Survey: Finite-State Technology in Natural Language Processing[1]

Andreas Maletti[a]

[a]*Institute for Natural Language Processing, Universität Stuttgart, Pfaffenwaldring 5b, 70569 Stuttgart, Germany*

Abstract

In this survey, we will discuss current uses of finite-state information in several statistical natural language processing tasks. To this end, we will review standard approaches in tokenization, part-of-speech tagging, and parsing, and illustrate the utility of finite-state information and technology in these areas. The particular problems were chosen to allow a natural progression from simple prediction to structured prediction. We aim for a sufficiently formal presentation suitable for readers with a background in automata theory that allows to appreciate the contribution of finite-state approaches, but we will not discuss practical issues outside the core ideas. We provide instructive examples and pointers into the relevant literature for all constructions. We close with an outlook on finite-state technology in statistical machine translation.

Keywords: finite-state automaton, tree automaton, context-free grammar, natural language processing, tokenization, part-of-speech tagging, parsing, machine translation

1. Introduction

Finite-state technology has played a major part in the development of several state-of-the-art tools in natural language processing [1, 2]. In this survey, we will recall three major problems in the domain of natural language processing, whose state-of-the-art solutions have successfully utilized finite-state technology. Several other areas (e.g., computational morphology) have benefitted from finite-state technology as well, so we necessarily had to select. We chose the mentioned topics because they allow a nice progression from simple prediction to structured prediction, and in particular, require no deep linguistic background. In general, this survey is written for the theoretical computer scientist, so concepts from the area of formal languages are introduced rather tersely. Additional room is provided for the explanation of the application problems, their solutions in terms of automata, and their evaluation. References to the relevant literature are provided in the content sections, which are supposed to be rather self-contained, although we reuse some methodology in the parsing section.

In all three applications, an inital model is extracted from annotated (positive) training data using standard methods in supervised training. This initial model is then transformed into an automaton for further processing. In the area of tokenization, the finite-state technology essentially delivers the operations required to work with the extracted model. We illustrate this on the basic approach, in which we model a scorer for fully annotated data. Using standard finite-state operations we can obtain a scorer for unannotated data from this scorer. In the area of part-of-speech tagging we follow a similar approach and first model a scorer for the fully annotated data. However, in the next step we show how such a model can be optimized to given unannotated training data with the help of the well-known forward-backward algorithm. In this unsupervised optimization step, the structure of the automaton remains the same, but the transition weights are adjusted to better fit the unannotated data. Practical considerations like overfitting the data (i.e., the over-adjustment of the automaton such that it exactly represents only the training data) will not be discussed to allow a clean presentation.

In the last part, we will discuss the classical parsing problem for natural languages. Whereas we discuss weighted finite-state automata in the first two application areas, we will mostly talk about weighted context-free grammars and

Email address: maletti@ims.uni-stuttgart.de (Andreas Maletti)

[1]Research financially supported by the German Research Foundation (DFG) grant MA 4959 / 1-1.

weighted tree automata in this section. A preliminary section is provided to recall those basic notions. As before, we use supervised training to obtain an initial (local) model from annotated training data. Next we demonstrate a successful approach that combines the weight adjustment procedure, which we already discussed for part-of-speech tagging, with a change of the structure of the automaton. In this case, the process derives truly hidden finite-state information that is not present in the fully annotated training data, so this optimization is again unsupervised. This automatic deduction of hidden finite-state information outperforms all comparable efforts to manually provide additional annotation for parsing models in the training data. We conclude with a short outlook on the use of finite-state information in statistical machine translation. In this area, finite-state models are used to model the translation process, but typically local variants, which do not actually use the power of "hidden" finite-state information (i.e., they only use clues from the part of the input currently under consideration), are actually used. Some minor uses have been demonstrated in the literature, but the majority of the used models remains essentially local.

