UNIVERSITY OF CALIFORNIA, SAN DIEGO

Information Extraction
Using Hidden Markov Models

A thesis submitted in partial satisfaction of the requirements for the degree Master of Science

in Computer Science

by

Timothy Robert Leek

Committee in charge:

Professor Charles Peter Elkan, Chairperson
Professor Larry Carter
Professor Terrence Sejnowski

1997

The Masters Thesis of Timothy Robert Leek is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

1997

TABLE OF CONTENTS

ABSTRACT OF THE THESIS

Information Extraction Using Hidden Markov Models

by

Timothy Robert Leek
Master of Science in Computer Science
University of California, San Diego, 1997
Professor Charles Peter Elkan, Chair

This thesis shows how to design and tune a hidden Markov model to extract factual information from a corpus of machine-readable English prose. In particular, the thesis presents a HMM that classifies and parses natural language assertions about genes being located at particular positions on chromosomes. The facts extracted by this HMM can be inserted into biological databases. The HMM is trained on a small set of sentence fragments chosen from the collected scientific abstracts in the OMIM (On-Line Mendelian Inheritance in Man) database and judged to contain the target binary relationship between gene names and gene locations. Given a novel sentence, all contiguous fragments are ranked by log-odds score, i.e. the log of the ratio of the probability of the fragment according to the target HMM to that according to a "null" HMM trained on all OMIM sentences. The most probable path through the HMM gives bindings for the annotations with precision as high as 80%. In contrast with traditional natural language processing methods, this stochastic approach makes no use either of part-of-speech taggers or dictionaries, instead employing non-emitting states to assemble modules roughly corresponding to noun, verb, and prepostional phrases. Algorithms for reestimating parameters for HMMs with non-emitting states are presented in detail. The ability to tolerate new words and recognize a wide variety of syntactic forms arises from the judicious use of "gap" states.

Chapter I

Good Facts Are Hard to Find

Finding facts in English prose is a task that humans are good at and computers are bad at. However, humans cannot stand to spend more than a few minutes at a time occupied with such drudgery. In this respect, finding facts is unlike a host of the other jobs computers are currently hopeless at, like telling a joke, riding a bike, and cooking a dinner. While there is no pressing need for computers to be good at those things, it is already of paramount importance that computers be proficient at finding information with precision in the proliferating archives of electronic text available on the Internet and elsewhere. The state of the art in information retrieval technology is of limited use in this application. Standard boolean searching, vector-based approaches and latent semantic indexing are geared more toward open-ended *exploration* than toward the targeted, detailed subsentence processing necessary for the fact finding or *information extraction* task. Since these approaches discard syntax, a large class of targets, in which the relationships between groups of words are important, must be fundamentally beyond them. The critical noun and verb groups of a fact can only be found by doing some kind of parsing.

Information extraction is in most cases what people really want to do when they first set about searching text, i.e. before they lower their sights to correspond to available tools. But this does not mean that nothing less than full-blown NLP (natural language processing) will satisfy. There are many real-world text searching tasks that absolutely require syntactic information and yet are restricted enough to be tractable. An historian might want to locate passages in the Virginia colony records mentioning the "event" of a slave running away. The words *slave*, *run*, and *away*, all very common words, and their various synonyms used in an unconstrained search would return much dross. To find this fact with precision we need to place constraints upon the arrangement of the words in the sentence; we need to limit the search with syntax. For instance, one might require that when two groups of words corresponding to *slave* and *run* appear in a sentence, that the slave is in fact the one doing the running. Similar examples of what we call *fact searching* are commonplace in most domains. A market analyst might want to scan the Wall Street Journal and pick out all mentions of corporate management changes. And a geneticist would be thrilled to be able to tease out of scientific abstracts facts mapping genes to specific locations on chromosomes.

Historically, the field of information extraction has employed discrete manipulations in order to process sentences into the critical noun and verb groups. An incoming sentence is tagged for part-of-speech and then handed off to a scaled-down parser or DFA (deterministic finite automaton) which uses local syntax to decide if the elements of a fact are present and to divide the sentence up into logical elements. Recent advances in statistical natural language processing have been applied to this problem but typically only in an ancillary role, e.g. in constructing dictionaries [17] and tagging words for part-of-speech [4]. The main processing engine remains combinatorial in flavor. Systems like FASTUS [8] and CIRCUS [14] do surprisingly well, considering the difficulty of the task, achieving precision and recall of better than 80%. But they require hand-built grammars or dictionaries of extraction patterns in order to attain this level of performance. A notable exception is the LIEP [9] system which learns to generalize extraction patterns from training examples.

We have chosen to pursue a unified stochastic approach to the information extraction task, modeling sentence fragments containing the target fact with a hidden Markov model (HMM) which we use both to decide if a candidate sentence fragment contains the fact and also to identify the important elements or slot fillers in the fact. An HMM trained to recognize a small set of representative sentence fragments differs radically from a DFA or discrete pattern matcher designed for the same task in that it outputs a probability. Unlike a DFA, an HMM will accept *any* sequence of words with non-zero probability. The probability it computes (after some corrections for sentence length and background frequencies of words) varies gracefully between the extremes of predicting extremely low probability for sequences that tend not to contain the fact to predicting high probability for ones that tend to contain it. There is no need, if we use an HMM to find and process facts, to employ heuristics in order to rank and choose between competing explanations for a sentence; symbolic approaches often do so [9]. The probability the HMM computes is meaningful information we can use directly to reason about candidate facts in principled ways that submit to analysis.

The HMM is a very compact and flexible representation for the information extraction task which seems to be less reliant upon human engineering and prior knowledge than non-probabilistic approaches. This thesis will discuss our efforts to construct a model for a binary relationship between gene names and gene locations, as found in a variety of syntactic forms in scientific abstracts. The model is structured hierarchically: at the top level states are collected into modules corresponding to noun or verb groups, whereas at the bottom level, in some cases, states function entirely deterministically, employing DFAs to recognize commonly occurring patterns. The HMM consists of only 64 states with an average of 3 transitions each, and explicitly mentions less than 150 words. When deploying the model to find facts in novel sentences, no attempt is made to tag for part-of-speech. “Gap” states, which assign emission probability according to word frequency in the entire corpus, permit the HMM to recognize disconnected segments of a fact and tolerate new words. Unknown words, if they appear in the right local context, are accepted by the HMM essentially without penalty. So while the list of words likely to participate in forming a gene name or gene location is long and populated by words *both* common and rare to the corpus our approach is competent at correctly identifying even unknown words as long as they appear flanked by other words that serve to index the fact well. The accuracy of this HMM approach to information extraction, in the context of the gene name—location fact, is on par with symbolic approaches.

This thesis is organized as follows. We begin with a description of the gene name—location information extraction task. Next, we present the modular HMM architecture constructed for this task, motivating our choice of null or background model and demonstrating the discriminatory power it adds to this approach. A brief technical discussion comes next, of the precise formulae used to reestimate parameters for an HMM with non-emitting states. Then we provide implementation and optimization details, followed by training and testing performance. We conclude with some remarks on the use of prior knowledge and ideas for future work.

Chapter II

Automatic Annotation Generation

We consider the question of finding facts in unrestricted prose in the context of filling in slots in a database of facts about genes. The slots in the database correspond to biological entities. These are described by single words or simple phrases, three examples of which might be the name of a gene, some specification of its location, and some list of diseases in which it is known to be involved. An example pair of acceptable entries is

SLOT	ENTRY
Gene Name:	(The gene encoding BARK2)
Gene Location:	(mouse chromosome 5)

which we might find buried in a sentence like

```
The gene encoding BARK2 mapped to mouse chromosome 5, whereas
that encoding BARK1 was localized to mouse chromosome 19.
```

This is valuable information that is available nowhere except in the published literature. Specialized databases like SwissProt and GenBank do not contain these kinds of associations. So there is interest in developing automated systems for filling in these slots.

