Semantic Tagging using a Probabilistic Context Free Grammar *

Michael Collins
Dept. of Computer and Information Science
University of Pennsylvania
200 South 33rd Street, Philadelphia, PA 19104
mcollins@gradient.cis.upenn.edu

Scott Miller
BBN Technologies
70 Fawcett Street
Cambridge, MA 02138
szmiller@bbn.com

Abstract

This paper describes a statistical model for extraction of events at the sentence level, or "semantic tagging", typically the first level of processing in Information Extraction systems. We illustrate the approach using a management succession task, tagging sentences with three slots involved in each succession event: the post, person coming into the post, and person leaving the post. The approach requires very limited resources: a part-of-speech tagger; a morphological analyzer; and a set of training examples that have been labeled with the three slots and the indicator (verb or noun) used to express the event. Training on 560 sentences, and testing on 356 sentences, shows the accuracy of the approach is 77.5% (if partial slot matches are deemed incorrect) or 87.8% (if partial slot matches are deemed correct).

1 Introduction

Statistical models have been used quite successfully in natural language processing for recovery of hidden structure such as part-of-speech tags, or syntactic structure. This paper considers semantic tagging of text within the context of information extraction, as in the Sixth Message Understanding Conference (MUC-6). MUC-6 looked at extraction of events concerning management successions in newspaper texts: recovering the post, company, person entering and person leaving the post. We will concentrate on the initial stage of processing, extraction of events at the sentence level. For example, given the sentence

> Last week Hensley West , 59 years old , was named as president , a surprising development.

the desired output from the system would be
$\{$IN = Hensley West, POST = president, IND = named $\}$

POST is a slot designating the title of the position, IN is the person coming in to fill the post, and IND is an "indicator" – usually a verb or a noun – used to express the event. Table 1 gives some more examples.

The traditional approach to this problem, as exemplified in SRI's FASTUS system (Appelt et al. 93), has been to use hand-coded rules. These are typically encoded in a series of finite-state transducers that progressively build information in a bottom-up fashion. We are interested in developing a machine-learning approach to this problem for two reasons: First, developing hand-coded rules is a lengthy task which requires a fairly considerable amount of expertise – and a new set of rules must be developed for each new domain. Annotating training text examples such as those in table 1 can conceivably be done by a non-expert. Second, writing accurate rules is difficult, as there are many complex interactions between the rules, and there are many details to be covered. This task becomes even more complex when the interaction between the sentence-level rules and the later stages of processing (co-reference and merging, see section 1.1) is considered. Machine learning techniques have been shown to be highly effective at managing this kind of complexity in applications such as speech recognition, part-of-speech tagging and parsing.

Not surprisingly this problem can be approached using finite-state tagging methods, which have previously been applied to part of speech tagging (Church 88) and named-entity identification (Bikel et al. 97). We initially consider this approach, but argue that the Markov approximation gives an extremely bad parameterization of the problem. Instead, the method uses a Probabilistic Context Free Grammar (PCFG), which has the advantages of being flexible enough to allow a good parameterization of the problem, while having an efficient decoding algorithm, a variant of the CKY dynamic programming algorithm for parsing with context-free grammars. The PCFG does *not* encode linguistic phrase structure;

(IN Jack Bradley) was (IND named) (POST acting president and chief executive officer) of this computer-network manufacturer .

(IN He) (IND succeeds) as (POST president) (OUT Edward Marinaro) , 56 , who is retiring .

(IN Mr. Stanley) 's (IND appointment) comes as AK Steel 's Middletown Works is under investigation by OSHA because of its safety record , which includes one accident that killed four men in April 1994 .

The unexpected (IND departure) of (OUT Citicorp 's highly regarded head of retail banking) and the appointment of a tobacco executive to fill his shoes has caught most Citicorp employees off-guard and confounded many analysts .

On Friday , the bank said (OUT Pei-yuan Chia) , head of retail banking , will (IND retire) this year .

Table 1: Some example sentences.

* The work reported here was supported in part by the Defense Advanced Research Projects Agency. Technical agents for part of this work were Fort Huachucha under contract number DABT63-94-C-0062. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States Government.

rather, it is semantically motivated, modeling choices such as the choice of indicator, number of slots, fillers for the slots, and generation of other "noise" words in the sentence.

While this paper largely concentrates on the management succession domain, and motivates many of the choices regarding representation with examples from it, the principles should be general enough to also work for other domains — in fact, the method was originally developed for an IE task involving company acquisitions (identifying the buyer, seller and item being bought), and then moved to the management succession domain with no domain-specific tuning.

1.1 The Complete Information Extraction Task

While a semantic tagging model may be useful for many tasks, our primary motivation is to use it within an information extraction (IE) system. Information extraction tasks involve processing an input text to fill slots in an output template, the DARPA-sponsored Message Understanding Conferences (MUCs) have evaluated IE systems from several research sites. Some previous tasks attempted at MUC involved extraction of information about terrorist attacks (MUC-4); joint ventures (MUC-5); and most recently, at MUC-6, management successions. In this paper we concentrate on the management succession task, figure 1 gives an example input-output pair from the domain[1].

```
(a) Who's News:  Restor Industries Inc.

 RESTOR INDUSTRIES Inc. (Orlando, Fla.) --
Hensley E. West, 50 years old, was named
president of this telecommunications-product
concern. Mr. West, who most recently was a
group vice president for DSC Communications
Corp. in Dallas, fills a vacancy created by
the retirement last September of John
Bradley, 63.
```

(b)

Event Number	Slot	Filler
1	IN OUT POST COMPANY	Hensley E. West John Bradley president RESTOR INDUSTRIES Inc.
2	OUT POST COMPANY	Hensley E. West group vice president DSC Communications Corp.

Figure 1: (a) A sample text from WSJ, involving management successions. (b) The two succession events in the text.

Most systems described in the MUC-6 proceedings followed the following three stages of processing in mapping an input text to a set of output templates:

[1] We've just shown the most important, "core" slots for the task – the MUC-6 specification includes additional information such as the reason for the change, the title of the people involved etc.

