given token, however, is not fixed in the rules but learned from the training dataset and may be conditioned on one or more previous tokens. Thus, using ngrams is one of the ways the probabilities of TEG rules can be context-dependent. The exact semantics of ngrams is explained in the next section.

Let us see a simple meaningful example of a TEG grammar:

```
output concept Acquisition(Acquirer, Acquired);
ngram AdjunctWord;
nonterminal Adjunct;
Adjunct:- AdjunctWord Adjunct | AdjunctWord;
termlist AcquireTerm = acquired bought (has acquired) (has bought);
Acquisition :- Company → Acquirer [","Adjunct ","]
AcquireTerm
Company → Acquired;
```

The first line defines a target relation **Acquisition**, which has two attributes, **Acquirer** and **Acquired**. Then an ngram **AdjunctWord** is defined followed by a nonterminal **Adjunct**, which has two rules separated by "|" that together define **Adjunct** as a sequence of one or more **AdjunctWord**-s. Then a termlist **AcquireTerm** is defined containing the main acquisition verb phrase. Finally, the single rule for the **Acquisition** concept is defined as a **Company** followed by optional **Adjunct** delimited by commas that are followed by **AcquireTerm** and a second **Company**. The first **Company** is the **Acquirer** attribute of the output frame and the second is the **Acquired** attribute.

The final rule requires the existence of a defined **Company** concept. The following set of definitions identifies the concept in a manner emulating the behavior of an HMM entity extractor:

```
output concept Company();
ngram CompanyFirstWord;
ngram CompanyWord;
ngram CompanyLastWord;
nonterminal CompanyNext;
Company:- CompanyFirstWord CompanyNext |
CompanyFirstWord;
CompanyNext:- CompanyWord CompanyNext |
CompanyLastWord;
```

Finally, in order to produce a complete grammar, we need a starting symbol and the special nonterminal that would match the strings that do not belong to any

of the output concepts:

```
start Text;
nonterminal None;
ngram NoneWord;
None:- NoneWord None | ;
Text:- None Text | Company Text | Acquisition Text;
```

These 20 lines of code are able to find a fair number of Acquisitions accurately after very modest training. Note that the grammar is extremely ambiguous. An ngram can match any token, and so **Company**, **None**, and **Adjunct** are able to match any string. Yet, using the learned probabilities, TEG is usually able to find the correct interpretation.

VIII.4.4 TEG Training

Currently there are three different classes of trainable parameters in a TEG rulebook: the probabilities of rules of nonterminals, the probabilities of different expansions of ngrams, and the probabilities of terms in a wordclass. All those probabilities are smoothed maximum likelihood estimates calculated directly from the frequencies of the corresponding elements in the training dataset.

For example, suppose we have the following simple TEG grammar that finds simple person names:

```
nonterm start Text;
concept Person;
ngram NGFirstName;
ngram NGLastName;
ngram NGNone;
termlist TLHonorific = Mr Mrs Miss Ms Dr;
(1) Person :- TLHonorific NGLastName;
(2) Person :- NGFirstName NGLastName;
(3) Text :- NGNone Text;
(4) Text :- Person Text;
(5) Text :-;
```

By default, the initial untrained frequencies of all elements are assumed to be 1. They can be changed using "<count>" syntax, an example of which is shown below. The numbers in parentheses on the left side are not part of the rules and are used only for reference. Let us train this rulebook on the training set containing one sentence:

```
Yesterday, <person> Dr Simmons, </person> the distinguished scientist, presented the discovery.
```

The difference is in the expansion of the **Person** nonterminal. Both **Person** rules can produce the output instance; therefore, there is an ambiguity. This is done in two steps. First, the sentence is parsed using the untrained rulebook but with the constraints specified by the annotations. In our case the constraints are satisfied by two different parses that are shown in Figure VIII.5 (the numbers below the nonterminals refer to the rules used to expand them):

The ambiguity arises because both **TLHonorific** and **NGFirstName** can generate the token "Dr." In this case the ambiguity is resolved in favor of the **TLHonorific** interpretation because in the untrained rulebook we have

---

P (Dr | TLHonorific) = 1/5
(choice of one term among five equiprobable ones),
P (Dr | NGFirstName) $\approx$ 1/N, where N is the number of all known words (untrained ngram behavior).

After the training, the frequencies of the different elements are updated, which produces the following trained rulebook (only lines that were changed are shown). Note the "<Count>" syntax:

```
termlist TLHonorific = Mr Mrs Miss Ms <2> Dr;
Person :- <2>TLHonorific NGLastName;
Text :- <11>NGNone Text;
Text :- <2>Person Text;
Text :- <2>;
```

Additionally, the training will generate a separate file containing the statistics for the ngrams. It is similar but more complex because the bigram frequencies, token feature frequencies, and unknown word frequencies are taken into consideration. In order to understand the details of ngrams training it is necessary to go over the details of their internal working.

An ngram always generates a single token. Any ngram can generate any token, but naturally the probability of generating one depends on the ngram, on the token, and on the immediate preceding context of the token. This probability is calculated at the runtime using the following statistics:

- $Freq(*)$ = total number of times the ngram was encountered in the training set.
- $Freq(W)$, $Freq(F)$, $Freq(T)$ = number of times the ngram was matched to the word W, the feature F, and the token T, respectively. Note that a token T is a pair consisting of a word $W(T)$ and its feature $F(T)$.
- $Freq(T \mid T_2)$ = number of times token T was matched to the ngram in the training set and the preceding token was T_2.
- $Freq(* \mid T_2)$ = number of times the ngram was encountered after the token T_2.

