7

Part-of-Speech Tagging Using Stochastic Techniques

7.1 The Noisy Channel Model

7.1.1 Presentation

Like transformation-based tagging, statistical (or stochastic) part-of-speech tagging assumes that each word is known and has a finite set of possible tags. These tags can be drawn from a dictionary or a morphological analysis. When a word has more than one possible tag, statistical methods enable us to determine the optimal sequence of part-of-speech tags $T = t_1, t_2, t_3, ..., t_n$, given a sequence of words $W = w_1, w_2, w_3, ..., w_n$.

Optimal part-of-speech sequence refers to Shannon's (1948) noisy channel model, where a sequence of symbols is transmitted over a noisy channel and received under the form of a sequence of signals. Here, we suppose that part-of-speech tags are transmitted and come out under the form of words:

$$t_1, t_2, t_3, ..., t_n \rightarrow \text{noisy channel} \rightarrow w_1, w_2, w_3, ..., w_n.$$

The optimal part-of-speech sequence knowing the word sequence corresponds to the maximization of the conditional probability:

$$\hat{T} = P(t_1, t_2, t_3, ..., t_n | w_1, w_2, w_3, ..., w_n).$$

Bayes' theorem on conditional probabilities of events A and B states that:

$$P(A|B)P(B) = P(B|A)P(A).$$

We denote $P(W) = P(w_1, w_2, w_3, ..., w_n)$ and $P(T) = P(t_1, t_2, t_3, ..., t_n)$. Using Bayes' theorem, the most probable estimate of the part-of-speech sequence is given by:

$$\hat{T} = \arg\max \frac{P(T)P(W|T)}{P(W)}.$$

For a given word sequence, $w_1, w_2, w_3, ..., w_n$, $P(W)$ is constant and we can leave it out. We can rewrite the formula as:

$$\hat{T} = \arg\max P(T)P(W|T).$$

7.1.2 The N-gram Approximation

Statistics on sequences of any length are impossible to obtain, and at this point we need to make some approximations on $P(T)$ and $P(W|T)$ to make the estimation tractable. A product of trigrams usually approximates the complete part-of-speech sequence:

$$P(T) = P(t_1, t_2, t_3, ..., t_n) \approx P(t_1)P(t_2|t_1) \prod_{i=3}^{n} P(t_i|t_{i-2}, t_{i-1}).$$

If we use a start-of-sentence delimiter `<s>`, the two first terms of the product, $P(t_1)P(t_2|t_1)$, are rewritten as $P(< s >)P(t_1| < s >)P(t_2| < s >, t_1)$, where $P(< s >) = 1$.

We estimate the probabilities with the maximum likelihood, P_{MLE}:

$$P_{MLE}(t_i|t_{i-2}, t_{i-1}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})}.$$

Probabilities on trigrams $P(t_i|t_{i-2}, t_{i-1})$ require an estimate for any sequence of three parts-of-speech tags. This is obtained from hand-annotated corpora. If N_p is the number of the different parts-of-speech tags, there are $N_p \times N_p \times N_p$ values to estimate. Most of the time, annotated data is not sufficient and some sequences are missing. Few corpora are likely to contain a reliable number of the article–article–article sequence, for instance. We already encountered this problem of sparse data in Chap. 4. We can solve it using a back-off strategy or a linear interpolation.

If data are missing, we can back-off to bigrams:

$$P(T) = P(t_1, t_2, t_3, ..., t_n) \approx P(t_1) \prod_{i=2}^{n} P(t_i|t_{i-1}).$$

We can further approximate the part-of-speech sequence as the product of part-of-speech probabilities:

$$P(T) = P(t_1, t_2, t_3, ..., t_n) \approx \prod_{i=1}^{n} P(t_i).$$

And finally, we can combine linearly these approximations:

$$P_{LinearInter}(t_i|t_{i-2}t_{i-1}) = \lambda_1 P(t_i|t_{i-2}t_{i-1}) + \lambda_2 P(t_i|t_{i-1}) + \lambda_3 P(t_i),$$

with $\lambda_1 + \lambda_2 + \lambda_3 = 1$, for example, $\lambda_1 = 0.6$, $\lambda_2 = 0.3$, $\lambda_3 = 0.1$.

Using the maximum likelihood estimate, this yields:

$$P_{LinearInter}(t_i|t_{i-2}t_{i-1}) = \lambda_1 \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})} + \lambda_2 \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} + \lambda_3 \frac{C(t_i)}{N},$$

where N is the count of words in the corpus.

We can obtain optimal λ values by using a development set: a part of the hand-annotated corpus distinct from the training set and the test set dedicated to the fine-tuning of parameters. After learning the probabilities from the training set, we will run the part-of-speech (POS) tagger on the development set. We will vary the λ values until we find the triplet that yields the best accuracy. We will finally apply the POS tagger to the test set to know its real accuracy.

The complete word sequence knowing the part-of-speech sequence is usually approximated as:

$$P(W|T) = P(w_1, w_2, w_3, ..., w_n|t_1, t_2, t_3, ..., t_n) \approx \prod_{i=1}^{n} P(w_i|t_i).$$

Like the previous probabilities, $P(w_i|t_i)$ is estimated from hand-annotated corpora using the maximum likelihood:

$$P_{MLE}(w_i|t_i) = \frac{C(w_i, t_i)}{C(t_i)}.$$

For N_w different words, there are $N_p \times N_w$ values to obtain. But in this case, many of the estimates will be 0.

7.1.3 Tagging a Sentence

We will now give an example of sentence tagging in French with *Je le donne* 'I give it'. Word *Je* is an unambiguous pronoun. Word *le* is either an article or a pronoun, and *donne* can be a noun (*deal*) or a verb (*donner*). Probabilistic tagging consists in finding the optimal path from the four possible in Fig. 7.1.

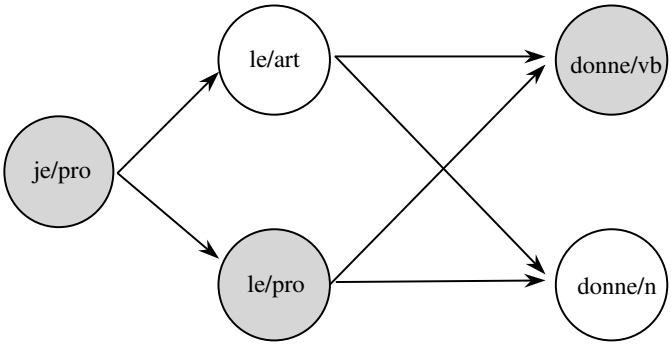

Fig. 7.1. Possible sequences of part-of-speech tags, where *pro* denotes a pronoun, *art* an article, *n* a noun, and *vb* a verb.