1) Pattern matching at the sentence level. This is the task that is approached in this paper. In the text in figure 1, "Hensley E. West, 50 years old, was named president of this telecommunications-product concern" would be processed to give { IN = "Hensley E. West", POST = "president", COMPANY = "this telecommunications-product concern", VERB = "named" } and ".... the retirement last September of John Bradley ...." would give { OUT = "John Bradley", NOUN = "retirement" , IND = "resignation"}

2) Coreference. Pronouns and definite NPs are resolved to their antecedents. For example, "this telecommunications-product concern" would be resolved to "RESTOR INDUSTRIES Inc.". This stage is important because pronouns and definite noun-phrases like "this telecommunications-product concern" are not informative slot-fillers.

3) Merging. The information in a template may be spread across several sentences. In the merging stage the information from multiple mentions of the same event is merged into a single template. In the example, the information centered around "named" and "retirement" would be identified as referring to the same event, and would be combined to give { IN = "Hensley E. West", OUT = "John Bradley", POST = "president", COMPANY = "RESTOR INDUSTRIES Inc." }

This paper's work attacks problem (1) alone, and is restricted to recovery of the IN, OUT and POST slots.

1.2 Previous Work

The majority of systems at MUC-6, including SRI's system FASTUS (Appelt et al. 93), and the best-performing system, from NYU (Grishman 95), used cascaded finite-state transducers, which were built by hand. The domain *independent* transducers tokenize the text, recognise person and company names, "chunk" noun and verb groups, and finally build some higher level, complete clauses. The domain *specific* rules then extract the slots from the sentence, using patterns such as the example on page 244 of (Appelt et al. 93): "*Company* hires or recruits *person* from *company* as *position*".

There have been a number of machine learning approaches to the sentence-level stage of information extraction. The AutoSlog system (Riloff 93; Riloff 96) automatically learned "concept node" definitions for use on the MUC-4 terrorist events domain. A concept node specifies a trigger word, usually a verb, and maps syntactic roles with respect to this trigger to semantic slots – for example, a concept node might specify *if trigger = "destroyed" and syntax = direct-object then concept = Damaged-Object* (Damaged-object is the name of the slot in this case). A concept node may also specify hard or soft constraints on the slot-fillers. The system uses the CIRCUS parser (Lehnert et al. 93) to find the syntactic roles in relation to the trigger. AutoSlog learns concept nodes given input-output pairs like those in figure 1, so the indicator words do not need to be specified.

Experiments showed that running AutoSlog followed by 5 hours of filtering the rules by hand gave a system that performed as well as a hand-crafted system.

The CRYSTAL system (Soderland et al. 95) also learns rules that map syntactic frames to semantic roles. The triggers can be more complicated than those in AutoSlog, in that they can specify whole sequences of words, or restrict patterns by specifying words or classes in the surrounding context. CRYSTAL learns patterns by initially specifying a maximally detailed pattern for each training example, then progressively simplifying and merging patterns until some error bound is exceeded. CRYSTAL uses the BADGER sentence analyzer to give syntactic information.

(Califf and Mooney 97) describe a system for extraction of information about job postings from a newsgroup. Relational learning is used to learn rule-based patterns that specify: 1) a pre-filler pattern that matches the text before the slot; 2) a pattern that must match the actual slot filler; and 3) a post-filler pattern that matches the text after the slot. The patterns can involve parts of speech, semantic classes of words, or the words themselves. An example pattern from (Califf and Mooney 97) for identifying locations is *pre-filler = in, filler = 2 or fewer words all proper nouns, post-filler = word1 is ",", word2 is a state*. This matches phrases like "in Kansas City, Missouri" or "in Atlanta, Georgia". The learning algorithm starts with the most specific rule for each training example, then generalizes by merging similar rules.

A major difference between the approaches described in (Riloff 93; Riloff 96; Soderland et al. 95) and the approach in this paper is that (Riloff 93; Riloff 96; Soderland et al. 95) rely on a syntactic parser to produce at least a shallow syntactic analysis. The approach described in this paper builds a system from a set of training examples, with only a part-of-speech tagger and a morphological analyzer as additional resources. The system in (Califf and Mooney 97) does not require a parser, but the patterns it uses are quite local (the pre-filler and post-filler patterns are adjacent to the slot). It isn't clear this method would work well for the management successions domain where there are often many "noise" words between the slots and the indicator. Another major difference between the methods is that the PCFG based method is probabilistic. This may be an advantage when the sentence-level stage of processing is combined with the later merging and coreference stages, as it gives a principled way of combining evidence from the different stages of processing: an uncertainty at the sentence level may, for example, be resolved at the merging stage — in this case it is useful for the sentence level system to be capable of giving a list of candidate analyses with associated probabilities.

2 Background

2.1 The Problem

We assume the following definitions:

1. A sentence, W, consists of n words, $w_1, w_2, ...w_n$.
2. A template, T, is a non-empty set of *slots*, where each slot is a label together with a tuple giving the start and end point of the slot in the sentence. For example, T = $\{$IN = $\langle 3, 4\rangle$, OUT = $\langle 5, 6\rangle\}$ means there is an IN slot spanning words 3 to 4 inclusive, and an OUT slot spanning words 5 to 6 inclusive.

   In the management succession domain there are three possible slots, IN, OUT and POST (abbreviated to I, O and P respectively). IN is the string denoting the person who is filling the post, OUT is the person who is leaving the post, and POST is the name of the post.
3. In addition, we assume that each template contains an additional *indicator* slot, which is the verb or noun used to express the template.

For example, a $\langle W, T\rangle$ pair might be

W = Last week Hensley West , 59 years old , joined the company as president , a surprising development .

T = $\{$IN = $\langle 3, 4\rangle$, POST = $\langle 14, 14\rangle$, IND = $\langle 10, 10\rangle\}$

As alternative notation in this paper we either list the strings in the template, for example T = $\{$IN = "Hensley West", POST = "president", IND = "joined"$\}$, or we show the $\langle W, T\rangle$ pair as a bracketed sentence:

> Last week (IN Hensley West) , 59 years old , (IND joined) the company as (POST president) , a surprising development .