In order to populate these slots, we must locate and correctly analyze binary (or perhaps even ternary and higher) relations between likely elements as they appear in English prose. Finding just a gene name might reasonably trigger the creation of a new database record, but only when we find the name explicitly related to a gene location in some small set of valid syntactic arrangements are we able to conclude that the two should appear in the same record. This gene name—location fact is the one we have chosen to try to capture first, taking as our natural language source of potential facts the collected abstracts of the OMIM (On-line Mendelian Inheritance in Man) database, almost 20MB of text. This application is a good place to start for the following reasons.

- **Relative Simplicity:** This fact is a simple binary relation on two noun groups, the simplest possible fact by our definition. We will demonstrate that HMMs are powerful enough to represent and capture this concept first before moving on to more complex facts involving higher-order relations.

- **Limited Grammar:** Authors tend to express the gene name—location fact in the same ways over and over. There is little variety in the syntax at a high level. Most of the difference between individual facts is at a lower level, i.e. in the subtle construction of the phrases that correspond to gene name and location. So a simple HMM, which has machinery to produce complicated gene names and locations but is limited in the number of ways in which it can combine those larger elements, should be all that is needed to process the vast majority of facts found in practice. It is not necessary to attempt to model all valid sentence fragments if only a small subset are typically employed by real authors.

- **Single Register:** The goal here is to isolate gene name—location facts in technical prose. More specifically, we want to be good at processing a very restricted class or *register* of prose: scientific abstracts such as are found in the OMIM database. We are not aiming to come up with a classifier that will be good at deciding that this fact is *not* in the text of Harper's Weekly or the Dictionary of Literary Biography. By limiting the task in this way (a human faced with the same task would surely do similar preprocessing) we can expect to do better.

- **Uncommon Words:** This particular task is a difficult one for traditional NLP methods in that many of the words we must succeed in categorizing correctly are in no dictionary. Gene names tend to consist of an irregular jumble of biomedical jargon and acronyms. Phrases like *adrenal medulla cardiac ventricle* and tokens like *IL1RN* and *Prim-2* are all valid parts of gene names. Gene locations are a similarly mixed bag. While we can use statistics from *common* words to infer much about syntax and semantics, we must deal differently with words sparsely represented in the corpus. Uncommon elements we have to treat *en masse*, as gaps or holes. Fortunately, stochastic approaches such as hidden Markov models demonstrate impressive tolerance for noisy sources and missing information in other domains such as speech recognition and protein family characterization [16, 12].

Chapter III

Hidden Markov Models for Facts

HMMs have long been used to model speech, and they have much to recommend them for the fact-finding problem. They offer a compact representation of word and word co-occurrence probabilities, which have been shown to carry information about syntax and semantics in English [5, 6, 13]. Further, efficient algorithms exist for learning model parameters from sequences of words, for computing the probability of a sequence given the model, and for finding the highest probability path through the states of the model for a given sequence of words. An HMM trained on a set of example sentence fragments known to contain a fact can be used to classify candidate sentence fragments according to whether or not each contains the fact. The same HMM will also serve for ranking the fragments by probability. Furthermore, if we design the model with care, it should be able to tease apart the actual components of the fact. An HMM can give us words to fill in the annotation slots, for instance, of gene name and gene location.

One might proceed in designing an HMM for a task such as the fact finding one by careful construction relying heavily upon linguistic knowledge. But we are computer scientists, not linguists, and we chose an approach that allowed the data to dictate most of the structure of the solution. We began with a reduced version of the top level model pictured in Figure III.1, consisting only of *four* modules: VERB1, X1, VERB2, and Y1. For each training example, we determined the lists of words

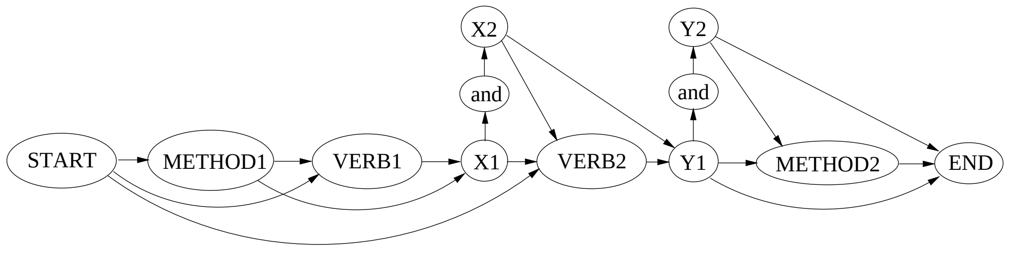
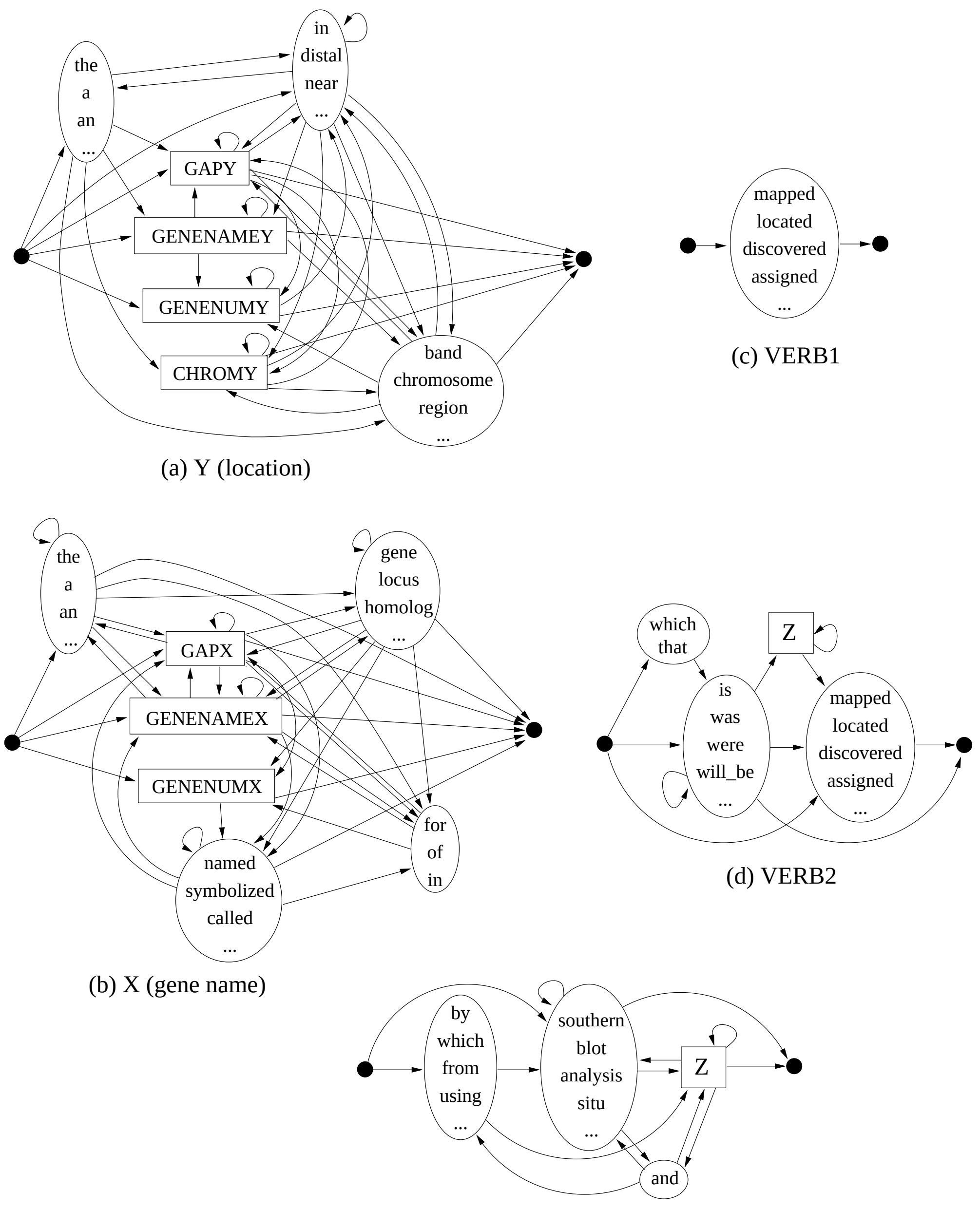


Figure III.1: Top level HMM for gene name—location fact

that must be accepted by each of the modules in order for the HMM to accept the entire example. Then, for each module, we divided the list of words it must accept into states containing words with similar function and/or meaning. As we proceeded through the training set, it became clear that a few more modules were in order. We added METHOD1 and METHOD2 modules to better align the HMM to the critical portion of the fact in those cases in which a statement of the method of discovery accompanied the statement of discovery itself. Additionally, we noticed that many examples contained parallelism, and replicated X1 and Y1 modules as X2 and Y2, connecting them in the configuration depicted in Figure III.1 in order to be able to extract this additional information. At the top level, we limited the allowable transitions between modules, i.e. the modules were not initally fully connected. This is because most modules contain gap states, and without constraining the order in which modules are employed in accepting an example, we believed it would be possible, e.g. for a gene name to be accepted as a location and vice versa. The modules themselves we allowed to be internally fully connected.