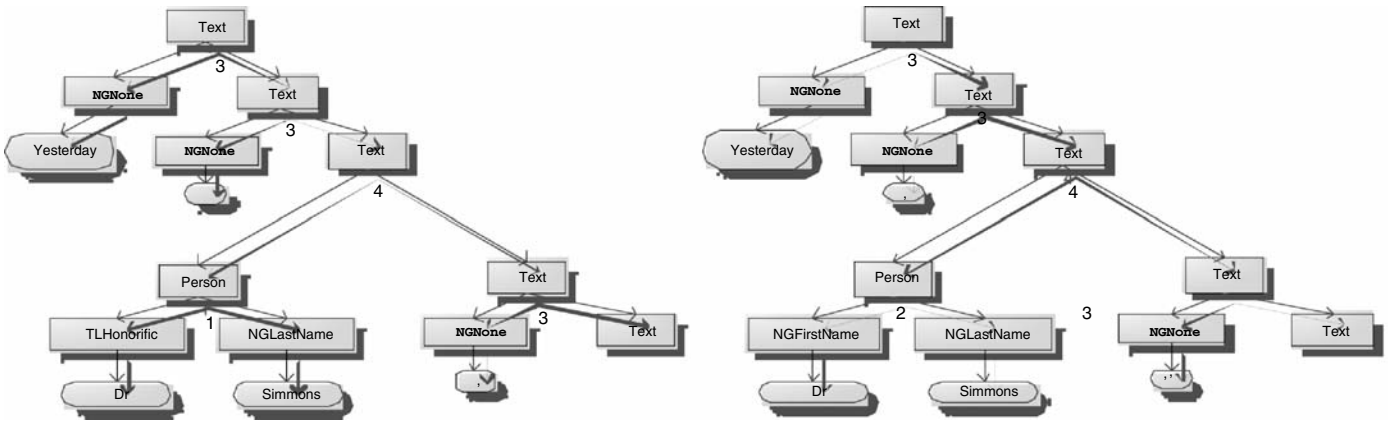

Figure VIII.5. Possible parse trees.

Thus, on the assumption all those statistics are gathered, the probability of the ngram's generating a token T given that the preceding token is T_2 is estimated as

$$\begin{aligned} P(T|T_2) = {} & 1/2 \cdot Freq(T|T_2)/Freq(^*|T_2) \\ & + 1/4 \cdot Freq(T)/Freq(^*) \\ & + 1/4 \cdot Freq(W) \cdot Freq(F)/Freq(^*)^2. \end{aligned}$$

This formula linearly interpolates between the three models: the bigram model, the backoff unigram model, and the further backoff word+feature unigram model. The interpolation factor was chosen to be 1/2, which is a natural choice. The experiments have shown, however, that varying the λ's in reasonable ranges does not significantly influence the performance.

Finally, matters are made somewhat more complicated by the *unknown* tokens. That a token was never encountered during the training gives by itself an important clue to the token's nature. In order to be able to use this clue, the separate "unknown" model is trained. The training set for it is created by dividing the available training data into two halves and treating one-half of the tokens, which are not present in the other half, as special "unknown" tokens. The model trained in this way is used whenever an unknown token is encountered during runtime.

VIII.4.5 Additional features

There are several additional features that improve the system and help to customize it for other domains. First, the probabilities of different rules of a nonterminal need not be fixed but may depend on their context. Currently, the rules for a specific nonterminal can be conditioned on the previous token in a way similar to the dependencey of ngram probabilities on the previous token. Other conditioning is of course possible – even to the extent of using maximal entropy for combining several conditioning events.

Second, an external tokenizer, token feature generator, or both can be substituted for the regular one. It is even possible to use several feature generators simultaneously (different ngrams may use different token feature sets). This is useful for languages other than English as well as for special domains. For instance, in order to extract the names of chemical compounds or complex gene names it may be necessary to provide a feature set based on morphological features. In addition, an external part-of-speech tagger or shallow parser may be used as a feature generator.

For real-life IE tasks it is often necessary to extract very rare target concepts. This is especially true for relations. Although there could be thousands of Persons or Organizations in a dataset, the number of Acquisitions could well be less than 50. The ngrams participating in the rules for such concepts will surely be undertrained. In order to alleviate this problem, the *shrinkage* technique can be used. An infrequent specific ngram can be set to *shrink* to another more common and more general ngram. Then the probability of generating a token by the ngram is interpolated with the corresponding probability for the more common "parent" ngram. A similar technique was used with a great success for HMM, and we found it very useful for TEG as well.

VIII.4.6 Example of Real Rules

This section demonstrates a fragment of the true rules written for the extraction of the PersonAffiliation relation from a real industry corpus. The fragment shows a usage of the advanced features of the system and gives another glimpse of the flavor of rule writing in TEG.

The PersonAffiliation relation contains three attributes – name of the person, name of the organization, and position of the person in the organization. It is declared as follows:

concept output PersonAffiliation(Name, Position, Org);

Most often, this relation is encountered in the text in the form "Mr. Name, Position of Org" or "Org Position Ms. Name." Almost any order of the components is possible with commas and prepositions inserted as necessary. Also, it is common for Name, Position, or both to be conjunctions of pairs of corresponding entities: "Mr. Name1 and Ms. Name2, the Position1 and Position2 of Org," or "Org's Position1 and Position2, Ms. Name." In order to catch those complexities, and for general simplification of the rules, we use several auxiliary nonterms: **Names**, which catches one or two Names; **Positions**, which catches one or two Positions; and **Orgs**, which catches Organizations and Locations. These can also be involved in PersonAffiliation as in "Bush, president of US":

```
nonterms Names, Positions, Orgs;
Names :- PERSON->Name | PERSON->Name "and" PERSON->Name;
Positions :- POSITION->Position | POSITION->Position "and"
  POSITION-> Position;
Orgs :- ORGANIZATION->Org | LOCATION->Org;
```

We also use auxiliary nonterms that catch pairs of attributes:

PosName, and **PosOrg**:

```
nonterms PosName, PosOrg;
PosName :- Positions Names | PosName "and" PosName;
wordclass wcPreposition = "at" "in" "of" "for" "with";
wordclass wcPossessive = ("'" "" "s") "'" "";
PosOrg :- Positions wcPreposition Orgs;
PosOrg :- Orgs [wcPossessive] Positions;
```

Finally, the PersonAffiliation rules are as follows:

```
PersonAffiliation :- Orgs [wcPossessive] PosName;
PersonAffiliation :- PosName wcPreposition Orgs;
PersonAffiliation :- PosOrg [","] Names;
PersonAffiliation :- Names "," PosOrg;
PersonAffiliation :- Names "is" "a" PosOrg;
```