Table 1 shows more examples from the management succession domain. The machine learning task is to learn a function that maps an arbitrary sentence W to a template T, given a training set of N pairs $\langle W_i, T_i\rangle$ $1 \leq i \leq N$. A test set of $\langle W, T\rangle$ pairs is used to evaluate the model.

In addition to a training set, we assume the following resources:

1. A part of speech (POS) tagger. The POS tagger described in (Ratnaparkhi 96) was used to tag both training and test data.
2. A lexicon which maps each indicator word in training data to a class, for example the morphological variants "join", "joins", "joined" and "joining" could all be mapped to the JOIN class. This can be done automatically by a morphological analyzer as in (Karp et al. 94), or by hand. (This resource is not strictly necessary, but will help to reduce sparse data problems).

A probabilistic approach defines a conditional probability $P(T \mid W)$ or a joint probability $P(T, W)$ for every candidate template for a sentence. The most likely template for a sentence W is then

$$T_{best} = \arg\max_T P(T \mid W) = \arg\max_T P(T, W) \quad (1)$$

The major part of this paper will be concerned with formalizing a stochastic model that defines $P(T, W)$.

3 A Probabilistic Model

3.1 A Naive Approach — Finite State Tagging

It is useful to note that a $\langle W, T \rangle$ pair can be represented as a tagged sentence $w_1/t_1, w_2/t_2, ...w_n/t_n$ where $T = t_1, t_2...t_n$ is the sequence of tags denoting the semantic type for each word in the sentence. For example, the tags could be I, O, P, IND, for the 3 slots and the indicator, and N for other (noise) words, as in

> Last/N week/N Hensley/I West/I ,/N 59/N years/N old/N ,/N joined/IND the/N company/N as/N president/P ,/N a/N surprising/N development/N ./N

As a straw man we consider using a standard bigram tagging model to tag test-set sentences. (Church 88) used this to recover part-of-speech tags, a related approach described in (Bikel et al. 97) gives a useful decomposition of $P(T, W)$ into two terms:

$$P(T, W) = P(L_1, L_2, ...L_m) \times \prod_{i=1...m} P(W_i|L_i) \quad (2)$$

$\{L_1, L_2, ...L_m\}$ is the underlying sequence of tags, in the above example $m = 7$ and the sequence is {N, I, N, IND, N , P, N}. W_i is the string of words under label L_i, for example $W_1 = \{\text{Last, week}\}$, $W_2 = \{\text{Hensley, West}\}$. The two terms are then simplified, using bigram Markov independence assumptions, to be

$$\begin{aligned} P(L_1, L_2, ...L_m) &= P(L_1|Start)P(End \mid L_m) \\ &\quad \times \prod_{i=2...m} P(L_i \mid L_{i-1}) \quad (3) \end{aligned}$$

and (if label L_i covers words $w_{si}...w_{ei}$)

$$\begin{aligned} &P(W_i|L_i) = P(w_{si}, w_{si+1}, ...w_{ei}|L_i) = \\ &P(w_{si} \mid Start, L_i)P(End \mid w_{ei}, L_i) \\ &\times \prod_{j=si+1...ei} P(w_j \mid w_{j-1}, L_i) \quad (4) \end{aligned}$$

This finite state approach has been highly effective for part of speech tagging (Church 88) and name finding (Bikel et al. 97). However, the next section considers the characteristics of the task in more detail, and argues that a finite-state tagger is a poor model for the task.

3.2 More about the task

In developing an intuition for the task, and motivating the choices made in modeling it, it is useful to consider the types of information that may be useful to a system. Consider the following 5 points:

1. There are 7 possible templates corresponding to the 7 non-empty subsets of {I,O,P}. The distribution over these alternatives is by no means uniform – see table 2 for the distribution.
2. The different slots tend to contain quite different lexical items or strings – for example, the IN and OUT slots are most likely to contain a proper name or a personal pronoun, whereas the POST slot contains strings such as "president", "chairman" etc.
3. The choice of indicator word depends greatly on the choice of template. For example "name" is very likely to be used to express an event involving a {I, P} template; "succeed" is very likely to express an {I, O, P} or {I, O} template. See the final column of table 2 for more examples.
4. The relative order of the slots and indicator in the text varies considerably depending on the choice of indicator. For example, given the template {I} and the verb "join" the order is most likely to be { I Indicator } (e.g. *IN joined the company*); whereas given the verb "hire" the order is usually {Indicator I} (e.g. *the company hired IN*).
5. In addition to the central indicator, there are often secondary indicators – mainly prepositions – which are strong signals of particular slots. For example, given the verb is "named" or "succeeded", the post is very likely to be preceded by the preposition "as" (e.g., *the company named her as president, he succeeds Jim Smith as president*).

By considering points 1-5 we can see that the finite-state tagging approach is deficient for the semantic tagging task. The lexical probabilities in equation (4) are probably sufficient to capture the lexical differences between different states (the preference of the IN slot to generate proper names, of the POST slot to generate words like "president" and so on). But the Markov approximation in equation (3) is deficient in many ways: it fails to capture the non-uniform distribution over the 7 possible templates, worse still it is deficient in that it can label more than one substring with the same slot label; it fails to capture the dependence of the slot order on the indicator word, or the dependence between the template and indicator.

3.3 A Probabilistic Context-Free Grammar

Our proposal is to replace the Markov assumption in (3) with a probabilistic context-free grammar, that is we assume that the label sequence has been generated by the application of r context-free rules $LHS_j \Rightarrow RHS_j$ $1 \leq j \leq r$ (LHS stands for left hand side, RHS stands for right hand side), and that

$$P(L_1 L_2....L_m) = \prod_{i=1..r} P(RHS_i \mid LHS_i). \quad (5)$$

Each LHS is a single non-terminal, and RHS is a string of one or more non-terminals. So for each non-terminal LHS in the grammar there is a distribution over possible RHSs which sums to one. Counts of context-free rules can be extracted from a training set of context free trees,

Template	%age	Typical example	Most frequent indicator/percentage
{I,P}	45.6	IN was named as POST	name/67%
{O}	18.6	OUT retired	retire/26%
{I,O}	16.6	IN succeeds OUT	succeed/98%
{I}	7.3	IN joined the company	join/31%
{O,P}	7.1	OUT resigned as POST	resign/46%
{I,O,P}	3.5	IN succeeded OUT as POST	succeed/100%
{P}	1.3	the company hired POST	–

Table 2: The distribution over the possible templates (the 7 non-empty subsets of {I,O,P}), and the most common indicator for each template. For example, 45.6% of the templates are {I,P}, and in these 45.6% of the cases "name" is chosen 67% of the time.

and used to estimate the parameters $P(RHS_i \mid LHS_i)$. Given a test data sentence, the most likely tree (and hence the most likely template) can be recovered efficiently using a variant of the CKY algorithm.