A schematic diagram of the final HMM appears in Figure III.1. METHOD1, METHOD2, VERB1, VERB2, X1, X2, Y1, Y2 represent modules that are given in detail in Figure III.2. START and END states stand for the start and end of a sentence. The METHOD module recognizes a phrase that describes the method by which a scientist has discovered a gene in a specific location, e.g.

- `By Southern analysis of DNA from human-hamster hybrid cell lines`



(a) Y (location)

(c) VERB1

(b) X (gene name)

(d) VERB2

(e) METHOD

Figure III.2: HMM Modules for Gene Name - Location Fact: filled dots represent non-emitting states (see Chapter V)

- `... using both a somatic cell hybrid mapping panel and fluorescence in situ hybridization.`

These modules contain no part of the fact, that is, no words we might want to put in the annotation slots. They are nevertheless important because they represent evidence that the fact is present in a given sentence, and they serve to align the model better to isolate the actual fact.

At the top level, our HMM is a linear model, i.e. a directed acyclic graph. There are no loops. Its behavior, therefore, is to parse a sentence and look for the elements of the *fact* strictly from left to right. The internal machinery of the various modules is designed to afford some productive power with respect to the individual elements of each annotation. This is a strong restriction for it is not difficult to invent sentences that express the fact but that the HMM as designed is unable to process. In particular any sentence in which the gene name comes after the gene location cannot be parsed, e.g.

```
It was in 2p11 that Balfour et al. discovered the BARK1 gene.
```

We are willing to accept this poverty in our model for two reasons. First, the form of the fact represented in Figure III.1 is overwhelmingly the most common one. Geneticists strongly prefer to express this information in the simpler, more declarative form. We place utility and efficiency above completeness; the ability to process this fact with good accuracy in the vast majority of the sentences that actually contain it means that we are not particularly troubled by the small number our model misses. Second, once the modules are trained and we are happy with their performance in the context of Figure III.1, then we can consider expanding that model to support alternate forms. The linearity of the model seems to require that each of the modules works well independently of the others; they do not interact. So cutting them out and gluing them together in novel configurations ought to work.

The precise configuration of VERB1 and VERB2 at the top allows a single model correctly to analyze all three of the following:

- `Beeson et al. (1993) demonstrated that the CHRNE gene is located on chromosome 17.`
- `Hiraoka et al. (1995) mapped the ALDH5 gene to 9p13.`
- `... the human filaggrin gene maps to 1q21.`

The set of verbs in the VERB1 module is the same as those in the second state in the VERB2 module. This duplication of the main verb state is necessary. For if it were possible to make a transition between X1 and Y1 that made use of one of these main verbs without consuming an auxiliary verb first, then in the second sentence above the HMM would give high probability to a parse of the sentence that assigned "`Hiraoka et al. (1995)`" to X1 (the gene name) and "`the ALDH5 gene to 9p13`" to Y1 (the gene location).

The X and Y modules detailed in Figure III.2 recognize gene name and gene location information, respectively. Both their employment at the top level in the HMM and their internal construction are crucial to the HMM's ability to isolate facts. At the top level, it is noteworthy that there are in fact two of each. Parallel constructions are common and a model fashioned in the manner depicted can successfully pair X1 and Y1 to X2 and Y2. Given a sentence like

```
The HKE4 (601416) and HKE6 genes are located in the major
histocompatibility complex in mouse and human, on mouse
chromosome 17 and human chromosome 6, respectively.
```

the HMM isolates *two* gene name—location facts. The underlying assumption is that the word "and" separates two different gene names and locations instead of perhaps merely dividing two parts of the same entity. Once again, in practice this is the most common situation, and we allow domain knowledge to inform the construction of the model.

There are five varieties of gap states which function identically in X and Y modules. If we are using the HMM to compute the probability of a sequence of words, then these states accept words with probability 1.0 according to the following rules.

1. The Z gap state accepts words that are neither part of the fact nor useful for recognizing it. Any word will match this state, with the following restrictions:
   (a) A Z gap state cannot match a word that would be accepted by any of the GENENAME, GENENUM, or CHROM gap states.
   (b) A Z gap state cannot match a word that appears explicitly in the model, e.g. `southern`, `analysis`, or `mapped`, unless it is from the *common* word list (see below).
2. The GAPX and GAPY gap states match uncommon words that are part of the gene name or location, e.g. `the gene for` *achondroplasia* `and` *hypochondroplasia*. Matching in this state is restricted in the same way as in the Z state, except that common words are not allowed; we require that words filling this gap be uncommon and therefore informative ones.
3. The GENENAME gap accepts words containing two or more capital letters, e.g. `BARK1` and `D11S750`. Many genes have names of this form and this heuristic is rough but surprisingly effective.
4. The GENENUM gap accepts natural numbers 6 digits long. These numbers appear frequently in sentences containing the gene name—location fact. They are references to a GENBANK entry.
5. The CHROM gap state accepts formulaic chromosome locations, like `16q12-q22`, `11q`, and `22q13.1` by the following set of productions

$$\begin{aligned} PQ &\rightarrow \mathtt{p}|\mathtt{q} \\ num &\rightarrow [\mathtt{0}-\mathtt{9}]+ \\ chrom &\rightarrow \mathtt{cen} \mid PQ\ num \mid num\ PQ \mid PQ\ \mathtt{ter} \\ &\quad \mid \mathtt{X}\ chrom \mid chrom\ num \mid num\ chrom \mid chrom\ .\ num \\ &\quad \mid chrom\ -\ chrom. \end{aligned} \tag{III.1}$$

Both Z and GAPX, GAPY states use a list of 77 common words, which are uninformative pronouns, articles, and verbs, e.g. `he`, `the`, and `is`. We compiled this list by hand but it is likely very close to what one might generate by taking the highest ranking words in a large corpus of text including examples from many different registers. There are no words in the list that are indicative of register.

Replacement of individual words with tokens in training data was necessary in order to train the model. So the sentence

```
The TCP1 gene (186980) is located on 6p in the vicinity of
the major histocompatibility complex, and the murine
homolog, Tcp-1, is located in the t-complex region of mouse
chromosome 17.
```

appears in the training set as *two* tokenized sentence fragments, each containing one gene name—location fact

```
The GENENAMEX1 gene GENENUMX1 is located on CHROMY1 in the
vicinity of the GAPY1 GAPY1 GAPY1
the GAPX1 homolog GAPX1 is located in the GAPY1 region of GAPY1
chromosome GAPY1
```

The model must contain abstract states if it is to be tolerant of novel words and if we haven't the time or space to try to model all of English. During training, these tokenized gap words are treated exactly like the other words in the model. The probability of emitting a GAPX1 token is 1.0 in the GAPX1 state. Without this kind of tokenizing, transitions to gap states with freedom to match most words (GAPX1 and Z gaps for instance) train to 1.0 and all other transitions relax to zero probability. By labeling gaps in the training data, we are able to learn the distribution of the various types of gaps by learning the transition probabilities into the states that accept them. Much of the work of tokenizing gap words is done automatically. GENENAME, GENENUM, and CHROM tokens are recognized by simple rules and therefore are assigned automatically. X1, Y1 suffixes we append by hand in the training data. Notice that the rules for recognizing token types, described earlier, are captured easily by a DFA which we can think of as a special case of an HMM with all emission and transition probabilities within a state set equal. So the construction of the Information Extraction model is nicely hierarchical in that it is composed of modules, and recursive in that the internals of lower levels are made up of the same machinery as higher levels.