Using the formulas given before, we associate each transition with a probability product: $P(w_i|t_i) \times P(t_i|t_{i-2}, t_{i-1})$. We compute the estimate of part-of-speech

sequences along the four paths by multiplying the probabilities. The optimal tagging corresponds to the maximum of these four values:

1. $P(pro| < s >) \times P(art| < s >, pro) \times P(verb|pro, art) \times P(je|pro) \times P(le|art) \times P(donne|verb)$
2. $P(pro| < s >) \times P(art| < s >, pro) \times P(noun|pro, art) \times P(je|pro) \times P(le|art) \times P(donne|noun)$
3. $P(pro| < s >) \times P(pro| < s >, pro) \times P(verb|pro, pro) \times P(je|pro) \times P(le|pro) \times P(donne|verb)$
4. $P(pro| < s >) \times P(pro| < s >, pro) \times P(noun|pro, pro) \times P(je|pro) \times P(le|pro) \times P(donne|noun)$

This method is very simple. However, it is very costly for long sequences. The computation with a sentence of N words and a tagset of T tags will have an upper bound complexity of N^T, which means it is exponential.

7.1.4 The Viterbi Algorithm: An Intuitive Presentation

Using the noisy channel model as we described it is not efficient in terms of speed and memory. This is because the algorithm has to maintain nonoptimal paths for all the intermediate nodes in the automaton. The Viterbi algorithm is a common way to optimize the search.

In the naïve implementation, we traversed all the paths and we computed the most probable POS sequence at the final node of the automaton, i.e., at the final word of the sentence. The Viterbi algorithm determines the optimal subpaths for each node in the automaton while it traverses the automaton and discards the others. We shall extend the example of the previous section to

Je le donne demain dans la matinée.
'I give it tomorrow morning.'

and let us consider bigrams instead of trigrams to simplify the presentation.

Figure 7.2 shows the possible POS tags and the number of possible paths, which is $1 \times 2 \times 2 \times 1 \times 1 \times 2 \times 1 = 8$. Let us traverse the automaton from *Je* to *dans*.

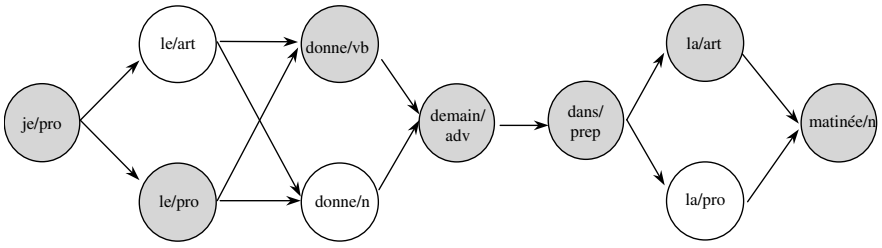

Fig. 7.2. The search space, where *adv* denotes an adverb and *prep* a preposition; the other tags are as given in Fig. 7.1.

The words *demain* and *dans* are not ambiguous, and we saw in the last section that there are four possible paths at this point. Up to *demain*, the most likely sequence will correspond to the most probable path out of the four we saw before:

1. $P(pro| < s >) \times P(art|pro) \times P(verb|art) \times P(adv|verb)$
   $P(je|pro) \times P(le|art) \times P(donne|verb) \times P(demain|adv)$
2. $P(pro| < s >) \times P(art|pro) \times P(noun|art) \times P(adv|noun)$
   $P(je|pro) \times P(le|art) \times P(donne|noun) \times P(demain|adv)$
3. $P(pro| < s >) \times P(pro|pro) \times P(verb|pro) \times P(adv|verb)$
   $P(je|pro) \times P(le|pro) \times P(donne|verb) \times P(demain|adv)$
4. $P(pro| < s >) \times P(pro|pro) \times P(noun|pro) \times P(adv|noun)$
   $P(je|pro) \times P(le|pro) \times P(donne|noun) \times P(demain|adv)$

Demain has still the memory of the ambiguity of *donne*: $P(adv|verb)$ and $P(adv|noun)$. This is no longer the case with *dans*. According to the noisy channel model and the bigram assumption, the term brought by the word *dans* is $P(dans|prep) \times P(prep|adv)$. It does not show the ambiguity of *le* and *donne*. The subsequent terms will ignore it as well.

This means that the optimal POS tag sequence of words before *dans* is already determined even if we have not yet reached the end of the sentence. It corresponds to the highest value of the four paths. It is then sufficient to keep it with the corresponding path. We can forget the others. This is the idea of the Viterbi optimization. We will describe the algorithm rigorously in the next section.

7.2 Markov Models

When we tagged words with a stochastic technique, we assumed that the current word's part of speech depended only on a couple of words before it. This limited history is a frequent property of many linguistic phenomena. It has been studied extensively since the end of the 19th century, starting with Andrei Markov. Markov processes form the theoretical background to stochastic tagging and can be applied to many problems. We introduce them now.

7.2.1 Markov Chains

A Markov chain or process is a sequence $\{X_1, X_2, ..., X_T\}$ where X_t denotes a random variable at time t. Variables have their values in a finite set of states $\{q_1, ..., q_N\}$ called the state space. Following Rabiner (1989), processes are Markovian if they have the following properties:

- A limited history. The current state depends only on a constant number of previous states: one in first-order processes, $P(X_t = q_j|X_1, ..., X_{t-1}) = P(X_t = q_j|X_{t-1})$, and two in second-order processes $P(X_t = q_j|X_1, ..., X_{t-1}) = P(X_t = q_j|X_{t-2}, X_{t-1})$.

- Independent of time t. For first-order processes, this means that they can be represented as a transition matrix with coefficients $P(X_t = q_j | X_{t-1} = q_i) = a_{ij}$, $1 \leq i, j \leq N$, with ordinary probability constraints $\sum_{j=1}^{N} a_{ij} = 1$, and $a_{ij} \geq 0$.

Markov chains define random transitions from one state to another one. We can represent them as **probabilistic** or **weighted automata**. We just need to augment transitions of automata we used in Chap. 2 with a probability. Unlike ordinary automata, the initial state can be any state in the set and will be modeled by a probability at time 1. The probability of initial states is $\pi_i = P(X_1 = q_i)$, with $\sum_{i=1}^{N} \pi_i = 1$.