4 The Grammar

This section describes the underlying context-free structure[2] that we assume has generated the labels, and motivates it in terms of the observations in section 3.2. The context-free structure (the tree topology, and the choice of non-terminal labels within the tree), is deterministically derived from the initial labeling of the sentences — so given a set of labeled sentences, the context-free structures can be recovered and the parameters can be estimated.

4.1 The Leaf Categories

The tagging model as applied in the above example assumed five tags – for the IN, OUT, and POST slots, the indicator, and for noise (other words). In fact, we used rather more categories, which are listed in table 3. These labels can still be deterministically recovered from the labeled sentence though, given the additional information of a mapping from indicator words to their morphological stem (for example, the mapping "joined" ⇒ JOIN). The example sentence would have the following underlying leaf labels:

> [PREN Last week] [I Hensley West] [NOISE+ , 59 years old ,] [IND(JOIN) joined] [NOISE- the company] [P.Prep-(JOIN) as] [P president] [POSTN , a surprising development .]

4.2 The Context-Free Component – a Brief Sketch

The PCFG model assumes the pre-terminal[3] label sequence $\{L_1, L_2 ... L_m\}$ has been generated by a stochastic process with the following steps:

1. Decide whether to have noise words (PREN) before the template TEMP.
2. Decide whether to have noise words (POSTN) after the template TEMP.
3. Decide which slots to have (one of the 7 subsets of {I, O, P}).
4. Decide the class of indicator words.
5. Decide the order of the slots and indicator word.
6. For each slot, choose whether to have noise between it and the indicator (NOISE+ or NOISE-).
7. For each slot, choose whether to have a preposition directly preceding or following it.

Figure 2 gives an example tree and describes the context-free rules within it. The next section describes the grammar in more detail, showing how these 7 types of decision can be encoded as context-free rules.

4.3 The Context-Free Component in Detail

This section describes the top-down derivation of a sequence of leaves within a PCFG framework.

Choosing noise at the start/end of the sentence
This level of the model chooses whether to have noise preceding or following the text which expresses the succession information.

```
TOP -> PREN TEMP1 #there is noise at start
    -> TEMP1      #or there isn't

TEMP1 -> TEMP POSTN #there is noise following
      -> TEMP       #or there isn't
```

The TEMP non-terminal covers the span of the succession information, in the above example "Hensley ... president". P(PREN TEMP1 | TOP) can be interpreted as the probability of having noise at the beginning of the sentence, P(TEMP POSTN | TEMP1) is the probability of having post-noise.

$P(Slots)$

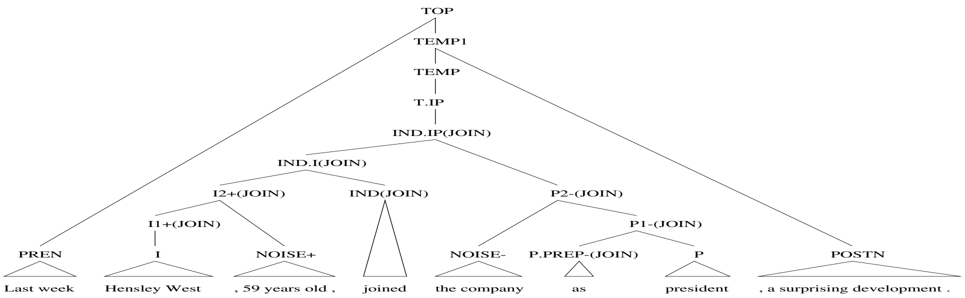
TEMP first re-writes in one of seven ways, corresponding to the 7 possible templates. The `T.` non-terminal encodes the slots that will be generated below it, for example `T.IO` would generate an IN and an OUT slot below it. So P(RHS | TEMP) will mirror the distribution in column 2 of table 2.

[2]The structures in this paper are non-recursive, and could, therefore, be equivalently handled by a hierarchy of finite-state transducers, or even a single equivalent non-deterministic finite-state automaton. However, it is quite possible that extensions to the models could require recursive structures.

[3]By pre-terminal, we mean a non-terminal that dominates words rather than other non-terminals.

Leaf label	Description
I, O, P	The IN, OUT and POST non-terminals
PREN	"Noise" words before the template
POSTN	"Noise" words after the template
NOISE+	"Noise" between slots and the indicator, which comes *before* the indicator
NOISE-	"Noise" between slots and the indicator, which comes *after* the indicator
IND(class)	Leaf dominating the indicator. $class$ can be any one of the morphological stems seen in training data. For example, IND(join) could dominate "join", "joins", "joined" or "joining".
I.Prep-(class) O.Prep-(class) P.Prep-(class)	Prepositions for the I, O and P slots, for a particular class, and which follow the indicator. For example, P.Prep-(join) would be a preposition for the Post slot, with an indicator in the $join$ class, and would most likely be "as"
I.Prep+(class) O.Prep+(class) P.Prep+(class)	Prepositions for the I, O and P slots, for a particular class, and which *precede* the indicator.

Table 3: The pre-terminal labels that are used in the system.