In many statistical NLP tasks employing HMMs, e.g. machine translation and speech recognition [5], researchers have chosen to represent trigram statistics, $Pr(w_3|w_1w_2)$, directly, by having emission probabilities be of word pairs, or bigrams. In our HMM we do not make use of bigram probabilities. States contain emission probabilities for individual words, and, given the transition probability matrix, we can estimate bigram probabilities by summing over all states,

$$Pr(w_2|w_1) \approx \sum_{i,j=1}^{N} Pr(w_2, state_j|w_1, state_i) = \sum_{i,j=1}^{N} b_j(w_2)a_{ij}b_i(w_1). \tag{III.2}$$

In our HMM, there is only limited duplication of words in different states, meaning that this sum in most cases consists of only one or two terms. On the one hand the estimate of bigram probability we might make using the HMM and Equation III.2 is certainly less accurate than if it were estimated directly from the corpus as

$$Pr(w_2|w_1) \approx \frac{Counts(w_2, w_1)}{Counts(w_1)}. \tag{III.3}$$

On the other hand, it is not the sum in Equation III.2 that interests us here but rather the terms themselves, which represent context-dependent bigram probabilities that are significant and ought to remain indivisible. It is important, for instance, to have a *different* bigram probability for the pair `[the,locus]` in the X module than in the Y module, and this is precisely what the HMM provides us, if only implicitly. Of course, the most compelling reason to use an HMM that employs unigram probabilities instead of bigram probabilities is the difficulty of finding training examples. If we wanted to use bigram counts as in Equation III.3 this would increase the number of model parameters by some constant exponent (the number of states would grow from N to N^b where $b \leq 2$), and therefore require proportionately more training data.

Decisions about top-level model architecture represent prior knowledge that we specify before training. These priors take the form of duplicated states and sparse initial connections (if a word or a transition is not explicitly listed, its probability is assumed to be zero) *between* modules (modules are fully connected internally prior to training). If we avail ourselves of no prior knowledge about English then we must begin with a fully connected model in which all words may be emitted in all states. Even in this case we must at least decide on the number of states beforehand. From such a starting point the model one converges to by training is entirely useless at recognizing and processing prose to find the gene name—location fact.

The final, trained HMM consists of only 65 states, sparsely connected with an average of 8 non-zero probability transitions each. We trained this model with an implementation, written in C, of the Baum-Welch method as described in [16], with extensions as detailed in Chapter V.

Chapter IV

The Importance of a Null Model

Using an HMM to compute sentence probabilities and, more specifically, having emission probabilities for individual words instead of for word pairs in each state reflects certain independence assumptions. In order to make sensible use of the probabilities the HMM computes, we must understand the effect of these assumptions in detail. The probability of a sentence of T words given an HMM, λ, is the sum of the probabilities of the P paths, $\phi_i, 1 \leq i \leq P$, through the model that assign the sentence nonzero probability:

$$Pr(w_1 w_2 ... w_T | \lambda) = \sum_{i=1}^{P} Pr(w_1 w_2 ... w_T | \phi_i). \qquad \text{(IV.1)}$$

The probability of a sentence for a given path ϕ_i through the model is computed as the product of the word emission $Pr(w_t | \phi_i(t))$ and state transition probabilities $a_{\phi_i(t-1) \rightarrow \phi_i(t)}$, traversed along that path through the states of the model, i.e.

$$Pr(w_1 w_2 .. w_T | \phi_i) = \pi_{\phi_i(1)} Pr(w_1 | \phi_i(1)) \prod_{t=2}^{T} Pr(w_t | \phi_i(t)) a_{\phi_i(t-1) \rightarrow \phi_i(t)}, \qquad \text{(IV.2)}$$

where π_j is the probability that j is the start state. Probability estimates obtained in this multiplicative manner are strongly correlated with sentence length. A shorter sentence will get higher probability merely by virtue of the fact that it must suffer fewer transitions and word choices. What is more useful than trying to deal in probabilities directly is to consider a ratio of probabilities, the odds of a fact being present given a particular sentence versus the fact not being present.

$$\text{odds} = \frac{Pr(\lambda|sentence)}{Pr(\lambda_\emptyset|sentence)} = \frac{Pr(sentence|\lambda)}{Pr(sentence|\lambda_\emptyset)}\frac{Pr(\lambda)}{Pr(\lambda_\emptyset)} \tag{IV.3}$$

Here, $\lambda_\emptyset$ is some background or "null" model for sentences not containing the target fact. If both $Pr(\lambda|sentence)$ and $Pr(\lambda_\emptyset|sentence)$ are computed by HMMs that exhibit similar sentence length dependence then these effects will cancel one another, and we can use $odds(sentence)$ both to decide if a sentence contains the fact and to rank candidate facts from sentences of arbitrary length. If $odds(sentence) < 1$ this indicates that the fact is not present, and $odds(sentence) > 1$ means it is. Intuitively, sentences for which $odds(sentence)$ is *higher* contain *more* of the words and subsequences of words that are statistically favored by the fact model, i.e. that are assigned probabilities significantly higher than in OMIM as a whole. In practice, it is more useful to rank sentences by log-odds score.

$$\text{log-odds}(sentence) = \log\left(\frac{Pr(sentence|\lambda)}{Pr(sentence|\lambda_\emptyset)}\right) - \theta. \tag{IV.4}$$

where the threshold value θ is the log-odds of the priors, i.e. the background prob-

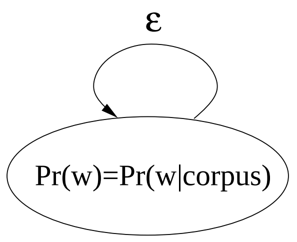


Figure IV.1: Simple Null Model: Word emission probabilities are unigram probabilities for the corpus. Looping transition probability, ϵ, is set to 0.9.

abilities of the finding the fact versus not finding the fact. In practice, we find that about five in a hundred sentences contain the fact, which means that we can estimate $Pr(\lambda)$ at 0.05. So an HMM constructed to model the entirety of the corpus should adequately model the absence of the fact as well. A simple null model, shown in Figure IV.1, consists of just one state and a looping transition with probability ϵ.

Word emission probability is the background probability of the word in the corpus. In this case, the log-odds score for a sentence of T words is given by

$$\text{log-odds}(sentence) = \log(Pr(sentence|\lambda)) - \sum_{t=1}^{T} \log(Pr(w_t)) - T \log \epsilon - \theta. \tag{IV.5}$$

This null model is a good choice because it decreases the impact of words that are common both to sentences containing the fact *and* to sentences not containing the fact. An example sentence will illustrate the effect of the null model upon log-odds score. Consider the pair of sentence fragments

- `mapped CYP3A4 to 7q22.1 by fluorescence in_situ_hybridization.`
- `isoforms of the beta subunit are encoded by a single gene.`

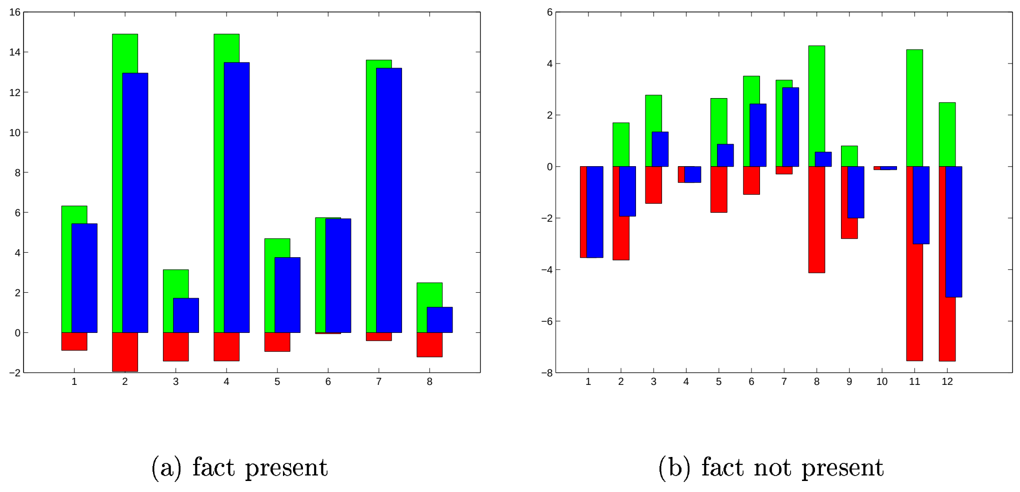


Figure IV.2: Word-by-word contribution to log-odds score for two sentence fragments. Heights of black bars indicate the total contribution of each word. Heights of light grey bars indicate contributions due to word emission probabilities. Differences in height between the two are due to transition probabilities in the fact model versus in the null model. These negative contributions are also plotted as the medium grey bars descending below the zero-line. The final bar in each plot is the contribution due to the period that ends each sentence.