2. String automata

2.1. Basic notions

We denote the set of nonnegative integers by $\mathbb{N}$. An alphabet is simply a finite set. The set of all sequences (or words) over the alphabet Σ is denoted by Σ^*, of which $\varepsilon \in \Sigma^*$ is the empty word. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, and we write $w(i)$ to denote the i-th letter in w for all $1 \leq i \leq |w|$. Given $1 \leq i \leq j \leq |w|$, we let $w[i, j] = w(i) \cdots w(j)$ be the substring from position i to j in w. We will use semirings [3, 4] as weight structures. A semiring is an algebraic structure $(S, +, \cdot, 0, 1)$ that consists of two monoids $(S, +, 0)$ and $(S, \cdot, 1)$ such that (i) the additive monoid $(S, +, 0)$ is commutative and (ii) the generalized distributivity law holds for both directions; i.e.,

$$\Big(\sum_{i=1}^{k} s_i\Big) \cdot s = \sum_{i=1}^{k} (s_i \cdot s) \qquad \text{and} \qquad s \cdot \Big(\sum_{i=1}^{k} s_i\Big) = \sum_{i=1}^{k} (s \cdot s_i)$$

for all $k \in \mathbb{N}$ and $s, s_1, \ldots, s_k \in S$. The semiring $(S, +, \cdot, 0, 1)$ is commutative if its multiplicative monoid $(S, \cdot, 1)$ is commutative. Two commutative semirings will be particularly relevant for this contribution:

- the semiring $(\mathbb{Q}_{\geq 0}, +, \cdot, 0, 1)$ of nonnegative rational numbers together with the usual operations and
- the semiring $([0, 1], \max, \cdot, 0, 1)$ of the (rational) unit interval together with the maximum (as addition) and the usual multiplication.

In the following, let $(S, +, \cdot, 0, 1)$ be a semiring. Given a mapping $f\colon A \to S$, we let $\operatorname{supp}(f) = \{a \in A \mid f(a) \neq 0\}$.

Finite-state automata [5] have been extended already in the 1960s to handle weights [6]. The recent handbook [7] provides an in-depth discussion of the developed theory and its applications (also NLP applications in chapter 14 [8]). A weighted string automaton [over the semiring $(S, +, \cdot, 0, 1)$] is a tuple $\mathcal{A} = (Q, \Sigma, I, \mathrm{wt}, F)$, where Q is a finite set of states, Σ an alphabet, $I\colon Q \to S$ is an initial weight assignment, $\mathrm{wt}\colon Q \times \Sigma \times Q \to S$ is a transition weight assignment, and $F\colon Q \to S$ is a final weight assignment. The labeled graph underlying the transition weight assignment is $\operatorname{supp}(\mathrm{wt}) = \{\langle q, \sigma, p\rangle \mid \mathrm{wt}(q, \sigma, p) \neq 0\}$. The automaton $\mathcal{A}$ is acyclic if the underlying graph $\operatorname{supp}(\mathrm{wt})$ has no cycle; i.e., there exists no sequence $\langle q_0, \sigma_1, q_1\rangle, \ldots, \langle q_{k-1}, \sigma_k, q_k\rangle$ of elements of $\operatorname{supp}(\mathrm{wt})$ such that $q_0 = q_k$. Moreover, it is deterministic if $I(q) \neq 0$ for at most one $q \in Q$ and for every $q \in Q$ and $\sigma \in \Sigma$, there exists at most one $p \in Q$ such that $\langle q, \sigma, p\rangle \in \operatorname{supp}(\mathrm{wt})$. Finally, the automaton $\mathcal{A}$ is unambiguous if for every $k \in \mathbb{N}$ and $\sigma_1, \ldots, \sigma_k \in \Sigma$ there exists at most one sequence $\langle q_0, \sigma_1, q_1\rangle, \ldots, \langle q_{k-1}, \sigma_k, q_k\rangle$ of elements of $\operatorname{supp}(\mathrm{wt})$ such that $I(q_0) \neq 0$ and $F(q_k) \neq 0$.