Rule	Interpretation
TOP $\Rightarrow$ PREN TEMP1	Choose to have pre-noise (PREN)
TEMP1 $\Rightarrow$ TEMP POSTN	Choose to have post-noise (POSTN)
TEMP $\Rightarrow$ T.IP	Choose to have IN and POST slots
T.IP $\Rightarrow$ IND.IP(JOIN)	Choose to use a member of the JOIN class of indicators
IND.IP(JOIN) $\Rightarrow$ IND.I(JOIN) P2-(JOIN)	Generate the POST slot to the right of the indicator
IND.I(JOIN) $\Rightarrow$ I2+(JOIN) IND(JOIN)	Generate the IN slot to the left of the indicator
I2+(JOIN) $\Rightarrow$ I1+(JOIN) NOISE+	Have noise between the IN slot and the indicator
I1+(JOIN) $\Rightarrow$ I	Choose not to have a preposition for the IN slot
P2-(JOIN) $\Rightarrow$ NOISE- P1-(JOIN)	Have noise between the POST slot and the indicator
P1-(JOIN) $\Rightarrow$ P.Prep-(JOIN) P	Choose to have a preposition attached to the POST slot

Figure 2: An example context-free tree. The table shows the interpretation of each of the rules in the tree.

```
TEMP -> T.IOP       TEMP -> T.IO      TEMP -> T.I
                    TEMP -> T.IP      TEMP -> T.O
                    TEMP -> T.OP      TEMP -> T.P
```

$P(Class|Slots)$

The next step is to choose the $Class$ of indicator that is used to express the transaction. Each $Class$ is a set of words with the same morphological stem, for example the JOIN class would include join, joins, joined and joining. $P(Class|Slots)$ is implemented in the CFG fragment shown below. The IND non-terminal encodes which slots need to be generated, and the $Class$ used to express the transaction. Each `T.` rule can re-write in N ways, where N is the number of classes.

```
T.IOP -> IND.IOP[Class]  T.I -> IND.I[Class]
T.IO  -> IND.IO[Class]   T.O -> IND.O[Class]
T.IP  -> IND.IP[Class]   T.P -> IND.P[Class]
T.OP  -> IND.OP[Class]
```

$P(Order|Class, Slots)$

Having chosen the $Slots$ to be generated, and the $Class$ used to express the event, there are many possible orders in which the slots and class can appear. In the above example (Slots = {I, P}, Class = JOIN) there are 6 permutations ({JOIN I P}, {I JOIN P}, {I P JOIN } and so on). It is necessary to estimate a distribution over these alternatives. The order is parameterized using a binary branching, context-free fragment: part of this (all rules with `LHS = IND.IP[Class]`) is shown below. The full grammar specifies similar rules for all `IND.X[Class]` where X is any one of the non-empty subsets of {I, O, P}.

```
IND.IP[Class]  -> IND.I[Class]   P2-[Class]
IND.IP[Class]  -> P2+[Class]     IND.I[Class]
IND.IP[Class]  -> IND.P[Class]   I2-[Class]
IND.IP[Class]  -> I2+[Class]     IND.P[Class]
```

The notation is:

- IND keeps tracks of which slots still need to be generated. For example $IND.IP[Class]$ means that the IN and POST slots need to be generated.
- The $I2$, $O2$, and $P2$ non-terminals will eventually generate the IN, OUT and POST leaves. The "2" stands for level 2 – more in the next section on why this is necessary. "+" means the slot appears before the head-word, "-" means it appears after. The $Class$ is propagated to the $I2$, $O2$ and $P2$ non-terminals. Propagation of the $Class$ and direction (+ or -) is important because the identity of any prepositions is conditioned on this information.

Each binary rule expresses a choice of which of the remaining slots to generate next, and which direction to generate it in. So IND.IP[Class] can re-write in 4 ways: either the IN or POST slot can be generated either to the left or right of the head-word itself.

Choosing to generate noise between the slots

Noise can appear after any slot preceding the indicator, or before any slot following the indicator. The CFG rules below encode the decision to have noise in a gap or not, for an IN slot generated before or after the indicator. The rules for OUT and POST are similar.

```
I2+[Class]      -> I1+[Class]  NOISE+
I2+[Class]      -> I1+[Class]
I2-[Class]      -> NOISE-  I1-[Class]
I2-[Class]      -> I1-[Class]
```

Choosing to generate a preposition (or other indicator) linked to a slot

Any of the slots can have an adjacent "indicator", usually a preposition. The rules below encode the binary decision of whether to include an indicator for an IN slot – the OUT and POST cases are similar.

```
I1+[Class]      -> I     I.Prep+[Class]
I1+[Class]      -> I
I1-[Class]      -> I.Prep-[Class]     I
I1-[Class]      -> I
```

Again, for each $I1$, $O1$ or $P1$ non-terminal there are two possible re-writes, one binary, one unary, encoding whether or not to generate a preposition. The $I.Prep$, $O.Prep$ and $P.Prep$ non-terminals then generate the indicator with a bigram model. The non-terminal encodes whether the slot appears before or after the head-word ("+" or "-"), and the $Class$ of the head-word.

5 Training the Model

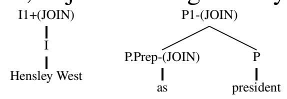
There are two steps to training the model: first, recovering the underlying tree structure from the training data labels; second, deriving counts of the CF rule applications and bigram sequences and using these to estimate the parameters of the model.

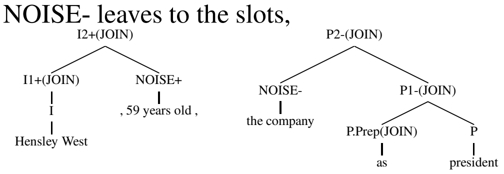
5.1 Deriving the Tree Structures in Training Data

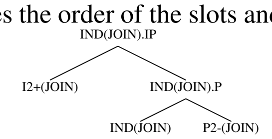
While the tree structure described in section 4.3 may seem complex, it is important to realise that it can be deterministically derived from an annotator's labeling of the Slots and Indicator. This section describes how the structure is derived in a bottom up fashion using the following annotated sentence as example input to the process:

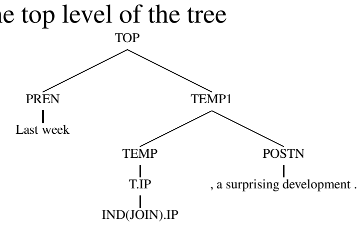
Last week [I Hensley West] , 59 years old , [IND joined] the company as [P president] , a surprising development .