The rules above catch about 50 percent of all PersonAffiliation instances in the texts. Other instances depart from the form above in several respects. Thus, in order to improve the accuracy, additional rules need to be written. First, the Organization name is often entered into a sentence as a part of a descriptive noun phrase as in "Ms. Name is a Position of the industry leader Org." To catch this in a general way, we define an **OrgNP** nonterm, which uses an external POS tagger:

```
ngram ngOrgNoun featureset ExtPoS restriction Noun;
ngram ngOrgAdj featureset ExtPoS restriction Adj;
ngram ngNum featureset ExtPoS restriction Number;
ngram ngProper featureset ExtPoS restriction ProperName;
ngram ngDet featureset ExtPoS restriction Det;
ngram ngPrep featureset ExtPoS restriction Prep;

nonterm OrgNounList;
OrgNounList :- ngOrgNoun [OrgNounList];
nonterms OrgAdjWord, OrgAdjList;
OrgAdjWord :- ngOrgAdj | ngNum | ngProper;
OrgAdjList :- OrgAdjWord [OrgAdjList];
nonterm OrgNP;
OrgNP :- [ngDet] [OrgAdjList] OrgNounList;
OrgNP :- OrgNP ngPrep OrgNP;
OrgNP :- OrgNP "and" OrgNP;
```

The external POS tagger provides an alternative token feature set, which can be used by ngrams via the ngram *featureset* declaration. The *restriction* clause in the ngram declaration specifies that the tokens matched by the ngram must belong to the specified feature. Altogether, the set of rules above defines an **OrgNP** nonterm in a way similar to the to the definition of a noun phrase by a syntax-parsing grammar. To use the nonterm in the rules, we simply modify the **Orgs** nonterm:

```
Orgs :- [OrgNP] ORGANIZATION->Org | LOCATION->Org;
```

Note that, although OrgNP is internally defined very generally (it is able to match any noun phrase whatsoever), the way it is used is very restricted. During training, the ngrams of OrgNP learn the distributions of words for this particular use, and, during the run, the probability that OrgNP will generate a true organization-related noun phrase is much greater than for any other noun phrase in text.

Finally, we demonstrate the use of ngram shrinkage. There are PersonAffiliation instances in which some irrelevant sentence fragments separate the attributes. For example, "'ORG bla bla bla', said the company's Position Mr. Name." In order to catch the "bla bla bla" part we can use the **None** nonterm, which generates all irrelevant fragments in the text. Alternatively, we can create a separate ngram and a nonterm for the specific use of catching irrelevant fragments inside PersonAffiliation. Both these solutions have their disadvantages. The None nonterm is too general and does not catch the specifics of the particular case. A specific nonterm, on the other hand, is very much undertrained. The solution is to use a specific nonterm but to shrink its ngram to None:

```
nonterm BlaBla;
ngram ngBlaBla -> ngNone;
BlaBla :- ngBlaBla [BlaBla];
PersonAffiliation :- Orgs BlaBla PosName;
```

The rules described above catch 70 percent of all PersonAffiliation instances, which is already a good result for relationship extraction from a real corpus. The process of writing rules, moreover, can be continued to further improve the accuracy.

	HMM			Emulation using TEG Manual Rules						Full TEG system		
	Recall	Prec	F1	Recall	Prec	F1	Recall	Prec	F1	Recall	Prec	F1
Person	86.91	85.1	**86.01**	86.31	86.83	**86.57**	81.32	93.75	**87.53**	93.75	90.78	**92.24**
Organization	87.94	89.8	**88.84**	85.94	89.53	**87.7**	82.74	93.36	**88.05**	89.49	90.9	**90.19**
Location	86.12	87.2	**86.66**	83.93	90.12	**86.91**	91.46	89.53	**90.49**	87.05	94.42	**90.58**

Figure VIII.6. Accuracy results for MUC-7.

VIII.4.7 Experimental Evaluation of TEG

The TEG techniques were evaluated using two corpora: MUC-7 and ACE-2. The results show the potential of utilizing hybrid approaches for text mining preprocessing.

The MUC-7 Corpus Evaluation – Comparison with HMM-based NER

The MUC-7 named-entity recognition (NER) corpus consists of a set of news articles related to aircraft accidents, containing about 200 thousand words with the named entities manually categorized into three basic categories: PERSON, ORGANIZATION, and LOCATION. Some other entities are also tagged such as dates, times, and monetary units, but they did not take part in our evaluation.

The corpus does not contain tagged relationships, and thus it was used to evaluate the difference in the performance between the four entity extractors: the regular HMM, its emulation using TEG, a set of handcrafted rules written in DIAL, and a full TEG system, which consists of the HMM emulation augmented by a small set of handcrafted rules (about 50 lines of code added).

The results of the experiments are summarized in Figure VIII.6: The small accuracy difference between the regular HMM and its emulation is due to slight differences in probability conditioning methods. It is evident that the handcrafted rules performed better than the HMM-based extractors but were inferior to the performance of the TEG extractor. Significantly, the handcrafted rules achieved the best precision; however, their recall was far worse.

The HMM named-entity recognition results published in Bikel et al. (1997) are somewhat higher than we were able to produce using our version of an HMM entity extractor. We hypothesize that the reason for the difference is the use of additional training data in the Nymble experiments. The paper (Bikel et al. 1997) mentions using approximately 750K words of training data, whereas we had only 200K. Regardless of the reasons for the difference, the experiment clearly shows that the addition of a small number of handcrafted rules can further improve the results of a purely automatic HMM-based named-entity extraction.

ACE-2 Evaluation: Extracting Relationships

The ACE-2 was a follow-up to ACE-1 and included tagged relationships in addition to tagged entities. The ACE-2 annotations are more complex than those supported by the current version of our system. Most significantly, the annotations resolve all anaphoric references, which is outside the scope of the current implementation. Therefore, it was necessary to remove annotations containing anaphoric references. This was done automatically using a simple Perl script.