The first fragment contains the gene location fact, and the second does not. The second does, however, contain many words (`subunit`, `encoded`, `gene`) that are statistically favored by the fact model. If considered without regard to syntax they imply strongly that the fact is present. The total contribution to the log-odds score of each word is plotted for both fragments in Figure IV.2 as the heights of the black bars. The heights of the lightest grey bars is the log of the ratio of individual word emission probabilities in the fact model (for the most probable path only) to those in the null model, i.e. the portion of the contribution to log-odds score that is due only to word emission probabilities. The effect of words common both to gene name—location facts and to OMIM as a whole is small because we deal only in relative probability. This means that a word like `mapped`, the first bar in the first plot, can contribute more to the log-odds score than a much more common word like `to`, the third bar in the first plot.

For every word, the total contribution to log-odds score (the black bars) is less than it would be if only word emission probabilities counted. This difference is due to the fact model's syntactic preferences. In the null model, $\epsilon = 0.9$ (which means that the expected sentence length is 10 words), which is larger than most of the transition probabilities learned in either our top-level HMM or the modules. Since the ratio of transition probability incurred by the fact model to that incurred by the null model is almost always less than one, syntax constraints generally serve to decrease the log-odds score.

These syntactic contributions appear on the plot as the medium grey bars descending below the zero-line for each word. The decrease is much *less* for fragments that contain the fact (are syntactically likely according to the model) than for those that do not. If we were merely to sum up the heights of the light grey bars for each of these two examples, we would assign the fragments scores of 65.8 and 26.50. Both of these scores are considerably above the threshold of 3.0, the log of the ratio of the priors, and so would be judged to contain the fact based on semantic evidence alone. The negative contributions due to relative transition probabilities, i.e. due to local syntax, bring the scores down to 57.5 and -8.0, respectively. The sentence that contains the right words in an arrangement not statistically favored by the fact model gets a score far below the decision threshold.

Chapter V

Learning Parameters in an HMM with Non-Emitting States

Our HMM includes non-emitting states, i.e. states that are traversed *without* consuming a word. These states are depicted in Figure III.2 as filled-in dots. Non-emitting states permit us to reduce the total number of model parameters, in particular we need fewer transition probabilities, by grouping states into modules roughly corresponding to noun, verb, and prepositional phrases, as described in previous chapters. There is only one way into a module, through a non-emitting state. Likewise, the only way out of a module is via a non-emitting state. Thus it is possible to snip a module like x1, which recognizes a gene name, out of the HMM by deleting only the transitions into and out of its non-emitting entrance and exit. This modularity forces state transitions at different levels in the model to assume different roles. Transition probabilities learned *within* modules capture local syntax, e.g., the structure of a noun phrase like x1. Transitions *between* modules reflect syntactic features and tendencies that are at higher levels, e.g. the statistics of alternate constructions like active versus passive voice, and optional information like a second gene name or a phrase describing the method of discovery.

The price of simplifying the model is some added complexity in the standard procedures for learning parameters and determining the most likely (Viterbi) path.

Consider the standard algorithm [16] for computing the so-called *forward* variable

$$\alpha_t(i) = Pr(w_1 w_2 ... w_t, \phi(t) = S_i|\lambda) \tag{V.1}$$

which is the probability of saying the first t words in the sentence and ending up in state S_i at time t, given a model λ consisting of N states. Precomputation of $\alpha_t(i)$ permits both efficient calculation of the probability of a sentence of T words given the model,

$$Pr(sentence|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{V.2}$$

and reestimation of model parameters. This forward variable we compute inductively, memoizing its values in a matrix of T columns and N rows:

$$\begin{aligned} \alpha_1(i) &= \pi_i b_i(w_1) \qquad 1 \le i \le N \\ \alpha_{t+1}(j) &= \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(w_{t+1}) \qquad 1 \le t \le T-1, \qquad 1 \le j \le N. \end{aligned} \tag{V.3}$$

where a_{ij} is the probability of making a transition from state i to state j, and $b_j(w_t)$ is the emission probability for word w_t in state j. We must make some modifications to this formula if we want to calculate $\alpha_t(i)$ for an HMM that contains non-emitting states. Figure V.1 is a dynamic programming-style grid illustrating the possible path sequences through the model for a short fragment of prose. We have chosen to memoize α values for transitions that begin with non-emitting states as being between rows in the *same* column of the grid, i.e. the same t value and therefore not consuming a word. Conversely, transitions starting with words and ending in non-emitting states are *between* columns in the grid, from t to $t+1$. This commitment is arbitrary but necessary, if we are to reformulate V.3 systematically in order to accommodate non-emitting states. But the recurrence we find to capture the mechanism underlying Figure V.1 will only terminate if we place a limit on the number of transitions we can make between non-emitting states before entering a word emitting state. In fact there is just such a limit implicit in the architecture of our HMM. Non-emitting

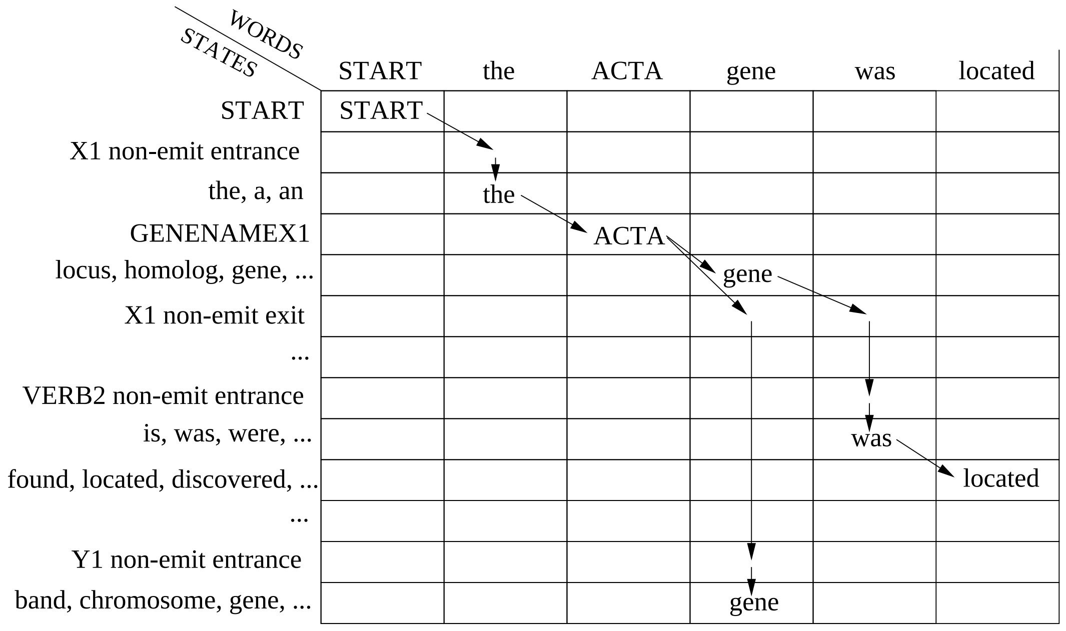


Figure V.1: Possible paths through an HMM with non-emitting states

states appear only as the entrances and exits of modules. So we can traverse *at most two* non-emitting states in between consuming words from a sentence. Given this limitation, we can rewrite the recurrence in Equation V.3 to work for models with non-emitting states.