The 6 stages are as follows:

1. Identify the class of the indicator, and add this information to the IND label. Mark any prepositions adjacent to the slots. Label "noise" words with either PREN, POSTN, NOISE+ or NOISE-. The output from this stage would be:
   [PREN Last week] [I Hensley West] [NOISE+ , 59 years old ,] [IND(JOIN) joined] [NOISE- the company] [P.Prep-(JOIN) as] [P president] [POSTN , a surprising development .]
2. Build level 1 of the slots, by including attached prepositions, or just building a unary rule
   

3. Build level 2 of the slots, by attaching NOISE+ or NOISE- leaves to the slots,
   

4. Build the binary-branching context-free structure that defines the order of the slots and indicator.
   

5. Add the top level of the tree
   

5.2 Context Free Rule Probabilities

Once the training data is processed to have full context-free trees, the grammar can be automatically read from these trees, and event counts can be extracted and used to estimate the parameters of the model. The maximum likelihood estimate for a CF rule `LHS -> RHS` is

$$P(RHS|LHS) = \frac{C(RHS, LHS)}{C(LHS)}$$

where $C(x)$ is the number of times event x has been seen in training data. This estimate can be unreliable, particularly for low values of $C(LHS)$. So we smooth this estimate with a "backed off" estimate P_b

$$P(RHS|LHS) = \lambda \frac{C(RHS, LHS)}{C(LHS)} + (1-\lambda)P_b$$

where $0 \leq \lambda \leq 1$. The backed off estimate $P_b = \frac{C(RHS, LHS_b)}{C(LHS_b)}$ is based on a subset of the context and the estimate is more robust but is less detailed. For example, P_b for $P(\text{T.IOP} \rightarrow \text{IND.IOP[Class]}|T.IOP)$ might be $P(\text{T} \rightarrow \text{IND [Class]}|T)$, i.e. an estimate that ignores the slots when choosing the class of indicators. This method borrows heavily from smoothing techniques in language modeling for speech recognition — (Jelinek 90) describes methods for estimating λ.

5.3 Bigram Probabilities

The bigram model is used at the leaves of the tree to generate the words themselves, for example to estimate `P(the president | P)`. The most obvious way to estimate this is as $P(the|START, P)$ $*P(president|the, P) * P(END|president, P)$ with smoothing being implemented by interpolation between $P(w|w_{-1}, State) \rightarrow P(w|State) \rightarrow \frac{1}{V}$ where V is the vocabulary size. Unfortunately we do not have space to go into the full details of the smoothing here (in the final implementation part-of-speech information was also used to smooth the estimates).

6 Experiments

This section describes experiments on the management successions domain. Before giving the results, we discuss how to deal with sentences that have more than one indicator.

6.1 Dealing with Sentences that have more than one Indicator

Thus far the model has assumed that there is only one indicator per sentence. However, training data frequently has more than one indicator, as in

> Mr. Smith was named president of the company, succeeding Fred Jones.

There are two events in this sentence, one centered around *named*, the other centered around *succeeding*. The solution is to transform sentences in both training and test data to give one sentence per indicator, in this case the sentence would be expanded to give two sentences:

> Mr. Smith was *named* president of the company, succeeding Fred Jones.

> Mr. Smith was named president of the company, *succeeding* Fred Jones.

The first sentence is for the *named* event, the second is for *succeeding*. The *indicator* is replaced with **indicator** to show that it is under interest — when decoding test data the model either recognises **named** as a potential indicator, but ignores *succeeding*, or ignores *named* and recognises **succeeding**. If the sentence appeared in training data it would be transformed to give two training data trees. We should stress that this process is *completely automatic* once the indicators have been identified in the text.

6.2 Results

The model was trained on 563 sentences, and tested on another 356 sentences. (That is, 563/356 sentences *after* producing one sentence per indicator as described in section 6.1). The sentences were taken from the "Who's news" section of Wall Street Journal, which is almost exclusively about management successions. The training sentences were taken from 219 Who's News articles in the 1996 section, the test sentences were taken from 131 articles in the 1995 section. The sentence level annotation was part of an annotation effort for the full extraction task, which therefore also marked the relevant coreference relationships and the complete output template as in figure 1.

The test data sentences always contain an event, and have all indicators marked as **indicator** — only those indicators that have 1 or more slots attached to them are marked. This is an idealization, in that we avoid problems of false positives, cases where a potential indicator is not used to express an event. See section 6.4 for suggestions about how to extend the model to deal with false positives.

The results are shown in table 4. We define precision and recall when comparing to the annotated test set answers (gold standard) as

$$\text{Precision} = \frac{\text{Number of correct slots}}{\text{Number of slots proposed}}$$

$$\text{Recall} = \frac{\text{Number of correct slots}}{\text{Number of slots in the gold standard}}$$

In addition we report the standard "F-Measure", which is a combination of precision and recall

$$\text{F-Measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The results are quoted for the IN, OUT and POST slots (the IND slot is not scored, as it is marked in test data and would score 100% recall/precision, inflating the scores). The number of "correct" slots varies depending on how partial matches are scored – a partial match is where an output slot does not match a gold standard slot exactly, but does partially overlap. For example, in

> Bill Smith was elected vice president, human resources.

Score for partial	Precision	Recall	F-Measure
0	80.6%	74.6%	77.5%
0.5	85.9%	79.6%	82.6%
1.0	91.3%	84.5%	87.8%

Table 4: Results on 356 test data sentences, training on 563 sentences

the gold standard might designate the slot as "vice president, human resources", whereas the program output might just mark "vice president". We present three precision/recall scores — where a partial match scores 0, 0.5 or 1.0, and

$$\text{Number of correct slots} = \text{Number of exact matches}$$
$$+\text{Score for a partial match} \times \text{Number of partial matches}$$

6.3 Analysis of the results

In this section we look at the errors the system makes in more detail. There are two categories of error: precision errors (incorrect slots); and recall errors (slots the system failed to propose). For these tests we ran experiments on the training data, jack-knifing (i.e. using cross-validation) it into 4 sections, in each case training on three-quarters of the training set and testing on the other quarter. Tables 5 and 6 show the results on this data set.