	Full TEG system (with 7 ROLE rules)		
	Recall	Prec	F
Role	83.44	77.30	**80.25**
Person	89.82	81.68	**85.56**
Organization	59.49	71.06	**64.76**
GPE	88.83	84.94	**86.84**

	HMM entity extractor			Markovian SCFG		
	Recall	Prec	F	Recall	Prec	F
Role				67.55	69.86	**68.69**
Person	85.54	83.22	**84.37**	89.19	80.19	**84.45**
Organization	52.62	64.735	**58.05**	53.57	67.46	**59.71**
GPE	85.54	83.22	**84.37**	86.74	84.96	**85.84**

Figure VIII.7. Accuracy results for ACE-2.

For evaluating relationship extraction we choose the ROLE relation (ACE 2002). The original ACE-2 annotations make finer distinctions between the different kinds of ROLE, but for the current evaluation we felt it sufficient just to recognize the basic relationships and find their attributes.

The results of this evaluation are shown in Figure VIII.7. For comparison we also show the performance of the HMM entity extractor on the entities in the same dataset.

As expected, the accuracy of a purely Markovian SCFG without additional rules is rather mediocre. However, by adding a small number of handcrafted rules (altogether about 100 lines of code), accuracy was raised considerably (by 15% in F1). The performances of the three systems on the named entities differ very little because they are essentially the same system. The slight improvement of the full TEG system is due to better handling of the entities that take part in ROLEs.

In Figure VIII.8 we can see how the accuracy of the TEG system changes as a function of the amount of available training data. There are three graphs in the figure: a graph that represents the accuracy of the grammar with no specific ROLE rules, a graph that represents the accuracy of the grammar with four ROLE rules, and finally a graph that represents the accuracy of the grammar with seven ROLE rules.

Analysis of the graphs reveals that, to achieve about 70-percent accuracy the system needs about 125K of training data when using all of the specific ROLE rules, whereas 250k of training data are needed when no specific rules are present. Thus, adding a small set of simple rules may save 50 percent of the training data requirements.

The seven ROLE rules used by the third TEG are shown below. The rules use nonterminals and wordclasses, which are defined in the rest of the grammar. The whole grammar, which has a length of about 200 lines, is too long to be included here.

```
1. ROLE :- [Position_Before] ORGANIZATION->ROLE_2
   Position ["in" GPE] [","] PERSON→ROLE_1;
2. ROLE :- GPE→ROLE_2 Position [","]
   PERSON→ROLE_1;
3. ROLE :-PERSON→ROLE_1 "of" GPE→ROLE_2;
```

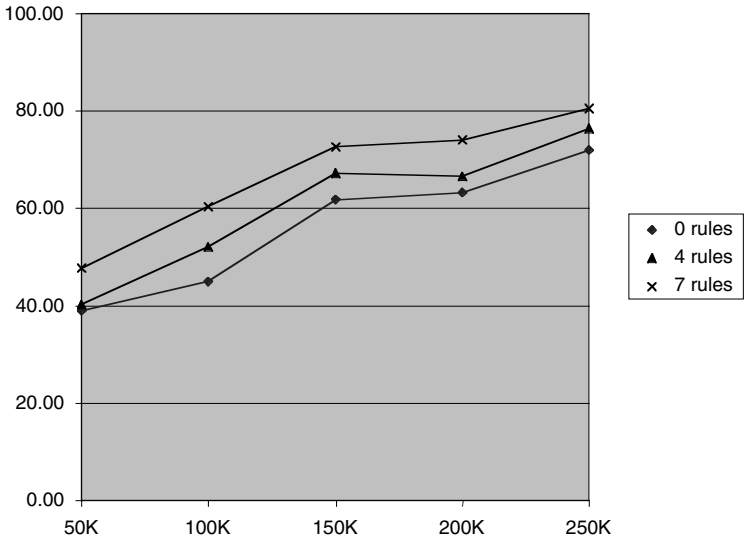

Figure VIII.8. Accuracy (F1) of the TEG system (with different grammars) as a function of the size of the training corpus (ACE-2).

```
4. ROLE :- ORGANIZATION→ROLE_2 “”” “s” [Position]
   PERSON→ROLE_1;
5. ROLE :- GPE→ROLE_2 [Position] PERSON→ROLE_1;
6. ROLE :- <5> GPE->ROLE_2 “”” “s””
   ORGANIZATION→ROLE_1;
ROLE :- PERSON→ROLE_1 “,” Position WCPreposition
   ORGANIZATION→ROLE_2;
```

VIII.5 BOOTSTRAPPING

VIII.5.1 Introduction to Bootstrapping: The AutoSlog-TS Approach

One of the main problems of the machine learning–based systems is that they rely on annotated corpora. A *bootstrapping* approach to IE takes a middle ground between the knowledge engineering and machine learning approaches. The main idea behind this approach is that the user provides some initial bias either by supplying a small initial lexicon or a small number of rules for inducing the initial examples. The bootstrapping approach attempts to circumvent the need for an annotated corpus, which can be very expensive and time consuming to produce.

One of the first approaches to bootstrapping was developed by Ellen Riloff and implemented in the *AutoSlog-TS* system (Riloff 1996a). Based on the original *AutoSlog*system developed previously by Riloff (Riloff 1993a), AutoSlog-TS uses a set of documents split into two bins: interesting documents and noninteresting documents. In contrast, the original AutoSlog required all relevant noun phrases within the training corpus to be tagged and, hence, put a much bigger load on the task of the training corpus construction. *Palka* (Kim and Moldovan 1995) was another system similar to AutoSlog, but it required a much heavier tagging in the training corpus:

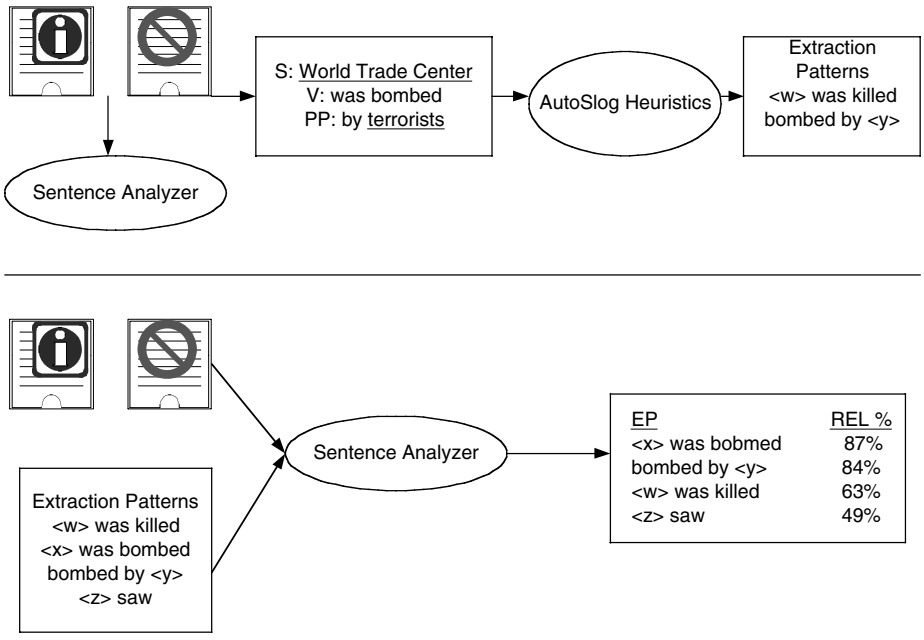

Figure VIII.9. Flow of the AutoSlog-TS system.

Each frame had to be fully tagged, and an ontology had to be provided along with the related lexicons.

AutoSlog-TS starts by using a parser that analyzes the sentences, determines clause boundaries, and marks subjects, verbs, direct objects, and prepositional phrases of each clause. It then uses a set of extraction pattern templates and generates an extraction pattern for each noun phrase in the corpus. The extraction patterns are graded by using the two bins of documents provided by the user. Extraction patterns that appear mostly in the bin of the important documents are ranked higher. An example of the flow of the AutoSlog-TS system is shown in Figure VIII.9.

The main steps within AutoSlog-TS can be broken down as follows:

1. The user provides two sets of documents, interesting (I) and noninteresting (N).
2. Shallow Parsing is performed for all the documents, and, on the basis of the predefined templates all patterns that match one of the templates are extracted (EP).
3. For each extraction pattern in EP, we compute the relevance of the pattern:

$$Rel(Pat) = \Pr(D \in I | Pat \in D) = \frac{\#(I, Pat)}{\#(I \bigcup N)},$$

   where #(I, Pat) is the number of documents in the document collection I that contain pattern P.
4. We compute the importance of each extraction pattern in EP according to the following formula and rank them in a decreased order:

$$Imp(Pat) = Rel(Pat) \log_2(\#(D, Pat)).$$

5. The system presents the top-ranked rules to the user for evaluation.

<subj> exploded	Murder of <np>	Assassination of <np>
<subj> was killed	<subj> was kidnapped	Attack on <np>
<subj> was injured	Exploded in <np>	Death of <np>
<subj> took place	Caused <dobj>	Claimed <dobj>
<subj> was wounded	<subj> occurred	<subj> was loctated
Took place on <np>	Responsibility for <np>	Occurred on <np>
Was wounded in <np>	Destroyed <dobj>	<subj> was murdered
One of <np>	<subj> kidnapped	Exploded on <np>

Figure VIII.10. Table of the top 24 extraction patterns in the AutoSlog-TS evaluation.

The system was evaluated on MUC-4 documents. A total of 1,500 MUC-4 documents were used, and 50 percent of them were relevant according to the user. The system generated 32,345 patterns and after patterns supported only by one document were discarded, 11,225 patterns were left. The top 24 extraction patterns are shown in Figure VIII.10.

The user reviewed the patterns and labeled the ones she wanted to use for actual extraction. So, for instance, "<subj> was killed" was selected for inclusion in the extraction process, and <subj> was replaced by <victim>. It took the user 85 minutes to review the top 1,970 patterns.

Certainly this approach shows much promise in building new extraction systems quickly because very little manual effort is needed in terms of rule writing and corpus annotation. The primary drawback is that a fairly strong parser needs to be used for analyzing the candidate sentences.

VIII.5.2 Mutual Bootstrapping

Riloff and Jones (Riloff and Jones 1999) took this idea of bootstrapping even further by suggesting *mutual bootstrapping*. Here the starting point is a small lexicon of entities (seed) that share the same semantic category.

In a way similar to AutoSlog-TS, the corpus is processed and all possible extraction patterns are generated along with the noun phrases that are extracted by them. The main purpose of this approach is to extend the initial lexicon and to learn accurate extraction patterns that can extract instances for the lexicon.

Initialization

- $N =$ total number of extraction patterns
- $\text{EP}_i =$ one extraction pattern $(i = 1..N)$
- EPData = a list of pairs (EP_i, Noun Phrases generated by the EP_i)
- SemLex = the list of seed words (the initial lexicon)
- EPlist = {}

Loop

1. Score all extraction patterns in EPData : Find for each EP_i how many items from SemLex it can generate.

www location	www company	terrorism location
offices in <x>	owned by <x>	living in <x>
facilities in <x>	<x> employed	traveled to <x>
operations in <x>	<x> is distributor	become in <x>
operates in <x>	<x> positioning	Sought in <x>
seminars in <x>	motivated <x>	presidents of <x>
activities in <x>	sold to <x>	parts of <x>
consulting in <x>	Devoted to <x>	To enter <x>
outlets in <x>	<x> thrive	ministers of <x>
customers in <x>	Message to <x>	part in <x>
distributors in <x>	<x> request information	taken in <x>
services in <x>	<x> has positions	returned to <x>
expanded into <x>	offices of <x>	process in <x>

Figure VIII.11. Table of extraction patterns from mutual bootstrapping.

2. Best_EP = highest scoring extraction pattern (extracted the highest number of items from SemLex)
3. Add Best_EP to EPList
4. Add Best_EP's extractions to SemLex
5. Goto 1

The top 12 extraction patterns in each of 3 problems (locations mentioned in company home pages, company names mentioned in company home pages, and locations mentioned in terrorist-related documents) are shown in Figure VIII.11.

VIII.5.3 Metabootstrapping

One of the main problems encountered with mutual bootstrapping is that once a word is added to the lexicon that does not belong to the semantic category, a domino effect can be created, allowing incorrect extraction patterns to receive high scores and thus adding many more incorrect entries to the lexicon. To prevent this problem, Riloff and Jones suggest using another method called *metabootstrapping*, which allows finer grain control over the instances that are added to the lexicon.