In the case of natural language processing, "time sequence" is not the most relevant term to describe the chain. More appropriately, the sequence corresponds to the word flow from left to right and t to the word position in the sequence. It is easy then to see that first-order processes reflect part-of-speech bigrams, while second-order processes correspond to trigrams. Figure 7.3 shows partial bigram probabilities using a Markov chain (numbers are fictitious and transitions are not complete). For part-of-speech tagging, a_{ij} coefficients correspond to probabilities of part-of-speech bigrams computed over the tagset.

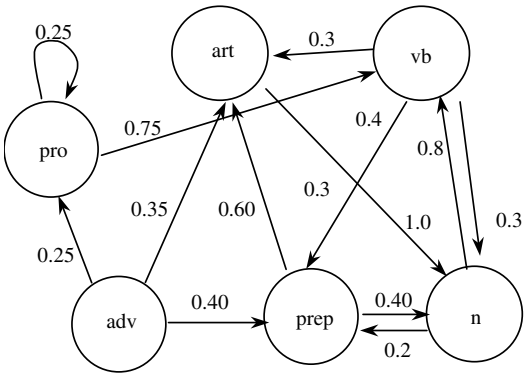

Fig. 7.3. A Markov chain representing bigram probabilities as part-of-speech transitions (numbers are fictitious, and transitions are not complete).

Instead of using an automaton, we can represent a Markov process as a trellis where states are a function of the time (the word's positions, here). In part-of-speech tagging, the vertical axis corresponds to the different part-of-speech values (the states) and the horizontal axis corresponds to the part-of-speech sequence (Fig. 7.4). All the possible bigram combinations are represented as arrows from states at time $t-1$ to states at time t. T is the sentence length.

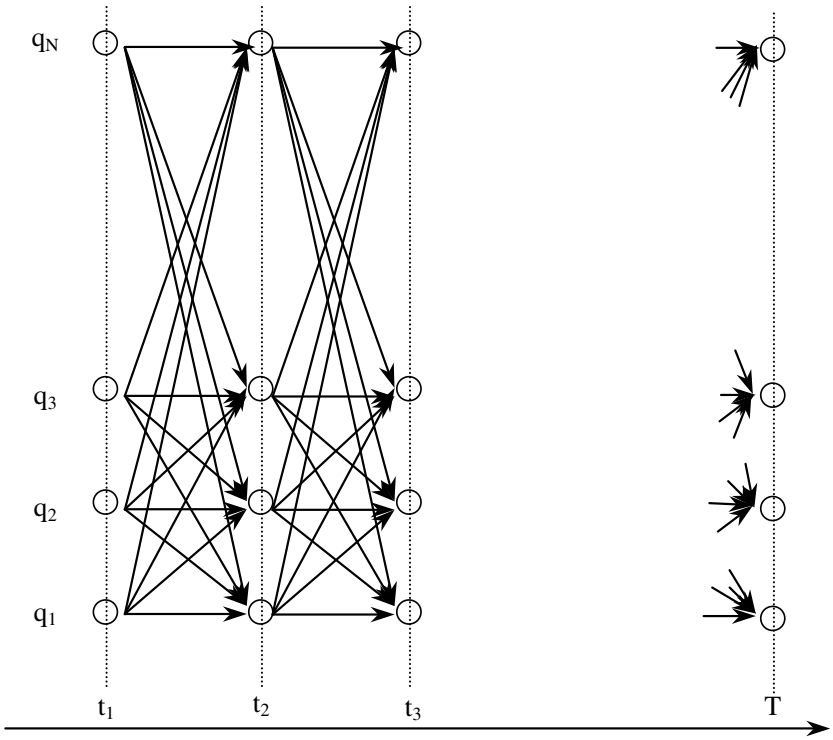

Fig. 7.4. Trellis that represents the states as the vertical axis and the time as the horizontal axis. The states correspond to part-of-speech values, and the discrete time values are the indices in the part-of-speech sequence. T is the sentence length.

7.2.2 Hidden Markov Models

Markov chains provide a model to the part-of-speech sequence. However, this sequence is not directly accessible since we usually only have the word sequence. Hidden Markov models (HMM) are an extension to the Markov chains that make it possible to include the words under the form of observed symbols. Each state of an HMM emits a symbol taken from an output set along with an emission probability. HMMs are then a stochastic representation of an observable output generated by a hidden sequence of states. They enable us to compute the probability of a state sequence (the parts of speech) given an output or observation sequence (the words).

We saw that part-of-speech tagging uses a stochastic formula that comprises two terms: $P(T)$ and $P(W|T)$. The first one, $P(T)$, corresponds to a Markov chain where transition probabilities between states represent the part-of-speech bigrams. The second term, $P(W|T)$, is an HMM superimposed on the chain. It augments each state with the capacity to emit a word using a probability function $P(w_i|t_i)$ that measures the association between the parts of speech and the words (Fig. 7.5). Although a state – a part of speech – can emit any word in the model, most probabilities

will be 0 in reality. This is because words have a finite number of possible parts of speech, most of the time, as we saw, only one or two.

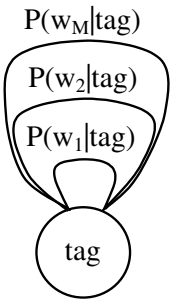

Fig. 7.5. Each state in the trellis is augmented with word emission probabilities.

The formal definition of HMMs is based on the Markov chains where we add the emission properties. Table 7.1 shows the notation and its application in part-of-speech tagging.

Table 7.1. The hidden Markov model notation and its application to part-of-speech (POS) tagging.

HMM notation	Application to POS tagging
$S = \{q_1, q_2, q_3, ..., q_N\}$ is a finite set of states.	The set of parts of speech.
$V = \{v_1, v_2, v_3, ..., v_M\}$ is an output alphabet: a finite set of symbols.	The set of words, the vocabulary.
$O = \{o_1, o_2, o_3, ..., o_T\}$ is the output or observation sequence, with $o_i \in V$ obtained from a sequence of states.	Each part of speech emits one word taken in the vocabulary. This is what we observe.
$A = \{a_{ij}\}$ is a state transition matrix.	The bigram probabilities $P(t_k = q_j \| t_{k-1} = q_i)$.
$B = \{b_j(v_k)\}$ are the emission probabilities of symbol v_k in state j.	The conditional probability to observe a word given a part of speech $P(w\|t)$.
$\Pi = \{\pi_i\}$ are the initial state probabilities.	The probability of the first part of speech.