For such a weighted string automaton $\mathcal{A}$ and $q, p \in Q$, we define the mapping $\mathcal{A}_{q,p}\colon \Sigma^* \to S$ recursively by

$$\mathcal{A}_{q,p}(\varepsilon) = \begin{cases} 1 & \text{if } q = p \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \mathcal{A}_{q,p}(\sigma w) = \sum_{r \in Q} \Big(\mathrm{wt}(q, \sigma, r) \cdot \mathcal{A}_{r,p}(w)\Big) \tag{1}$$

for all $\sigma \in \Sigma$ and $w \in \Sigma^*$. The weighted string automaton $\mathcal{A}$ generates the mapping $\mathcal{A}\colon \Sigma^* \to S$ given by $\mathcal{A}(w) = \sum_{q,p \in Q} \big(I(q) \cdot \mathcal{A}_{q,p}(w) \cdot F(p)\big)$ for every $w \in \Sigma^*$. The such generated mappings are called the regular weighted string languages, which are closed under several important operations. For example, when the semiring is commutative, they are closed under the Hadamard product [9], which generalizes the classical intersection and for two given mappings $f, g\colon \Sigma^* \to S$ returns the mapping $(f \cdot g)\colon \Sigma^* \to S$ given by $(f \cdot g)(w) = f(w) \cdot g(w)$ for all $w \in \Sigma^*$. Given

two weighted string automata $\mathcal{A}_1 = (Q_1, \Sigma, I_1, \mathrm{wt}_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, I_2, \mathrm{wt}_2, F_2)$, their Hadamard product is the weighted string automaton $\mathcal{A}_1 \cdot \mathcal{A}_2 = (Q_1 \times Q_2, \Sigma, I, \mathrm{wt}, F)$ given by $I(\langle q_1, q_2\rangle) = I_1(q_1) \cdot I_2(q_2)$,

$$\mathrm{wt}(\langle q_1, q_2\rangle, \sigma, \langle q_1', q_2'\rangle) = \mathrm{wt}(q_1, \sigma, q_1') \cdot \mathrm{wt}(q_2, \sigma, q_2') \ ,$$

and $F(\langle q_1, q_2\rangle) = F_1(q_1) \cdot F_2(q_2)$ for all $\langle q_1, q_2\rangle, \langle q_1', q_2'\rangle \in Q_1 \times Q_2$ and $\sigma \in \Sigma$. As suggested by the notation, provided that the semiring $(S, +, \cdot, 0, 1)$ is commutative, the weighted string automaton $\mathcal{A}_1 \cdot \mathcal{A}_2$ indeed generates the weighted language $\mathcal{A}_1 \cdot \mathcal{A}_2$. Similarly, regular weighted string languages are closed under alphabet change [9]. Let $r\colon \Sigma \to \Delta$ be a symbol relabeling, which we extend to strings $\overline{r}\colon \Sigma^* \to \Delta^*$ by $\overline{r}(\sigma_1 \cdots \sigma_k) = r(\sigma_1) \cdots r(\sigma_k)$ for all $k \in \mathbb{N}$ and $\sigma_1, \ldots, \sigma_k \in \Sigma$. If $f\colon \Sigma^* \to S$ is (effectively) regular, then also the weighted string language $g\colon \Delta^* \to S$ given by $g(v) = \sum_{w \in \Sigma^*,\, \overline{r}(w) = v} f(w)$ is (effectively) regular. Given a weighted string automaton $\mathcal{A} = (Q, \Sigma, I, \mathrm{wt}, F)$ generating f, we can construct a weighted string automaton $\mathcal{B} = (Q, \Delta, I, \mathrm{wt}', F)$ generating g as follows: For every $q, p \in Q$ and $\delta \in \Delta$, we let $\mathrm{wt}'(q, \delta, p) = \sum_{\sigma \in \Sigma,\, r(\sigma) = \delta} \mathrm{wt}(q, \sigma, p)$.