In metabootstrapping, only the top five instances that are extracted by using the best extraction pattern are retained and added to the permanent semantic lexicons. All other instances are discarded. The instances are scored by counting, for each instance, how many extraction patterns can extract it.

Formally, the score of instance I_j is computed as follows:

$$score(I_j) = \sum_{k=1}^{N_j} 1 + (.01 * \text{Imp}(\text{Pattern}_k)),$$

where N_j is the number of extraction patterns that generated I_j.

After the new instances are added to the permanent semantic lexicon, the mutual bootstrapping starts from scratch. A schematic view of the flow of the metabootstrapping process is presented in Figure VIII.12.

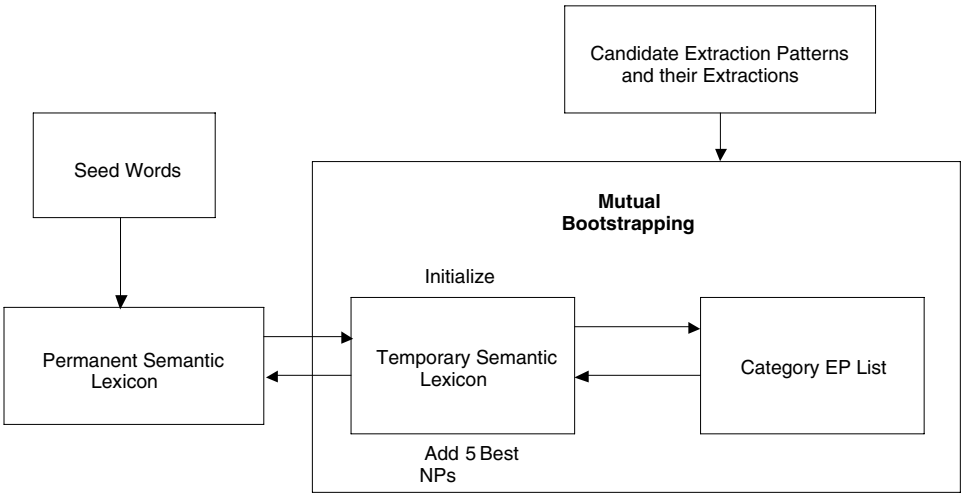

Figure VIII.12. Flow diagram of metabootstrapping.

Evaluation of the Metabootstrapping Algorithm

Two datasets were used: one of 4,160 company Web pages, and one of 1,500 articles taken from the MUC-4 corpus. Three semantic categories were extracted from the Web pages (locations, company names, and titles of people), and two semantic categories were extracted from the terror-related articles (locations and weapons). The metabootstrapping algorithm was run for 50 iterations. During each iteration, the mutual bootstrapping was run until it produced 10 patterns that extracted at least one new instance that could be added to the lexicon.

In Figure VIII.13, one can see how the accuracy of the semantic lexicon changes after each number of iterations. The easiest category is Web location, and the most difficult categories are weapon and Web title (titles of people mentioned on the Web page).

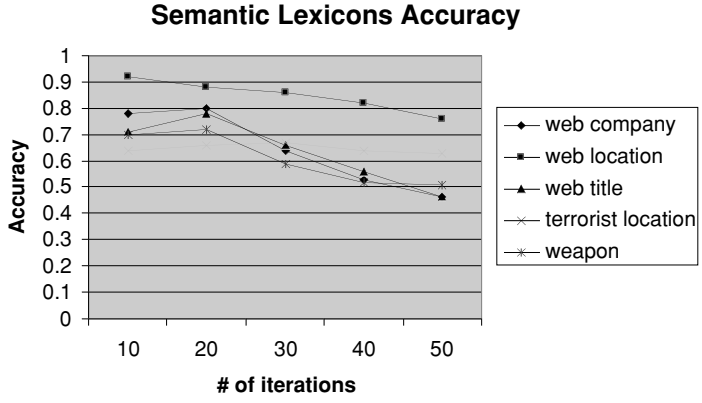

Figure VIII.13. Accuracy of the semantic lexicons as a function of the number of mutual bootstrapping iterations.

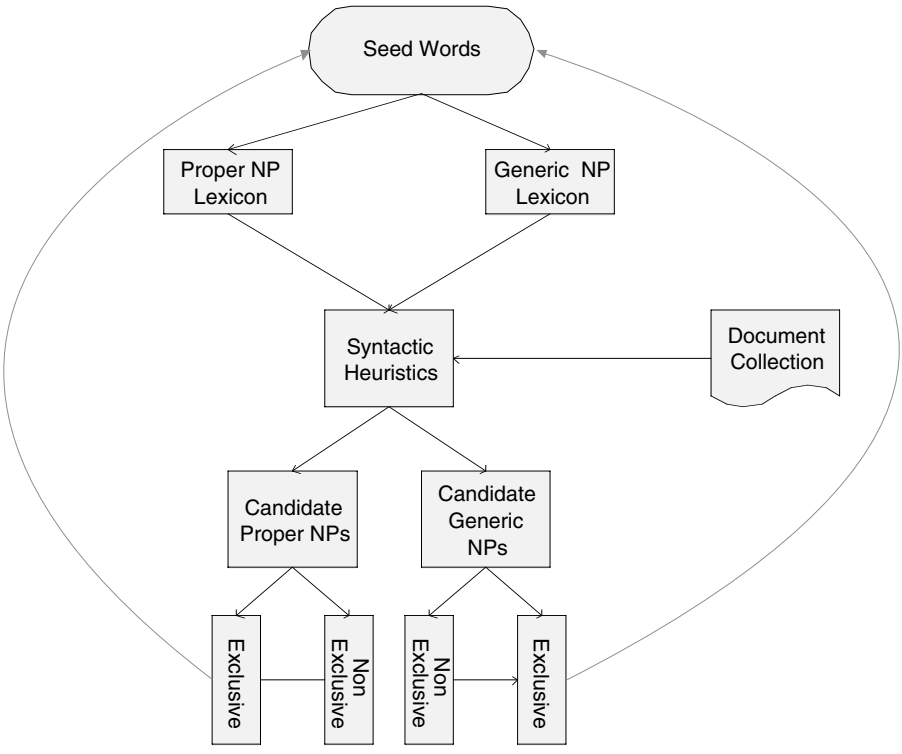

Figure VIII.14. Heuristic-based bootstrapping.