7.2.3 Three Fundamental Algorithms to Solve Problems with HMMs

Hidden Markov models are able represent associations between word and parts-of-speech sequences. However, they do not tell how to solve the annotation problem. We need complementary algorithms for them to be useful. More generally, problems to solve fall into three categories that correspond to three fundamental algorithms (Rabiner 1989):

- Estimate the probability of an observed sequence. This corresponds to the sum of all the paths producing the observation. It is solved using the forward procedure. In the specific case of POS tagging, it will determine the probability of the word sequence. Although the forward procedure is not of primary importance here, it is fundamental and has many other applications.
- Determine the most likely path of an observed sequence. This is a decoding problem that is solved using the Viterbi algorithm.
- Determine (learn) the parameters given a set of observations. This algorithm is used to build models when we do not know the parameters. It is solved using the forward–backward algorithm.

We now present the algorithms where we follow Rabiner's notation (1989).

7.2.4 The Forward Procedure

The first problem to solve is to compute the probability of an observation sequence $O = \{o_1, o_2, o_3, ..., o_T\}$, given an HMM model $\lambda = (A, B, \pi)$.

If we consider only one specific sequence of states $Q = \{s_1, s_2, s_3, ..., s_T\}$, with $s_i \in S$, we compute it with the observation probability given the state sequence

$$\begin{aligned} P(O|Q, \lambda) &= \prod_{t=1}^{T} P(o_t|s_t, \lambda), \\ &= b_{s_1}(o_1)b_{s_2}(o_2)b_{s_3}(o_3)...b_{s_T}(o_T). \end{aligned}$$

multiplied by the state sequence probability

$$\begin{aligned} P(Q|\lambda) &= \pi_{s_1} \prod_{t=2}^{T} P(s_t|s_{t-1}), \\ &= \pi_{s_1} a_{s_1 s_2} a_{s_2 s_3} ... a_{s_{T-1} s_T}. \end{aligned}$$

In HMMs, any sequence of state can produce the observation. This means that the observation probability is the sum of observation probabilities for all the possible state combinations:

$$\begin{aligned} P(O|\lambda) &= \sum_{\text{All } Q} P(O|Q, \lambda)P(Q|\lambda), \\ &= \sum_{\text{All } s_1, s_2, ..., s_T} \pi_{s_1} b_{s_1}(o_1) a_{s_1 s_2} b_{s_2}(o_2) a_{s_2 s_3} b_{s_3}(o_3) ... a_{s_{T-1} s_T} b(o_T). \end{aligned}$$

This method, however, is intractable for long sequences because of its complexity, N^T.

The forward procedure simplifies the brute-force method by factoring all paths incoming into a state at time t. This means that at each instant of time of the observation sequence, we maintain exactly N paths: the number of different states.

Let us denote $\alpha_t(j)$ the probability of an observation $o_1, o_2, o_3, ..., o_t$, with the condition that we are in state q_j at time t: $P(o_1, o_2, o_3, ..., o_t, s_t = q_j|\lambda)$. We compute $\alpha_{t+1}(i)$ by induction with transitions from all states at time t to state i at time $t + 1$. Figure 7.6 shows how α_t values are summed to obtain $\alpha_{t+1}(i)$.

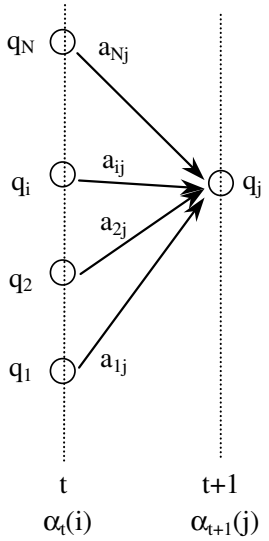

Fig. 7.6. Transitions from states $q_1, q_2, q_3, ..., q_N$ at time t to state q_j at time $t+1$.

We can compute an observation probability with a matrix reproducing the structure of the trellis in Fig. 7.4. The algorithm iteratively fills the trellis' columns from left to right. Each column is an array of length N corresponding to the number of states where we store the probabilities of the observation so far. The element of index i in the tth column contains the $\alpha(i)$ value at time t.

The first step of the algorithm fills the first column with the initial probabilities. The induction loop updates the values from t to $t+1$ by summing all the incoming transitions for each element in the $(t+1)$th column from the tth column (Table 7.2). Finally, we obtain the observation probability by summing all the elements of the last column in the matrix. The complexity of this algorithm is $O(N^2T)$.

Table 7.2. The forward procedure.

Steps	**Operations**
1. Initialization	$\alpha_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N$
2. Induction	$\alpha_{t+1}(j) = b_j(o_{t+1}) \times \sum_{i=1}^{N} \alpha_t(i) a_{ij}, 1 \leq j \leq N$, and $1 \leq i \leq T-1$
3. Termination	$P(O\|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

7.2.5 Viterbi Algorithm

The Viterbi algorithm is an efficient method to find the optimal sequence of states given an observation. As with the forward procedure, it iterates from $t = 1$ to $t = T$ and searches the optimal path leading to each state in the trellis at time t.

Let us denote $\delta_t(j)$ the maximal probability of an observation $o_1, o_2, o_3, ..., o_t$ with the condition that we are in state q_j at time t:

$$\max_{s_1,s_2,...,s_{t-1}} P(s_1, s_2, ..., s_{t-1}, o_1, o_2, o_3, ..., o_t, s_t = q_j|\lambda),$$

and $\psi_t(j)$ the corresponding optimal path.

The Viterbi algorithm resembles the forward procedure. It moves from left to right iteratively to fill the columns in the trellis. Each column element contains the most probable path, $\psi(j)$, to reach this element and its probability $\delta(j)$. In fact, $\psi(j)$ just needs to store the preceding state in the optimal path.

The first step of the algorithm fills the first column with the initial probabilities. The induction loop updates the values from t to $t + 1$ by taking the maximum of all the incoming transitions for each element in the $(t + 1)$th column and the node that led to it. Finally, we determine the most probable path from the maximum of all the elements of the last column in the matrix. We backtrack in the matrix to find the state sequence that led to it (Table 7.3).

Table 7.3. The Viterbi algorithm.

Steps	Operations
1. Initialization	$\delta_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N$
	$\psi_1(i) = null$
2. Induction	$\delta_{t+1}(j) = b_j(o_{t+1}) \times \max_{1\leq i\leq N} \delta_t(i)a_{ij}, 1 \leq j \leq N$, and $1 \leq i \leq T - 1$
	$\psi_{t+1}(j) = \arg\max_{1\leq i\leq N} \delta_t(i)a_{ij}$
3. Termination	$P* = \max_{1\leq i\leq N} \delta_T(i)$
	$s_T* = \arg\max_{1\leq i\leq N} \delta_T(i)$
	The optimal path sequence is given by the backtracking: $s^*_T, s^*_{T-1} = \psi_T(s^*_T), s^*_{T-2} = \psi_{T-2}(s^*_{T-1}), ...$

The Viterbi algorithm is a dynamic programming technique comparable to the computation of the min-edit distance. Its implementation also uses a table. Table 7.4 shows how to fill the three first columns with the sentence <s> *Je le donne demain dans la matinée*.