2.2. Supervised training and tokenization

Two of the most basic tasks that are performed early in many natural language processing applications [1, 2] are tokenization and sentence boundary disambiguation. In general, the input to a natural language processing application is simply a character stream in some encoding (e.g., UTF-8 or ASCII). During tokenization this character stream is separated into tokens, which are the basic units for further processing. Similarly, in applications that receive documents (i.e., character streams representing multiple sentences) as input, the token (or character) stream often needs to be further subdivided to indicate sentence boundaries. The goal of sentence boundary disambiguation is the identification of sentences in such streams. For many applications, tokens correspond roughly to words, but the task of sentence boundary disambiguation can be understood as tokenization, where the desired tokens are sentences. Due to this close connection, in the following we will focus on the general task of tokenization, which is similar to the task of lexical analysis [10] used in compilers or other text processors.

While programming languages are designed such that token identification is simple, natural language input cannot be strongly constrained (e.g., 50 years ago the sequence ':-)' was probably considered illegal in English) and applications have to deal the ambiguities present in natural language. However, many natural languages offer strong clues indicating tokens (e.g., whitespace to indicate word boundaries or full stops to indicate sentence boundaries), but some ambiguity typically remains. For example, in English a full stop is also used to denote ordinal numbers (e.g., '1.' for 'first' and '42.' for 'fourty-second') or at the end of an abbreviation (e.g., 'M.S.' for 'Master of Science' and 'Hon.' for 'Honorable'). In Chinese, the sentence end marker 。 is unambiguous, but words can consist of several characters and word boundaries are not indicated by whitespace. For example, the sentence [11]

上海计划到本世纪末实现人均国内生产总值五千美元。

Pinyin:[2] Shàng hǎi jì huà dào běn shì jì mò shí xiàn rén jūn guó nèi shēng chǎn zǒng zhí wǔ qiān měi yuán.

[Shanghai plans to reach the goal of 5,000 dollars in per capita GDP by the end of the century.]

should be word-segmented (i.e., tokenized into words) according to the Penn Chinese tree bank standard [13, 14] into

上海 计划 到 本 世纪 末 实现 人均 国内 生产 总值 五千 美元 。

The standard baseline for Chinese word-segmentation uses a pre-compiled (finite) lexicon of known Chinese words and matches (beginning-to-end) the longest possible lexicon entry (or an individual character if there is no match). Given a tokenization D (e.g., the sequence of tokens output by a tokenizer) and a reference tokenization D_{gold} (e.g., the tokenization performed by a human), precision p, recall r, and the F_1-score F_1 are defined as follows:

$$p = \frac{c}{|D|} \qquad \text{and} \qquad r = \frac{c}{|D_{\mathrm{gold}}|} \qquad \text{and} \qquad \mathrm{F}_1 = \frac{2pr}{p + r} \ ,$$

where c is the number of correctly recognized tokens in D. In other words, the F_1-score is the harmonic mean between precision p and recall r. The simple longest-match strategy already achieves almost 90% F_1-score [11]. For example,

[2] Official romanization system used in mainland China and Taiwan. Pinyin [12] essentially renders a Chinese character as the syllable (or the sound) that it represents. Unfortunately, this translation from character to syllable is not injective.