Gold	Proposed	Correct	Partial	Correct +Partial
986	944	769	71	840

Table 5: Results on the jack-knifed training set (Counts)

Score for partial	Precision	Recall	F-Measure
0	81.5%	78.0%	79.7%
0.5	85.2%	81.6%	83.4%
1.0	89.0%	85.2%	87.0%

Table 6: Results on the jack-knifed training set (Percentages)

6.3.1 Precision errors

Table 7 shows the 104 precision errors categorized by hand into four categories. These four categories were:

1) Semantically Plausible. Here the model has selected a slot-filler that looks good semantically, but is ruled out for other reasons (usually syntactic). For example,

> The appointment puts (IN Mr. Zwirn) , 41 years old , in line to succeed the unit 's president , Frank R. Bakos , 58 , who is (IND retiring) at year end .

Here "Mr. Zwirn" is semantically a good filler for "retiring", but syntactically this is almost impossible.

Error type	%age	Error sub-type	%age
Semantically Plausible	37.1%	Relative clauses	18.1%
		Subject	10.5%
		Other	8.6%
"Correct"	25.7%	Good alternative	17.1%
		> 1 reference	8.6%
Bad lexical information	8.6%		
Others	28.6%		

Table 7: The percentage of errors in each error category

We sub-divided this class into 3 sub-categories: problems with relative clauses, as in the example above; problems with non-relativized subjects, for example "Brandon Sweitzer , 53 , succeeds (IN Mr. Wakefield) as president of Guy Carpenter and also (IND becomes) (POST the unit 's CEO) , succeeding Richard Blum , 56 ."; and problems that fell outside these categories.

2) "Correct". These slots were not seen in the gold-standard, but were deemed pretty much correct, in that they would not hurt (and might even help) the score of a full system. They fall into two sub-categories – "good alternative", where the model's output is different from the gold standard but still looks reasonable, either because the sentence has more than one reasonable answer, or the gold standard is simply wrong; "> 1 reference", where there is more than one reference to the slot filler in the sentence, and the model has chosen a different one from the gold standard. For example,

> (OUT Mr. Johnson) , 52 , said he resigned (POST his positions as chief executive officer)

Here the model marked "Mr. Johnson" as OUT, the annotator marked "he", and both are in some sense correct.

3) Bad Lexical Information. In these cases the model selected a slot filler that is clearly bad for lexical reasons, for example

> Mr. Broeksmit is the (OUT latest) in a string of employees to (IND leave) the firm ...

4) Other. Miscellaneous errors which do not fall into the above three categories.

6.3.2 Recall Problems

Of the 356 test-set sentences, 330 (92.7%) were processed by the system to give some output — no output was produced for 26 cases. This accounts for the recall figures in table 4 being lower than the precision figures (for example, with a score of 0 for partials, precision = 80.6%, recall = 74.6%, and 92.7%*80.6% = 74.7%). Of these 26 cases, 24 involved an indicator word that had never been seen in training data. The other 2 cases involved an unusual usage of "succeed", which had never been seen in training data and was peculiar enough for the system to fail to get an analysis (we set a probability threshold such that the machine gives up if it fails to find an analysis above this probability).

6.4 Dealing with False Positives

This work has made a simplifying assumption, that test sentences were marked with indicators that had one or more slots. This section considers how this process could be automated.

A first step would be to identify in test data morphological variants of words that had been seen as indicators in training data. However this would inevitably lead to false positives — that is, potential indicators appearing in cases where they don't indicate an event. We could see two potential approaches for filtering out these spurious cases: first, word-sense disambiguation methods similar to those in (Yarowsky 95); second, we could extend the model to have an eighth, empty, template as a possibility — the model should then learn how often null templates occur, and what kind of lexical items tend to produce them.

We leave this to future work. At least in this dataset (Who's News articles) we believe that the false positive problem will not be severe, as the articles contain information almost exclusively on management successions, and most of the indicators are unambiguous within this sub-domain.

The models have also made the assumption that an indicator is used to express each event. This may not be the case in all information extraction tasks, in some there may not be clear indicator words; again, we leave dealing with this limitation to future work.

7 Future Work

We anticipate two directions for future work: first, refining the current model to improve its performance, and second, extending the current model to encompass the complete information extraction task.

7.1 Refining the Model

When deciding on the direction of future work, it is useful to consider the error analysis in table 7. The majority of errors (the "semantically plausible" class) were cases where the model picked a slot that was semantically plausible, but syntactically impossible. It is unlikely that this problem can be solved with the approach described here, even with vastly increased amounts of training data. Our feeling is that a full syntactic parser as a first stage could radically improve performance. An improved approach might be to fully integrate the recovery of syntactic structure and semantic labelings, in a similar way to the approach used in BBN's SIFT system (Miller et al. 98).

7.2 Extending the Model

As discussed in section 1.1, the standard approach to information extraction involves three stages of processing: sentence level pattern matching, coreference, and template merging. Of these stages, our current work addresses only sentence level pattern matching. However, we believe that the generative statistical framework described in this paper could be extended advantageously to the complete information extraction problem. In extending the framework, we envision that the information extraction task would be performed using an inverted "information production" model.

We can think of this model as approximating, to some degree, the process by which text is produced by an author. Specifically, we assume that each message is produced according to a four stage process:

1) First, the author decides what facts to express. For example, the text in figure 1 can be thought of as expressing two succession events: IN = "Hensley E. West", OUT = "John Bradley", POST = "president", COMPANY = "RESTOR INDUSTRIES Inc.", and OUT = "Hensley E. West", POST = "group vice president", COMPANY = "DSC Communications Corp.". This process can be modeled as a prior probability distribution over sets of templates. In this example, the model would give the prior probability of a message containing exactly two succession templates: one containing slots IN, OUT, POST, COMPANY and the other containing slots OUT, POST, COMPANY.

2) After deciding what facts to express, the author must decompose them into one or more component events. For example, the succession event IN = "Hensley E. West", OUT = "John Bradley", POST = "president", COMPANY = "RESTOR INDUSTRIES Inc." is decomposed into two smaller events: IN = "Hensley E. West", POST = "president", COMPANY = "RESTOR INDUSTRIES Inc." and IN = "Hensley E. West", OUT = "John Bradley". This process can be modeled as a probability distribution over "template splitting operations", conditioned on the full template being expressed. Template splitting operations are thus the generative analogue of the merging operations used in most information extraction systems.