We start the sentence with $\delta_1(< s >) = 1.0$ and $\delta_1(i) = 0$ for the rest of indices $i \neq < s >$. This means that in the first column, all the cells equal 0, except for one. The computation of the second column is easy. Each cell i is filled with the term $P(i| < s >) \times P(Je|i)$, with $i \in \{prep, adverb, pronoun, verb, noun, art, <$

$s >\}$. The algorithm really starts with the third column. For each cell j, we compute $\max_i P(j|i) \times P(le|j) \times \delta_2(i)$. The *pronoun* cell, for instance, is filled with $\max_i P(pronoun|i) \times P(le|pronoun) \times \delta_2(i)$. This process is iterated for each column to the end of the matrix.

Table 7.4. The Viterbi algorithm applied to the sentence <s> *Je le donne demain dans la matinée*.

$i\backslash\delta$	δ_1	δ_2	δ_3	δ_4	δ_5	δ_6	δ_7	δ_8
prep	0							
adverb	0							
pronoun	0							
verb	0							
noun	0							
art	0							
<s>	1.0	0	0	0	0	0	0	0
	<s>	*Je*	*le*	*donne*	*demain*	*dans*	*la*	*matinée*

7.2.6 The Backward Procedure

We have computed the estimation of an observation from left to right. Although less natural, we can also compute it from right to left. We now present this backward procedure to introduce the forward–backward algorithm in the next section.

The backward variable $\beta_t(j) = P(o_{t+1}, o_{t+2}, o_{t+3}, ..., o_T, s_t = q_j|\lambda)$ is the probability of an observation $o_{t+1}, o_{t+2}, o_{t+3}, ..., o_T$ with the condition that we are in state q_j at time t. We compute $\beta_t(i)$ by induction with transitions from state i at time t to all states at time $t+1$. Figure 7.7 shows how $\beta_{t+1}(i)$ values are summed to obtain β_t, and Table 7.5 shows the procedure.

Table 7.5. The backward procedure.

Steps	**Operations**
1. Initialization	$\beta_T(i) = 1, 1 \leq i \leq N$
2. Induction	$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(i), 1 \leq j \leq N$, and for $t = T-1$ to $t = 1$.
3. Termination	$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)$

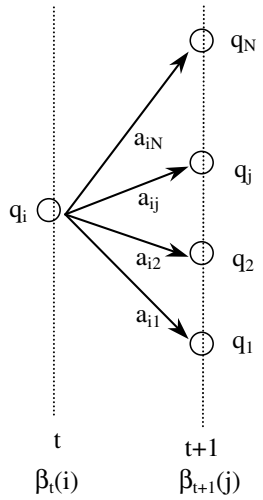

Fig. 7.7. Transitions from state q_i at time t to states $q_1, q_2, q_3, ..., q_N$ at time $t + 1$.

7.2.7 The Forward–Backward Algorithm

The forward–backward algorithm will enable us to derive the a_{ij} and $b_j(o_t)$ coefficients, here $P(w_i|t_i)$ and $P(t_i|t_{i-1})$, from raw, unannotated texts. Although this yields results inferior to those obtained from a hand-annotated corpus, it makes it possible to build a part-of-speech tagger when no annotation is available.

The forward–backward algorithm is referred to as an **unsupervised learning** method, because no additional information is available except the text. This is opposed to **supervised learning**, when the algorithm has access to some sort of reference annotation.

Informal Presentation. The idea of the forward–backward algorithm is to guess initial estimates to $P(t_i|t_{i-1})$ and $P(w_i|t_i)$ and tag the corpus. Once we have a tagged corpus, we can derive new estimates of $P(w_i|t_i)$ and $P(t_i|t_{i-1})$ that we will use to retag the corpus. We repeat the process until it converges (Table 7.6).

However, we have no guarantee that the algorithm converges, and when it converges, we can also hit a local maximum. In the latter case, the learning procedure will stop without finding correct figures. This is the drawback of this method. For this reason some quantity of hand-annotated data is always preferrable to a raw corpus (Merialdo 1994).

The Algorithm. In the presentation above, we had to tag the text before we could derive new estimates of probabilities $P(t_i|t_{i-1})$ and $P(w_i|t_i)$, or more generally a_{ij} and $b_j(o_t)$. In fact, we can avoid the tagging stage. The coefficients can be computed directly using the forward procedure. We will reestimate $\hat{a}_{ij}$ at step n of the estimation process from estimates a_{ij} at step $n - 1$.

Table 7.6. Iterative estimation of $P(t_i|t_{i-1})$ (figures are fictitious).

Steps	**Estimates used to tag the corpus**	**Estimates derived from the tagged corpus**
Initial estimates	$P(pronoun\|pronoun) = 0.2$ $P(art\|pronoun) = 0.2$ $P(verb\|pronoun) = 0.6$	
We tag the corpus and we derive new estimates.		$P(pronoun\|pronoun) = 0.15$ $P(art\|pronoun) = 0.05$ $P(verb\|pronoun) = 0.8$
Second estimates	$P(pronoun\|pronoun) = 0.15$ $P(art\|pronoun) = 0.05$ $P(verb\|pronoun) = 0.8$	
We retag the corpus and we derive estimates.		$P(pronoun\|pronoun) = 0.18$ $P(art\|pronoun) = 0.02$ $P(verb\|pronoun) = 0.9$
Third estimates	$P(pronoun\|pronoun) = 0.18$ $P(art\|pronoun) = 0.02$ $P(verb\|pronoun) = 0.9$	

The algorithm idea is to consider one observation – one word – and then to average it on all the other observations – the whole sentence. For one specific observation $b_j(o_{t+1})$ at time $t+1$, corresponding here to the word of index $t+1$, the transition probability from state $s_t = q_i$ to state $s_{t+1} = q_j$ corresponds to

$$\begin{aligned}\xi_t(i,j) &= P(s_t = q_i, s_{t+1} = q_j|O,\lambda),\\ &= \frac{P(s_t=q_i,s_{t+1}=q_j,O|\lambda)}{P(O|\lambda)},\\ &= \frac{P(s_t=q_i,s_{t+1}=q_j,O|\lambda)}{\sum\limits_{1\le i\le N}\sum\limits_{1\le j\le N} P(s_t=q_i,s_{t+1}=q_j,O|\lambda)}.\end{aligned}$$

We can use the forward and backward probilities to determine the estimate. Figure 7.8 shows how to introduce them in the equation.