VIII.5.4 Using Strong Syntactic Heuristics

Phillips and Riloff (Phillips and Riloff 2002) took a different approach to building semantic lexicons. They learned two lexicons; one contained proper noun phrases (PNP) and the other generic noun phrases (GN). The lexicons were acquired by using a set of *syntactic heuristics*. In particular, they used three types of patterns. The architecture of the heuristic-based bootstrapping is shown in Figure VIII.14.

The first type included appositives such as "the president, George Bush," or "Mary Smith, the analyst." The second type consisted of IS-A clauses, which are NP followed by "to be" VP followed by NP. An example of an IS-A clause is "Bill Gates, the chairman of Microsoft." The last type comprised compound nouns that have the form GN + PNP. An example of such a construct is "the senator John Kerry."

A mutual property of all three types is that they establish a relationship between at least one GN and one PNP. The bootstrapping algorithm will infer relationships between an element that is already in one of the lexicons and an element that is not yet in any of the lexicons. These relations enable the algorithm each time to extend either the PNP lexicon or the GN lexicon. The algorithm alternates between learning a new GN based on the PNP lexicon and learning a new PNP based on the GN lexicon.

As an example, if one is trying to extend the people lexicons and we have in our PNP person lexicon the name "John Kerry" and the sentence "senator John Kerry" is encountered, one would learn that "senator" is a generic noun that stands for a

person; it will be added to the GN person lexicon. We can now learn new names of people that come after the GN "senator."

Normally, when a proper noun phrase is added to the PNP lexicon, the full phrase is used, whereas typically a generic noun phrase is added to the GN lexicon when just the head noun is used. This is done to increase the generality of the lexicon without sacrificing accuracy.

Take, for instance, the generic noun phrase "financial analyst." It is enough just to add analyst to the GN lexicon, and no harm will result. On the other hand, consider the proper noun phrase "Santa Barbara." Clearly, we can not add just Santa or just Barbara to the PNP lexicon of locations.

One of the main problems of bootstrapping approaches in general is that some generic phrases are ambiguous and can be used with a variety of semantic classes. An example is the generic noun "leader." This noun can designate either a company (which is a leader in its area) or a person (in the political domain or in the financial–corporate domain). If we add "leader" to the GN lexicon of people, in the next iteration it will add many corporations and contaminate our PNP people lexicon.

To alleviate this problem, the authors suggested using an exclusivity measure that is attached to each of the noun phrases. Only noun phrases that have an exclusivity measure exceeding some predefined threshold are added to the lexicon.

Given a phrase P and a semantic category C,

$$Exclusivity(P, C) = \frac{\#(P, C)}{\#(P, \neg C)},$$

where $\#(P, C)$ is the number of sentences in which P is collocated with at least one member of C, and $\#(P, C)$ is the number of sentences in which P is collocated with at least one member of all the semantic classes other than C. A typical exclusivity threshold is 5.

VIII.5.4.1 Evaluation of the Strong Syntactic Heuristics

This approach was tested on 2,500 *Wall Street Journal* articles (People and Organizations) and on a set of 1,350 press releases from the pharmacology domain (People, Organizations, and Products). The heuristics that added the highest number of entries to the PNP semantic lexicons were the compounds heuristics, whereas the appositives heuristics added the highest number of entries to the GN lexicons. The accuracy for the *Wall Street Journal* articles was between 80 percent and 99 percent for the PNP lexicons and between 30 and 95 percent for the GN lexicons. The accuracy results dropped when tested against the pharmaceutical press releases (77–95% for the PNP and 9–91% for the GN).

VIII.5.4.2 Using Cotraining

Blum and Mitchell (Blum and Mitchell 1998) introduced the notion of cotraining – a learning technique that tries to learn from a variety of views and sources simultaneously. Clearly, because there are three heuristics for learning the semantic lexicons, cotraining can be used after each boot-strapping cycle. All three lexicons will be joined after each step, and will a richer lexicon will result for each of them. A simple filtering mechanism can be used to eliminate entries with low support.

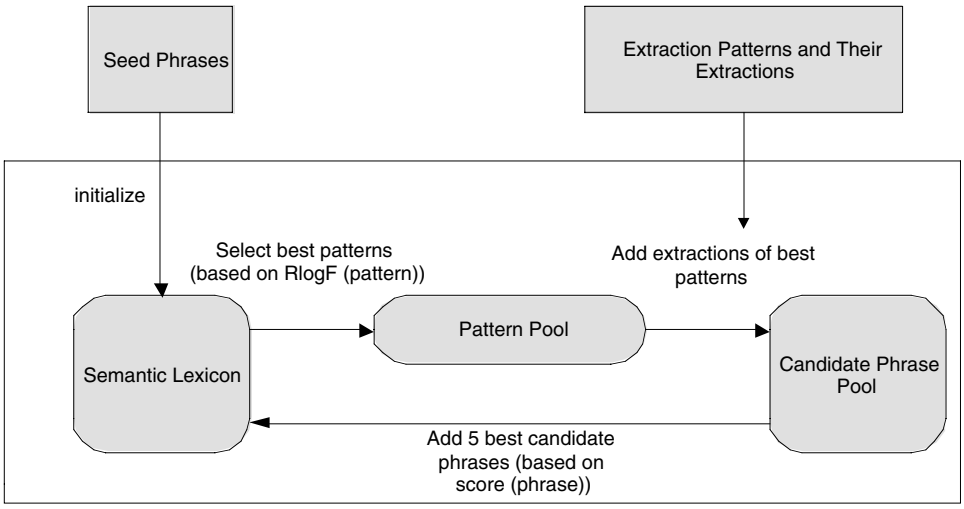

Figure VIII.15. The Basilisk algorithm.