$$\begin{aligned}
\alpha_{t+1}(j) &= \left[\sum_{i=1}^{N} I(i)\alpha_t(i)a_{ij}\right] b_j(w_{t+1}) \\
&+ \left[\sum_{i=1}^{N} |1-I(i)| \left[\sum_{k=1}^{N} I(k)\alpha_t(k)a_{ki}\right] a_{ij}\right] b_j(w_{t+1}) \\
&+ \left[\sum_{i=1}^{N} |1-I(i)| \left[\sum_{k=1}^{N} |1-I(k)| \left[\sum_{l=1}^{N} I(l)\alpha_t(l)a_{lk}\right] a_{ki}\right] a_{ij}\right] b_j(w_{t+1})
\end{aligned} \tag{V.4}$$

$$I(i) = \begin{cases} 1 & \text{if } i \text{ is an emitting state} \\ 0 & \text{otherwise} \end{cases}$$

$$1 \le t \le T-1, \qquad 1 \le j \le N.$$

The first line in this new recurrence relation is equivalent to Equation V.3, for which all states were assumed to be emitting. The second and third lines in Equation V.5 are new terms and they represent contributions to $\alpha_{t+1}(j)$ due to words consumed after one and two intervening non-emitting states, respectively.

If we use this new recurrence relation to fill in a table like Figure V.1 then we will have to do more work than if we used Equation V.3. Assume the worst case, that the model is fully connected. And take f to be the fraction of states in the model that are non-emitting. We will have to do $(1-f)f^2N^3 + (1-f)fN^2 + (1-f)N$ multiplications of some transition probability by some $\alpha_t(i, k,$ or $l)$ to compute each $\alpha_{t+1}(j)$. For our model, there are 16 non-emitting states and 59 emitting states, so $f_n = 0.25$. To compute each $\alpha_t(j)$ for our model, then, if it were fully connected, we would need to do 13377 multiplications. Our model is sparsely connected, though. If we take N_{ne} to be the average number of transitions into a non-emitting state, and N_e to be the average number of transitions into an emitting state, then the number of multiplications necessary is $N_{ne}^2N_e + N_{ne}N_e + N_e$. In our model N_{ne} is about 2.6 and N_e is about 2.9, which means that we might expect to do only about 30 multiplications in order to compute each $\alpha_{t+1}(j)$. In practice we find that much fewer multiplications are necessary since typically only a small number of $\alpha_t(i)$ are nonzero for each t, i.e. less than five. Notice that the amount of computation, per $\alpha_{t+1}(j)$ in such a sparsely connected model is a function of N_e and N_{ne}, and not of N. So to compute $\alpha_{t+}(j)$ for all j and t, we must fill in every cell in a table like that in Figure V.1, and the total work to fill in such a table is less than $30TN$, and in practice probably more like $5TN$ multiplications.

The inductive procedure which generates the backward variable $\beta_t(i)$, the probability of saying words $t+1$ through T given state S_i at time t and the model λ, must be modified similarly. And the algorithm for determining the Viterbi path, the most probable path through the model given the sentence, which is also naturally implemented with dynamic programming, must be expanded as well. Lastly, the formula for reestimating transition probabilities a_{ij} needs a slight adjustment for transitions *starting* with non-emitting states. The reestimation formula in the case without non-emitting states is given by

$$\overline{\alpha}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{Pr_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(w_{t+1}^k) \beta_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{Pr_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \tag{V.5}$$

where $Pr_k = Pr(sentence_k|\lambda)$ is the probability of training sentence k given the HMM, computed as in Equation V.2. For transitions starting from non-emitting states, the relevant α and β values will have been memoized in the same t column, so we must update those transition probabilities with a slightly different formula:

$$\overline{\alpha_{ij}} = \frac{\sum_{k=1}^{K} \frac{1}{Pr_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(w_t^k) \beta_t^k(j)}{\sum_{k=1}^{K} \frac{1}{Pr_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \qquad \forall \text{ i non-emitting} \tag{V.6}$$

Chapter VI

Implementation and Optimization

Log-odds scores alone do not provide sufficient information to perform the final task of extracting gene name and gene location. Facts usually appear embedded in long sentences, with irrelevant information to either side. Fragments of sentences that receive higher log-odds score fit the model better and thus are more likely both to contain the fact and also to be free of extraneous flanking words. However, if for a given sentence, we rank all possible fragments of at least 5 words by the log-odds score computed as in Equation IV.5, then we often find that the highest scoring fragment contains only either a highly probable gene name or location. So we post-process the ranked fragments in the following manner. Starting with the highest scoring fragment, we compute the most likely path through the fact model using the Viterbi algorithm [16], and then follow this path to collect the words accepted when in the various modules. Fragments for which we fail to find at least one word that is not a common word (see Chapter VII) in the X1, Y1, and verb modules are rejected.

However this technique frequently leaves us with *incomplete* annotations of a very particular flavor. The highest scoring fragment frequently misses important words that are at the *beginning* of the gene name and at the *end* of the gene location. These missed words are always ones that should be accepted by GAPX or GAPY states. This sort of word is assigned the same word emission probability during scoring by both fact and null models: the background probability of the word in the corpus. The only difference between the log-odds scores of two fragments one of which contains an additional GAP word added to the right or left is the log of the ratio of the transition probabilities. The fact model tends to have transition probabilities less than the null model looping probability of $\epsilon = 0.9$, so the fragment with one additional word will usually get a lower score. There is no clear solution to this difficulty, for the HMM does not represent the information necessary to decide between the two alternatives correctly (the value of ϵ is arbitrary). This is to be expected; HMMs are typically tolerant of gaps in the *middle* of a sequence, but not to either side.

We finish filling in the annotations by a procedure that works well in practice. Starting with the first fragment that binds both X1, Y1 and a verb, we consider successively lower ranking fragments that contain the best fragment, looking for any whose most likely path through the model is the same as for the best fragment (for those words they have in common). Any additional X and Y bindings picked out by lower scoring fragment are appended to those from the first fragment.

These two forms of post-processing are likely to apply whenever HMMs are used for information extraction. They make use of no domain knowledge peculiar to the gene name—location task. If one wants to find some binary relation on X and Y a general principle is to discard all fragments that are found to lack information about X, Y, or the relation. And it seems reasonable to try to make use of bindings for fragments that are not the highest scoring when the model parses them in a manner consistent with the optimal fragment.

Some optimization is in order if we intend to score all fragments. It is possible to make this computation tractable if we sensibly recycle forward variable calculations. We find the probability of the first five words in a sentence given the model as

$$Pr(w_1 w_2 ... w_5 | \lambda) = \sum_{i=1}^{N} \alpha_5(i). \tag{VI.1}$$

And instead of discarding $\alpha_5(i)$ once we have used it to compute this probability, we use it again, with Equation V.5 to generate $\alpha_6(i)$ efficiently. Likewise, $\alpha_6(i)$ is just what we require both in order to compute the probability of the first six words given the model, $Pr(w_1w_2...w_6|\lambda)$, and in order to generate $\alpha_7(i)$, efficiently.

But even with such optimizations, it is a good idea to further speed up execution of the end-to-end system by discarding sentences unlikely to contain the fact because they contain few or no words from the model, i.e. sentences we can reasonably deduce would receive low scores without having to actually compute the log-odds score. We prefilter novel sentences by content words occurring in the HMM. Specifically, we first select sentences that contain one of the verbs in the VERB1 module, such as "assigned", "mapped", "isolated", etc. The VERB2 module contains this same set of verbs. Of those candidates that survive this first cut, we select those that contain a noun from the X module ("locus", "homolog", "oncogene", etc). Finally, we filter the remaining sentences, looking for those that have a noun from the Y module ("loci", "region", "band"). In this way, with boolean keyword searches, we discard 83% of the sentences in the corpus without having to score them. It is important to obtain some estimate of the rate of positive examples in the 83% we ignore and compare that to the rate for sentences that pass through the prefilter. Close examination of 1000 sentences chosen at random from the 83% reveals that 21 contained the fact. Conversely, for 692 sentences which passed the filter, and therefore were in fact scored by the HMM (see Chapter VII), 127 were positive examples. So the respective rates of positives are 2.1% and 18.3% for these disjoint sets of examples. The 2.1% positives discarded out of hand are all false negatives, which will be addressed in the next chapter.