We have:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum\limits_{i=1}^{N}\sum\limits_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}.$$

We denote $\gamma_t(i) = \sum\limits_{j=1}^{N}\xi_t(i,j)$ the probability to be in state q_i at time t.

To consider all the observations, we sum $\xi_t(i,j)$ from $t=1$ to $t=T-1$. The expected number of transitions from state q_i to state q_j is $\sum\limits_{t=1}^{T-1}\xi_t(i,j)$, and the expected

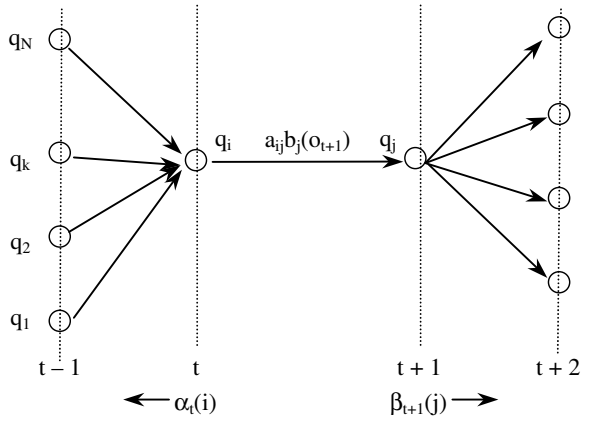

Fig. 7.8. Transition from state q_i at time t to state q_j at time $t+1$ with observation o_{t+1}. After Rabiner (1989).

number of transitions from state q_i is $\sum_{t=1}^{T-1} \gamma_t(i)$. The last sum also corresponds to the number of times we are in state q_i. We derive:

- The new estimate of a_{ij}:

$$\begin{aligned}\hat{a}_{ij} &= \frac{\text{expected number of transitions from state } q_i \text{ to state } q_j}{\text{expected number of transitions from state } q_i}, \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}.\end{aligned}$$

- The initial state estimates $\pi_i = \gamma_1(i)$.
- The observation estimates:

$$\begin{aligned}\hat{b}_i(v_k) &= \frac{\text{expected number of times in state } q_i \text{ and observing symbol } v_k}{\text{expected number of times in state } q_i}, \\ &= \frac{\sum_{o_t=v_k, 1\le t\le T}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}.\end{aligned}$$

7.3 Tagging with Decision Trees

So far, we used the maximum likelihood to estimate part of speech and word probabilities. We can replace it with decision trees induced from an annotated corpus. The tagging performance could be superior when the training set is small.

TreeTagger (Schmid 1994, 1995) is a stochastic tagger that replaces the maximum likelihood estimate with a binary decision tree to estimate $P(t_i|t_{i-2}, t_{i-1})$. Figure 7.9 shows an example of an imaginary tree where the conditional probability

$P(NN|DET, ADJ)$ is read from the tree by examining t_{-1} and t_{-2}, here ADJ and DET, respectively. The probability estimate is 0.70.

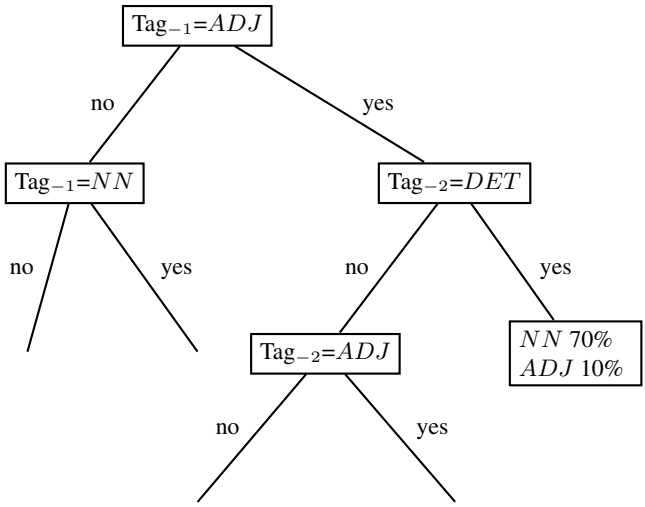

Fig. 7.9. A decision tree to estimate POS frequencies where is a noun, DET, a determiner, and ADJ, an adjective. After Schmid (1994).

The decision tree is built from a training set of POS trigrams t_{-2}, t_{-1}, t_0 extracted from an annotated corpus. The condition set is $t_{-i} = v$, with $i \in \{1, 2\}$ and $v \in T$, where T is the tagset.

The idea is to use the entropy of the POS trigams where the random variable is t_0. The entropy is then defined as:

$$-\sum_{t_0 \in T} P(t_0) \log_2 P(t_0).$$

If the total number of tokens is N, the entropy is estimated as:

$$-\sum_{t_0 \in T} \frac{C(t_0)}{N} \log_2 \frac{C(t_0)}{N}.$$

The decision tree minimizes the information it needs to identify the third tag, t_0, given the two preceding tags, t_{-2} and t_{-1}. This reflects the minimal amount of information brought by the third tag of a trigram.

To find the root node, the algorithm creates all the possible partitions of the training set according to the values of t_{-2} and t_{-1}. It computes the weighted average of the entropy of the positive and negative examples. The root condition corresponds to the values i and v with $i \in \{1, 2\}$ and $v \in T$ that minimize

$$-\frac{p}{p+n}\sum_{t_0\in T}\frac{C(t_0,t_{-i}=v)}{p}\log_2\frac{C(t_0,t_{-i}=v)}{p}-\frac{n}{p+n}\sum_{t_0\in T}\frac{C(t_0,t_{-i}\neq v)}{n}\log_2\frac{C(t_0,t_{-i}\neq v)}{n},$$

where p is the count of the trigrams that pass the test to be the root condition, and n is the count of trigrams that do not pass the test. $C(t_0, t_{-1} = v)$ is the count of trigrams t_{-2}, t_{-1}, t_0 that pass the test and where the third tag is t_0, and $C(t_0, t_{-1} \neq v)$ the count of trigrams that do not pass the test and where the third tag is t_0.

The algorithm stops expanding the tree and creates a leaf when the next node would gather a number of positive or negative trigrams below a certain threshold, 2, for example.

7.4 Unknown Words

For stochastic taggers, the main issue to tag unknown words is to estimate $P(w|t)$. Carlberger and Kann (1999) proposed to use suffixes or more precisely word endings to compute the estimate. They counted the number of word types with common word endings of length i, $C(w_{end-i}, t)$, for each tag t in the tagset, with i ranging from 0 to L. The estimate $P(w|t)$ for unknown word is then

$$P_{est}(w|t) = \sum_{i=0}^{L} \alpha_i \frac{C(w_{end-i}, t)}{\sum_{\tau\in\text{tagset}} C(w_{end-i}, \tau)}.$$

where α_i are parameters optimized on the training set. They tried their formula with increasing values of L, and they found that tagging accuracy did not improve for $L > 5$.