3) Next, each component event must be expressed as a linguistic pattern. For example, the event IN = "Hensley E. West", POST = "president", COMPANY = "RESTOR INDUSTRIES Inc." is expressed as the linguistic pattern "IN ... was named POST of COMPANY", and the event IN = "Hensley E. West", OUT = "John Bradley" is expressed as the linguistic pattern "IN ... fills a vacancy created by the retirement ... of OUT". This process can be modeled as a probability distribution over linguistic patterns, conditioned on the partial template being expressed. Modeling this distribution is the subject of the main body of this paper.

4) Finally, the entities involved in events must be realized as word strings within patterns. For example, "RESTOR INDUSTRIES Inc." is realized as "this telecommunications-product concern", and "Hensley E. West" is realized as "Mr. West". This process can be modeled as a probability distribution over "descriptor generating operations", conditioned on the entity being expressed and other features of the text. For example, given that the author intends to express "Hensley E. West", and given that the full name appears earlier in the text, the model would assign a certain probability to generating the word string "Mr. West". In this case, the descriptor generating operation would be [title + last name].

Clearly, there are many details that would need to be resolved before a complete generative model of information extraction could be implemented. In this paper, we have described a model containing two of the necessary components: a prior model over templates, and a model of linguistic patterns conditioned on those templates. A complete generative model for IE would offer two potentially powerful advantages. First, the model would provide principled probability estimates for selecting the most likely set of templates given an input message:

T = set of templates (the final output)
M= the message, C = components
P = linguistic patterns, S = slot fillers in T
D = descriptions used to express the slots

$$P(\mathrm{T}|\mathrm{M}) = \sum_{\mathrm{C,\,P}} P(\mathrm{T,\,C,\,P}|\mathrm{M})$$

where

$$P(\mathrm{T,\,C,\,P}|\mathrm{M}) = \frac{P(\mathrm{T}) \times P(\mathrm{C}|\mathrm{T}) \times P(\mathrm{P}|\mathrm{C}) \times P(\mathrm{D}|\mathrm{S})}{P(\mathrm{M})}$$

The second potential advantage derives from the generative aspect of the proposed model. While there is an analogue in conventional IE systems for each of stages 2 through 4 described above, there is no conventional analogue to stage 1: the prior model. We can think of this prior model as encoding domain-specific world knowledge about the plausibility of proposed sets of relations.

8 Conclusions

We have shown that a simple statistical model can identify semantic slot-fillers in a management succession task with 83% accuracy (F-measure with a score of 0.5 for partial matches). The system was trained on only 560 sentences, with the additional requirements of only a part-of-speech tagger and a morphological analyser. We initially considered a finite-state approach similar to that used for POS tagging (Church 88), or named-entity identification (Bikel et al. 97), but argued that the Markov approximation gives a poor model for this task. The alternative, which has a PCFG component to define the probability of the underlying sequence of labels, allows a good parameterization of the problem, and can be decoded efficiently using the CKY algorithm. Finally, we believe that the framework presented in this paper can be extended to model the complete information extraction process.

Acknowledgements

We would like to thank Richard Schwartz and Ralph Weischedel for many helpful discussions and suggestions concerning this work. We would also like to thank the anonymous reviewers for several useful comments.

References

D. Appelt, J. Hobbs, J. Bear, D. J. Israel, and M. Tyson. 1993. FASTUS: a finite-state processor for information extraction from real-world text. In *Proceedings of IJCAI-93*, (Chambery, France), September 1993.

D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. 1997. Nymble: a High-Performance Learning Name-finder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 194-201.

M. E. Califf and R. J. Mooney. 1997. Relational Learning of Pattern-Match Rules for Information Extraction. In *Proceedings of the ACL Workshop on Natural Language Learning*, Madrid, Spain.

K. Church. 1988. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *Second Conference on Applied Natural Language Processing, ACL.*

R. Grishman. 1995. The NYU System for MUC-6 or Where's the Syntax? In *proceedings of the Sixth Message Understanding Conference*, Morgan Kaufmann.

F. Jelinek. 1990. Self-organized Language Modeling for Speech Recognition. In *Readings in Speech Recognition*. Edited by Waibel and Lee. Morgan Kaufmann Publishers.

Daniel Karp, Yves Schabes, Martin Zaidel and Dania Egedi. A Freely Available Wide Coverage Morphological Analyzer for English. In *Proceedings of the 15th International Conference on Computational Linguistics*, 1994.

W. Lehnert, J. McCarthy, S. Soderland, E. Riloff, C. Cardie, J. Peterson, F. Feng, C. Dolan, and S. Goldman. 1993. University of Massachussets/Hughes: Description of the CIRCUS system as used for MUC-5. In *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, pages 277-290.

M. Marcus, B. Santorini and M. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313-330.

S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz, R. Stone, R. Weischedel and the Annotation Group. 1998. *Algorithms that Learn to Extract Information. BBN: Description of the SIFT System as used for MUC-7.* In Proceedings of the Seventh Message Understanding Conference.

Proceedings of the Third, Fourth, Fifth and Sixth Message Understanding Conferences (MUC-3, MUC-4, MUC-5 and MUC-6). Morgan Kaufmann, San Mateo, CA.

A. Ratnaparkhi. 1996. A Maximum Entropy Model for Part-Of-Speech Tagging. *Conference on Empirical Methods in Natural Language Processing*, May 1996.

E. Riloff. 1993. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC. AAAI Press / MIT Press. 811-816.

E. Riloff. 1996. Automatically Generating Extraction Patterns from Untagged Text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR. AAAI Press / MIT Press. 1044-1049.

S. Soderland, D. Fisher, J. Aseltine and W. Lehnert. 1995. CRYSTAL: Inducing a Conceptual Dictionary. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence.* AAAI Press / MIT Press. 1314-1319.

Yarowsky, D. 1995. 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*. Las Cruces, NM, pp. 88-95, 1994.