Chapter VII

Performance on Training and Novel Sentences

Choosing sentences one-by-one at random from OMIM, we took the first 197 judged actually to express the relationship between a gene and its location. From these sentences we trimmed extraneous material (neither part of the fact nor useful in recognizing it) from beginning and end and replaced words corresponding to one of the five gap categories as described in Chapter III. A first question to answer is how many training examples are truly necessary. From the set of 197 sentence fragments, we set aside 40 at random for testing. Out of the remainder, we randomly chose training sets of size 2, 4, 6, 8, 10, 20, 30, ..., 150. The mean log-odds score of the 40 test sentences is plotted as a function of the size of the training set in Figure VII.1. We repeated this experiment ten times with different sets of 40 test examples. The error bars in the figure depict standard deviation from the mean for these ten trials. From this plot it is clear that more training examples are useful, that the ability of the HMM to statistically favor a test sentence known to contain the fact increases with additional training sentences. It is also clear that we are not overtraining; performance on test sentences does not fall off after some number of examples.

Two features of this learning curve require discussion. First, it rises very

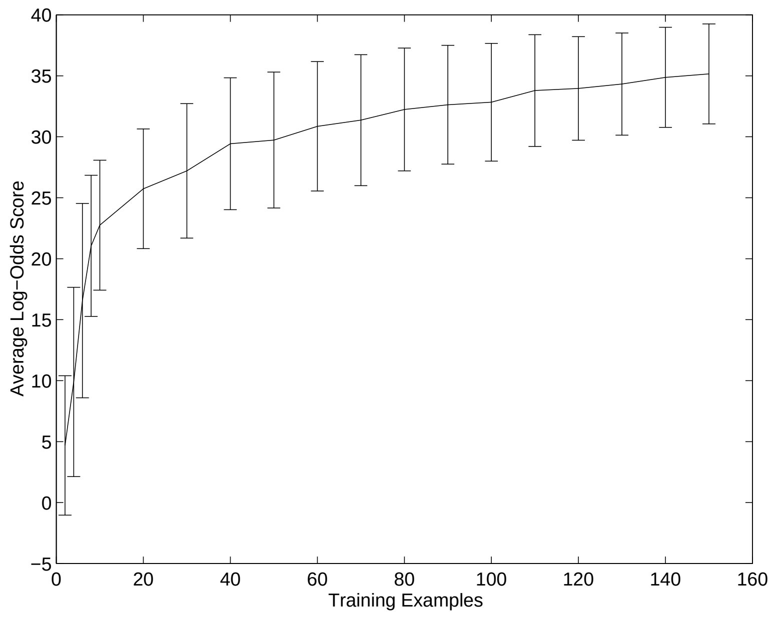


Figure VII.1: Learning Curve for the Gene Name - Gene Location HMM. Mean Log-Odds score for 40 Test Sentences selected at random from 197 training examples. Training sets of various sizes chosen randomly from remaining 157 examples. Error bars represent standard deviation from the mean for ten trials.

steeply in the range 2-10 examples. The mean log-odds score has reached 23 by 10 examples, which is *well* above threshold and about two-thirds its value of 35 for 150 training examples. So it would appear that a very small number of training examples goes a long way toward demonstrating the concept. Second, the curve does not appear to have flattened out after 150 training examples, but instead continues to rise gently. These two effects are related. The log-odds score is the sum of the per-word contributions of emission and transition probabilities (relative to the null model). A training set of only 10 examples contains very little information about word frequencies, i.e. emission probabilities. But for a simple fact, it may very well contain enough examples of local syntax to learn appropriate transition probabilities, *especially* transitions between modules. The steep rise in average log-odds score in the 2-10 training example range of the learning curve is where the model is learning enough syntax to get by. The flip side of this coin serves to explain why the learning curve does not flatten out. In order to learn word frequencies accurately, we need large numbers of examples, and can expect slow progress when this is what we are primarily engaged in learning. As we encounter more examples of words many of which are rare, we accumulate counts that better reflect the semantic preferences of the fact. Deciding on a fair system for evaluating the actual annotations obtained via these automatic means is perilous. The standard performance metrics of information retrieval, *precision* and *recall*, suggest themselves but we must proceed with care.

$$Precision = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \tag{VII.1}$$

$$Recall = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \tag{VII.2}$$

Terms like “True Positive” and “False Positive” require clarification in the context of an information extraction task. In the first place, the conditions of success or failure are not uniquely defined; two different humans assigned the task of grading 50 machine-generated annotations disagreed with one another on 10%. So there may well be a ceiling on the performance we can expect of even the best system. In the second place, we should be open to the possibility of giving partial credit when a small number of forgivable errors appear in an annotation. For instance, if a few unimportant extra words are included in a slot, but no part of the correct annotation is missing or confused, then the grade might reasonably lie somewhere between True and False Positive. Consider the annotation

SLOT	ENTRY
`Gene Name:`	`( provisionally that the Jk locus )`
`Gene Location:`	`( at 18q11-q12 (Geitvik et al 1987) )`
`Verb:`	`( concluded is )`

which is not exactly correct given the sentence

```
 Franks concluded provisionally that the Jk locus is at
18q11-q12 (Geitvik et al 1987).
```

Yet neither is the annotation by any reasonable measure useless. There are merely a few extra words in the gene name, `( provisionally that the ... )`, and in the location, `( ... (Geitvik et al 1987) )`. The pair of non-contiguous verbs, `( concluded is )` in the `Verb` slot is due to the system collecting words in both the VERB1 and VERB2 modules into a single slot. Similarly, if a spurious Y2 binding appears in an otherwise correct annotation, as is the case for the sentence

```
 The COL4A1 and COL4A2 genes are located in a head-to-head
configuration on chromosome 13 and the COL4A3 and COL4A4
genes are similarly arranged on chromosome 2
```

which the system assigns the annotation

SLOT	ENTRY
`Gene Name 1:`	`( The COL4A1 )`
`Gene Name 2:`	`( COL4A2 genes )`
`Gene Location 1:`	`( in a head-to-head configuration on chromosome 13 )`
`Gene Location 2:`	`( the COL4A3 )`
`Verb:`	`( are located )`

then it seems unreasonable to assign *zero* credit, by labeling this annotation a false positive; this is a good annotation and a biologist would have little difficulty tolerating the noise. It seems as thought it would make sense in these situations to give some partial credit.

We therefore computed *Precision* and *Recall* here in two ways, the first is an *unforgiving* accounting because it allows for *no* partial credit. The second accounting is *forgiving* in that it assigns half credit when minor errors occur, in the following way:

True Positive	All annotation slots correct with no content words missing. Extra words tolerated if they are not content words (nouns or verbs)
True/False Positive	No content words that are part of the fact are missing from any slot. Extra words tolerated even if they are content words. Spurious annotations fall under this category as well.
False Positive	The fact either is not contained in the sentence, or the system parsed it incorrectly, i.e. into wrong slots.
True Negative	The sentence does not contain fact and system discarded it correctly.
False Negative	The sentence contained the fact and system incorrectly discarded it.