If $L = 0$, $P_{est}(w|t) = \frac{C(t)}{\sum_{\tau\in\text{tagset}} C(\tau)}$ corresponds to the proportion of part of speech t among the word types.

7.5 An Application of the Noisy Channel Model: Spell Checking

An interesting application of the noisy channel model is to help a spell checker rank candidate corrections (Kernighan et al. 1990). In this case, the source sequence is a correct string c that produces an incorrect one called the typo t through the noisy channel. The most likely correction is modeled as

$$\hat{c} = \arg\max P(c)P(t|c).$$

Possible typos are deletion, insertion, substitutions, and transpositions. In their original paper, Kernighan et al. allowed only one typo per word. Typo frequencies are estimated from a corpus where:

- $del(xy)$ is the number of times the characters xy in the correct word were typed x in the training set.
- $ins(xy)$ is the number of times x is typed as xy in the training set.

- $sub(xy)$ is the number of times the character x is typed y.
- $trans(xy)$ is the number of times xy is typed as yx in the training set.

$P(t|c)$ is estimated as:

$$P(t|c) = \begin{cases} \frac{del(c_{p-1},c_p)}{C(c_{p-1},c_p)} & \text{if deletion,} \\ \frac{ins(c_{p-1},t_p)}{C(c_{p-1})} & \text{if insertion,} \\ \frac{sub(t_p,c_p)}{C(c_p)} & \text{if substitution,} \\ \frac{trans(c_p,c_{p+1})}{C(c_{p-1},c_p)} & \text{if transposition.} \end{cases}$$

where c_p is the pth character of c, and t_p the pth of t.

The algorithm needs four confusion matrices, of size 26×26 for English, that contain the frequencies of deletions, insertions, substitutions, and transpositions. The *del* matrix will give the counts $del(xy)$, how many times y was deleted after x for all the letter pairs, for instance, $del(ab)$.

The matrices can be obtained through hand-annotation or automatically. Hand-annotation is expensive, and Kernighan et al. described an algorithm to train automatically the matrices. It resembles the forward–backward procedure (Sect. 7.2.7).

The training phase initializes the matrices with equal values and applies the spelling algorithm to generate a correct word for each typo in the text. The pairs typo/corrected word are used to update the matrices. The algorithm is repeated on the original text to obtain new pairs and is iterated until the matrices converge.

7.6 A Second Application: Language Models for Machine Translation

Natural language processing was born with machine translation, which was one of its first applications. Facing competition from Russia after the Second World War, the government of the United States decided to fund large-scale translation programs to have a quick access to documents written in Russian. It started the field and resulted in programs like SYSTRAN, which are still in use today.

Given the relatively long history of machine translation, a variety of methods have been experimented on and applied. In this section, we outline how language models and statistical techniques can be used to translate a text from one language into another one. IBM teams pioneered statistical models for machine translation in the early 1990s (Brown et al. 1993). Their work is still the standard reference.

7.6.1 Parallel Corpora

Parallel corpora are the main resource of statistical language translation. Administrative or parliamentary texts of multilingual countries are widely used because they are easy to obtain and are often free. The Canadian Hansard or the European Parliament

Table 7.7. Parallel texts from the Swiss federal law on milk transportation.

German	French	Italian
Art. 35 Milchtransport	**Art. 35 Transport du lait**	**Art. 35 Trasporto del latte**
1 Die Milch ist schonend und hygienisch in den Verarbeitungsbetrieb zu transportieren. Das Transportfahrzeug ist stets sauber zu halten. Zusammen mit der Milch dürfen keine Tiere und milchfremde Gegenstände transportiert werden, welche die Qualität der Milch beeinträchtigen können.	1 Le lait doit être transporté jusqu'à l'entreprise de transformation avec ménagement et conformément aux normes d'hygiène. Le véhicule de transport doit être toujours propre. Il ne doit transporter avec le lait aucun animal ou objet susceptible d'en altérer la qualité.	1 Il latte va trasportato verso l'azienda di trasformazione in modo accurato e igienico. Il veicolo adibito al trasporto va mantenuto pulito. Con il latte non possono essere trasportati animali e oggetti estranei, che potrebbero pregiudicarne la qualità.
2 Wird Milch ausserhalb des Hofes zum Abtransport bereitgestellt, so ist sie zu beaufsichtigen.	2 Si le lait destiné à être transporté est déposé hors de la ferme, il doit être placé sous surveillance.	2 Se viene collocato fuori dall'azienda in vista del trasporto, il latte deve essere sorvegliato.
3 Milchpipelines sind nach den Anweisungen des Herstellers zu reinigen und zu unterhalten.	3 Les lactoducs des exploitations d'estivage doivent être nettoyés et entretenus conformément aux instructions du fabricant.	3 I lattodotti vanno puliti e sottoposti a manutenzione secondo le indicazioni del fabbricante.

proceedings are examples of them. Table 7.7 shows an excerpt of the Swiss federal law in German, French, and Italian on the quality of milk production.

The idea of machine translation with parallel texts is simple: given a sentence, a phrase, or a word in a **source language**, find its equivalent in the **target language**. The translation procedure splits the text to translate into fragments, finds a correspondence for each source fragment in the parallel corpora, and composes the resulting target pieces to form a translated text. Using the titles in Table 7.7, we can build pairs from the phrases *transport du lait* 'milk transportation' in French, *Milchtransport* in German, and *trasporto del latte* in Italian.

The idea of translating with the help of parallel texts is not new and has been applied by many people. A notable example is the Egyptologist and linguist Jean-François Champollion, who used the famous Rosetta stone, an early parallel text, to decipher Egyptian hieroglyphs from Greek.

7.6.2 Alignment

The parallel texts must be aligned before using them in machine translation. This corresponds to a preliminary segmentation and mark-up that determines the corresponding paragraphs, sentences, phrases, and words across the texts. Inside sen-

tences, aligned fragments are called **beads**. Alignment of texts in Table 7.7 is made easier because paragraphs are numbered and have the same number of sentences in each language. This is not always the case, however, and some texts show a significantly different sentence structure.

Gale and Church (1993) describe a simple and effective method based on the idea that

> "longer sentences in one language tend to be translated into longer sentences in the other language, and that shorter sentences tend to be translated into shorter sentences."