It is common to add just entries supported by at least three sentences to the combined lexicon. Using the cotraining method results in a much more rapid learning of the lexicons (between 20 and 250% more entries were acquired) without much loss in accuracy.

VIII.5.5 The Basilisk Algorithm

Following in the footsteps of Riloff and Jones, Thelen and Riloff (Thelen and Riloff 2002) suggested a similar algorithm called Basilisk (Bootstrapping Approach to SemantIc Lexicon Induction using Semantic Knowledge). Differing from the metabootstrapping approach that uses a two-level loop (with mutual bootstrapping in the inner loop), Basilisk uses just a one-level loop and hence is more efficient. It solves the accuracy problem of the mutual bootstrapping by utilizing a weighted combination of extraction patterns. In particular, the approach utilizes $20 + i$ (where i is the index of the bootstrapping loop) extraction patterns as the pattern pool. The general architecture of Basilisk is shown in Figure VIII.15.

RlogF(pattern) was defined when we discussed the AutoSlog system. Score of phrase PH is defined as average log of the number of valid extraction patterns (for the given semantic category). The rationale is that a pattern is more trusted if it extracts a higher number of valid members of the semantic category. The log of the number of extractions is used so that a small number of extraction patterns having a particularly high number of valid extraction will not affect the average too drastically.

Formally,

- $\#(PH_i)$ is the number of extraction patterns that extract phrase PH_i.
- F_j = the number of valid extractions that were extracted by pattern P_j.

$$score(PH_i) = \frac{\sum_{j=1}^{\#(PH_i)} \log_2(F_j + 1)}{\#(PH_i)} \tag{1.1}$$

Note that here the assumption is that we have just one semantic category. If we are dealing with several semantic categories, then we will change score (PH_i) to be score(PH_i, C).

VIII.5.5.1 Evaluation of Basilisk on Single-Category Bootstrapping

Basilisk was compared against metabootstrapping on 1,700 MUC-4 documents. In the specific experiment performed by Thelen, just single nouns were extracted in both systems. Basilisk outperformed metabootstrapping in all six categories (building, event, human, location, time, weapon) by a considerable margin.

VIII.5.5.2 Using Multiclass Bootstrapping

Rather than learning one semantic category at a time, it seems that it will be beneficial to learn several semantic classes simultaneously. Clearly, the main hurdle would be those words that are polysemic and could belong to several semantic classes. To alleviate this problem we make the common assumption of "one sense per domain," and so our task is to find a conflict resolution strategy that can decide to which category each polysemic word should belong. The conflict resolution strategy used by Thelen preferred the semantic category assigned in a former iteration of the boot-strapping algorithm to any given phrase, and if two categories are suggested during the same iteration the category for which the phrase got the higher score is selected.

Another change that was able to boost the results and distinguish between the competing categories is to use mscore(PH_i, C_a), as defined below, rather than score(PH_i, C_a), as in equation (1.1).

$$mscore(PH_i, C_a) = score(PH_i, C_a) - \max_{b \neq a}(score(PH_i, C_b)) \tag{1.2}$$

This definition will prefer phrases or words that are highly associated with one category, whereas they are very loosely (if at all) associated with any of the other categories.

VIII.5.5.3 Evaluation of the Multiclass Bootstrapping

The performance of Basilisk improved when using the conflict resolution with the mscore function. The improvement was more notable on the categories BUILDING, WEAPON, and LOCATION. When the same strategy was applied to the metabootstrapping, the improvement was much more dramatic (up to 300% improvement in precision).

In Figure VIII.16 we can see the precision of the Basilisk system on the various semantic categories after 800 entries were added to each of the lexicons. The recall for these categories was between 40 and 60 percent.

VIII.5.6 Bootstrapping by Using Term Categorization

Another method for the semiautomatic generation of thematic lexicons by means of term categorization is presented in Lavelli, Magnini, and Sebastiani (2002). They view the generation of such lexicons as an iterative process of learning previously

Semantic Category	Number of Correct Entries	Precision
Building	109	13.6%
Event	266	26.6%
Human	681	85.1%
Location	509	63.6%
Time	43	5.4%
Weapon	88	11.0%

Figure VIII.16. Precision of the multicategory bootstrapping system Basilisk.

unknown associations between terms and themes. The process is iterative and generates for each theme a sequence of lexicons that are bootstrapped from an initial lexicon. The terms that appear in the documents are represented as vectors in a space of documents and then are labeled with themes by using classic categorization techniques. Specifically, the authors used the AdaBoost algorithm. The intermediate lexicons generated by the AdaBoost algorithm are cleaned, and the process restarts by using the cleaned lexicon as the new positive set of terms. The authors used subsets of the Reuters RCVI Collection as the document corpus and some of WordNetDomains's synsets as the semantic lexicons (split into training and test). The results for various sizes of corpora show that quite an impressive precision (around 75%) was obtained, and the recall was around 5–12 percent. Clearly, because there is no inherent connection between the corpus selected and the semantic lexicons, we can not expect a much higher recall.

VIII.5.7 Summary

The bootstrapping approach is very useful for building semantic lexicons for a variety of categories. The approach is suitable mostly for semiautomatic processes because the precision and recall we can obtain are far from perfect. Bootstrapping is beneficial as a tool to be used in tandem with other machine learning or rule-based approaches to information extraction.

VIII.6 FURTHER READING

Section VIII.1

More information on the use of HMM for text processing can be found in the following papers: Kupiec (1992); Leek (1997); Seymore, McCallum, and Rosenfeld (1999); McCallum, Freitag, and Pereira (2000); and Sigletos, Paliouras, and Karkaletsis (2002).

Section VIII.2

Applications of MEMM for information extraction are described in the following papers: Borthwick (1999), Charniak (2000), and McCallum et al. (2000).

Section VIII.3

Applications of CRF for text processing are described in Lafferty et al. (2001) and Sha and Pereira (2003).

Section VIII.4

TEG is described in Rosenfeld et al. (2004).

Section VIII.5

More details on bootstrapping for information extraction can be found in the following papers: Riloff (1993a), Riloff (1996a), Riloff and Jones (1999), Lavelli et al. (2002), Phillips and Riloff (2002), and Thelen and Riloff (2002).