The performance of our system on 692 novel sentences, for both the unforgiving and forgiving accounting schemes, as well as for various threshold values is depicted in the Precision-Recall plot of Figure VII.2. For this particular information extraction task, precision is more important than recall. There are two reasons for this. In the first place, the corpus tends to contain multiple statements of the same fact; these are not corrections or revisions, but complete restatements in novel syntactic forms. If we discard a sentence containing a fact it is likely we will have another chance at extracting it as it will be reiterated elsewhere. Second, if this is to be an entirely automated system for generating a database, then we might well prefer entirely to miss some facts than to fill up our database with a significant proportion of incorrect ones. Experimentally, it appears that a threshold of 30 might be a good choice, for even using the unforgiving accounting this gives us about 74% good annotations while only discarding about 64%. We can justify this setting with a second look at the learning curve of Figure VII.1; the HMM trained with all of the examples

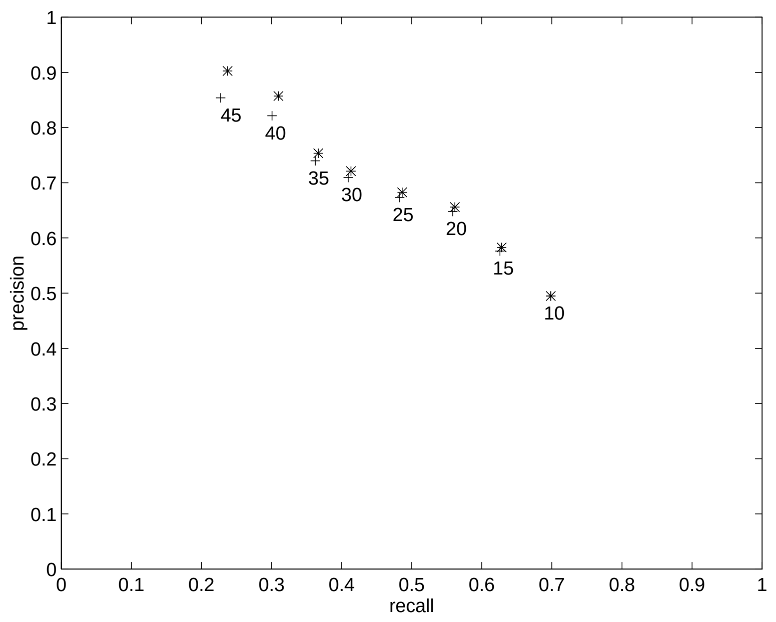


Figure VII.2: How Precision and Recall vary with choice of threshold θ. As θ (the number next to each of the crosses and stars in the plot) decreases, Precision decreases and Recall increases. The crosses are for the *unforgiving* accounting scheme and the stars are for the *forgiving* accounting.

assigns log-odds score of about 35 +/- 4 to sentences in the training set.

In fact, this 36% recall rate is optimistic. It tells us that the system correctly classified and processed 36% of the positives contained in the 692 examples that passed the prefilter described in Chapter VI. Approximately 4050 sentences were considered by the prefilter in order to accumulate 692, which means that about 3358 were discarded without scoring. If 2% of those 3358 were positives as in the last Chapter, that means the number of false negatives should, for each recall point in Figure VII.2 be higher by about 70. The number of false negatives for the 692 examples varies between 36, for a threshold of $\theta = 10$, to 54, for a threshold of $\theta = 45$. The recall rates with respect to the entire corpus, therefore are lower. They would range between 47% and 23%, instead of between 72% and 49%.

Chapter VIII

Concluding Remarks on Prior Knowledge and Future Work

This research demonstrates that an HMM can be constructed and tuned to perform an information extraction task with impressive precision and recall. Our HMM is composed of modules whose internal transition probabilities reflect local syntax and are learned entirely from training data. The prior knowledge afforded our system consists of the following.

1. Training Examples are pruned to remove extraneous words to the right or left of the fact.

2. Uncommon words (not explicitly listed in any state as valid words in the model) in training sentence fragments are abstracted into fixed token words.

3. The number of states and the valid words for each state are specified.

4. States are grouped into modules corresponding roughly to high-level syntactic entities such as noun groups, verb groups and prepositional phrases.

5. Between-module transition probabilities are initialized before training our HMM. That is, the valid set of transitions out of a module is restricted, but the relative magnitudes of their probabilities are learned.

An important question to answer next if this sort of stochastic approach is to be practical is to what extent can we do without, infer, or easily gather this prior knowledge. There is evidence that suggests the first body of prior knowledge listed above, the trimming of training sentences, might be unnecessary. HMMs designed to characterize proteins by family make use of "free-insertion" modules that permit training on examples that include irrelevant information to either side of the pertinent section [12, 11, 1]. Tokenizing of uncommon words (item 2) might occur automatically according to their frequency in the corpus, but the more specific information (coded here into the token itself) of what module an uncommon word in a training fragment belongs to would be more difficult to deduce. Additionally, the necessary grouping of words into states (item 3 above) might be achieved in an automatic fashion via word clustering; in practice, the set of allowed words in a state all take roughly the same part-of-speech and are similar or related in meaning.

The remainder of the prior knowledge, in particular the details of which states are to be collected into modules and valid between-module transitions, must be specified explicitly in part at least, for it in many cases depends upon the particular extraction task. Given the same set of example sentences, we might undertake a number of different extraction tasks and each of these might require different modularity. But while we cannot infer this information, we can conceive of a dialog between machine and expert by which it might be obtained without too much effort. The expert would highlight for a small number of examples the groups of words which correspond to the slots in the annotation. Training an HMM to extract information might take place within a relevance feedback loop. With each iteration, the program would return a set of probable annotations and source sentences. The user would be asked to make a small number of corrections and judgements. This information would both guide repairs to HMM architecture and add examples to the training set. The learning curve in Figure VII.1 suggests that a small number of such iterations might suffice to achieve good performance.

Chapter IX

Acknowledgements

This research was supported by the Computational Biology Group of the San Diego Supercomputer Center, under NSF grant 9630339.

Bibliography

[1] C. Barrett, R. Hughey, and K. Karplus. Scoring hidden Markov models. In *Proceedings, 4th International Conference on Intelligent Systems for Molecular Biology, St. Louis, MO, poster only.* AAAI, June 1996.

[2] L. E. Baum and J. A. Egon. An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology. *Bulletin of the American Meteorological Society*, 73:360–363, 1967.

[3] D. Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name finder. In *Proceedings, Fifth Conference on Applied Natural Language Processing.* Association for Computational Linguistics, April 1997.

[4] E. Brill. Some advances in transformation-based part of speech tagging. In *Proceeding of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 1, pages 722–727. MIT Press, Cambridge, MA, 31 July-4 Aug 1994.

[5] Peter Brown, John Cocke, Stephen Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, June 1990.

[6] Peter Brown, Vincent Della Pietra, Peter V. deSouza, and Junifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[7] J. R. Hobbs. The generic information extraction system. In *Proceedings of the Fifth Message Understanding Conference (MUC-5).* Morgan Kaufmann, San Mateo, CA, 1993.

[8] J. R. Hobbs, D. E. Appelt, J. S. Bear, D. J. Israel, and W. Mabry Tyson. Fastus: A system for extracting information from natural-language text. Technical Report 519, SRI International, November 1992.

[9] S. B. Huffman. Learning information extraction patterns from examples. In S. Wermter, G. Scheler, and E. Riloff, editors, *Connectionist, Statistical and*

Symbolic Approaches to Learning for Natural Language Processing, pages 246–260. Springer-Verlag, 1996.

[10] R. Hughey and A. Krogh. SAM: Sequence alignment and modeling software system. Technical Report UCSC-CRL-95-7, University of California, Santa Cruz, 1995.

[11] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, (12(2)):95–107, 1996.

[12] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, (235):1501–1531, February 1994.

[13] J. Lafferty, D. Sleator, and D. Temperley. Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language, Cambridge MA*, 1992.

[14] W. Lehnert, J. McCarthy, S. Soderland, E. Riloff, C. Cardie, J. Peterson, F. Feng, C. Dolan, and S. Goldman. UMASS/HUGHES: Description of the CIRCUS system used for MUC-5. In *Proceedings of the Fifth Message Understanding Conference (MUC-5)*. Morgan Kaufmann, San Mateo, CA, 1993.

[15] Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of English words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics, Ohio State University, Columbus, Ohio*, pages 183–190. Association for Computational Linguistics, June 1993.

[16] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.

[17] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93 and IAAI-93, Washington D. C., USA)*, pages 811–816. AAAI Press, July 1993.

[18] S. Soderland and W. Lehnert. Wrap-Up: A trainable discourse module for information extraction. *Journal of Artificial Intelligence Research*, (2):131–158, 1994.