Their method generates pairs of sentences from the target and source texts, assigns them a score, which corresponds to the difference of lengths in characters of the aligned pairs, and uses dynamic programming to find the maximum likelihood alignment of sentences.

The sentences in the source language are denoted $s_i, 1 \leq i \leq I$, and the sentences in the target language $t_i, 1 \leq i \leq J$. $D(i,j)$ is the minimum distance between sentences $s_1, s_2, ..., s_i$ and $t_1, t_2, ..., t_j$, and $d(source_1, target_1; source_2, target_2)$ is the distance function between sentences. The algorithm identifies six possible cases of alignment through insertion, deletion, substitution, expansion, contraction, or merger. They are expressed by the formula below:

$$D(i,j) = \min \begin{pmatrix} D(i,j-1) + d(0,t_j;0,0) \\ D(i-1,j) + d(s_i,0;0,0) \\ D(i-1,j-1) + d(s_i,t_j;0,0) \\ D(i-1,j-2) + d(s_i,t_j;0,t_{j-1}) \\ D(i-2,j-1) + d(s_i,t_j;s_{i-1},0) \\ D(i-2,j-2) + d(s_i,t_j;s_{i-1},t_{j-1}) \end{pmatrix}.$$

The distance function is defined as $-\log P(alignment|\delta)$, with $\delta = (l_2 - l_1 c)/\sqrt{l_1 s^2}$, and where l_1 and l_2 are the lengths of the sentences under consideration, c the average number of characters in the source language L_2 per character in the target language L_1, and s^2 its variance. Gale and Church (1993) found a value of c of 1.06 for the pair French–English and 1.1 for German–English. This means that French and German texts are longer than their English counterparts: 6% longer for French and 10% for German. They found $s^2 = 7.3$ for German–English and $s^2 = 5.6$ for French–English.

Using Bayes' theorem, we can derive a new distance function:

$$-\log P(\delta|alignment) - \log P(alignment).$$

Gale and Church (1993) estimated the probability $P(alignment)$ of their six possible alignments with these figures: substitution 1–1: 0.89, deletion and substitution 0–1 or 1–0: 0.0099, expansion and contraction 2–1 or 1–2: 0.089, and merger 2–2: 0.011. They rewrote $P(\delta|alignment)$ as $2(1-P(|\delta|))$, which can be computed from statistical tables. See Gale and Church's original article.

Alignment of words and phrases uses similar techniques, however, it is more complex. Figures 7.10 and 7.11 show examples of alignment from Brown et al. (1993).

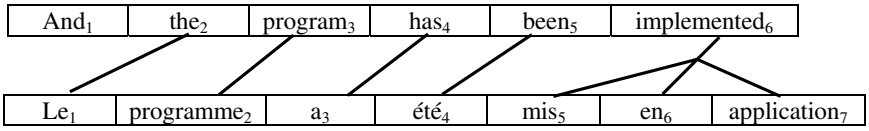

Fig. 7.10. Alignment. After Brown et al. (1993).

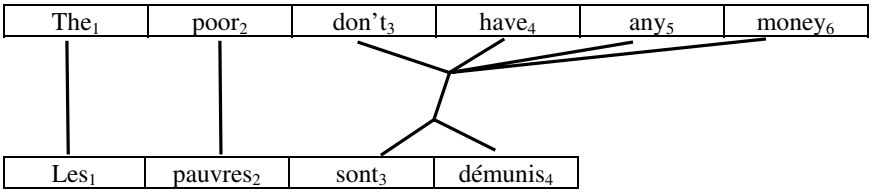

Fig. 7.11. A general alignment. After Brown et al. (1993).

7.6.3 Translation

Using a statistical formulation, given a source text, S, the most probable target text, T, corresponds to $\arg\max_T P(T|S)$, which can be rewritten as $\arg\max_T P(T)P(S|T)$. The first term, $P(T)$, is a language model, for instance, a trigram model, and the second one, $P(S|T)$, is the translation model. In their original article, Brown et al. (1993) used French as the source language and English as the target language with the notations F and E. They modeled the correspondence between a French string $f = f_1, f_2, ..., f_m$ and an English string, $e = e_1, e_2, ..., e_l$.

The first step is to rewrite the translation model as

$$P(f|e) = \sum_a P(f, a|e),$$

where a is the alignment between the source and target sentences and where each source word has one single corresponding target word. The target word can be the empty string. The alignment is represented by the string $a = a_1, a_2, ..., a_m$, where a_j is the position of the corresponding word in the English string as $a_j = i$, which denotes that word j in the French string is connected to word i in the English string. When there is no connection $a_j = 0$. In the example of Fig. 7.10, we have the alignment $a = (2, 3, 4, 5, 6, 6, 6)$.

Brown et al. (1993) proposed five models ranging from relatively simple to pretty elaborate to work out concretely the formula. In their simplest model 1, they introduce the simplification:

$$P(f, a|e) = \frac{\varepsilon}{(l+1)^m} \prod_{j=1}^{m} t(f_j|e_{a_j}),$$

where $t(f_j|e_{a_j})$ is the translation probability of f_j given e_{a_j} and ε a small, fixed number.

Using the example in Fig. 7.10, the product in

$$P(\text{Le programme a été mis en application}, a|\text{And the program has been implemented})$$

for $a = (2, 3, 4, 5, 6, 6, 6)$ corresponds to the terms:

$$t(\text{Le}|\text{the}) \times t(\text{programme}|\text{program}) \times t(\text{a}|\text{has}) \times t(\text{été}|\text{been}) \times$$
$$t(\text{mis}|\text{implemented}) \times t(\text{en}|\text{implemented}) \times t(\text{application}|\text{implemented})$$

where t values are derived from aligned corpora. Summing over all the possible alignments, we obtain the probability of the translation of *Le programme a été mis en application* into *And the program has been implemented*.

7.7 Further Reading

A complete introduction to stochastic methods can be found in Charniak (1993). It notably includes a description of the Viterbi algorithm that enables users to speed the search of the optimal part-of-speech sequence. Magerman (1995) noted some errors in Charniak's book that are worth being corrected. Carlberger and Kann (1999) is a very readable and complete text to implement a stochastic tagger.

Brown et al. (1993) started the field on statistical translation models. The original article is worth reading. GIZA++ (Och and Ney 2000), a free software to train translation models, is available from: http://www.fjoch.com/GIZA++.html.

Exercises

7.1. Implement the stochastic part-of-speech tagging algorithm in Prolog or Perl using unigrams.

7.2. Implement the stochastic part-of-speech tagging algorithm in Prolog or Perl using bigrams.

7.3. Implement the Viterbi search for the bigram part-of-speech tagger.

7.4. Implement a spell checker in Prolog or Perl.