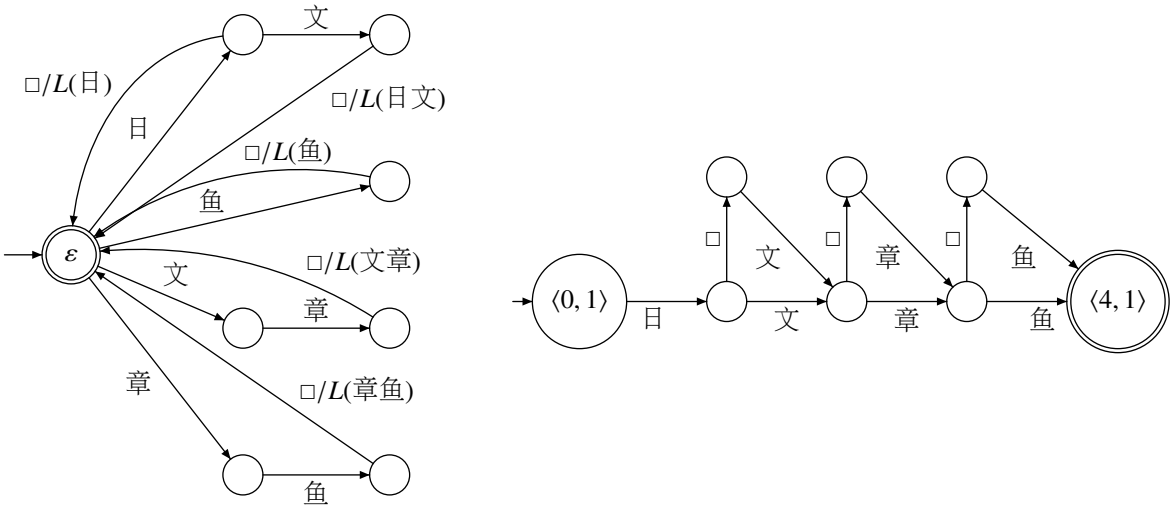


Figure 1: Illustration of the lexicon representation and the interspersed input. States are (mostly) unnamed here and unless indicated otherwise transitions have weight 1.

it correctly tokenizes the sentence [15]: 日文 章鱼 怎么 说 ？ [Pinyin: Rì wén zhāng yú zěn me shuō? Gloss: How to say octopus in Japanese?] as indicated, although the tokenization into lexicon entries is not unique in this example. The (wrong) tokenization 日 文章 鱼 怎么 说 ？ [Gloss: Japan (article) fish how say?] would also match lexicon entries.

The main drawback of the static heuristic is that it consistently makes mistakes and simply cannot tokenize certain sentences according to standard. Sproat et al. [15] suggest a weighted finite-state approach to Chinese word segmentation, in which they represent their scoring function (or model) as a weighted string automaton.[3] More precisely, they turn their probabilistic lexicon $L\colon \Sigma^+ \to [0,1]$ with finite support, where $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ and the probabilities represent occurrence probabilities, into a weighted string automaton $\mathcal{A} = (Q, \Sigma \cup \{\square\}, I, \mathrm{wt}, F)$ over $([0,1], \max, \cdot, 0, 1)$ such that

- $\square \notin \Sigma$ is a new distinguished symbol indicating a token boundary,
- $Q = \{\varepsilon\} \cup \{w[1,j] \mid w \in \mathrm{supp}(L),\ 1 \le j \le |w|\}$ contains a distinguished state ε and a state for each (non-empty) prefix of the strings represented by the lexicon L,
- $I(\varepsilon) = F(\varepsilon) = 1$ and $I(q) = F(q) = 0$ for all $q \in Q \setminus \{\varepsilon\}$, and
- the weights of the transitions are as follows:
  - $\mathrm{wt}(w, \sigma, w\sigma) = 1$ for all $\sigma \in \Sigma$, $w \in Q$ such that $w\sigma \in Q$,
  - $\mathrm{wt}(w, \square, \varepsilon) = L(w)$ for all $w \in \mathrm{supp}(L)$, and
  - all remaining transitions have weight 0.

The structure of the automaton $\mathcal{A}$ is presented in Figure 1. In other words, the automaton essentially recognizes a word w in the support of the lexicon L character-by-character and then recognizes a blank-symbol $\square$ behind w with the occurrence weight $L(w)$. This last transition resets the automaton into the initial state, which allows it to recognize the next word. A wealth of literature exists on the efficient representation of lexicons [16] that may also include morphological derivations of lemmas. This automaton $\mathcal{A}$ can score a given tokenization $z \in (\Sigma \cup \{\square\})^*$ by simply computing $\mathcal{A}(z) = \mathcal{A}_{\varepsilon,\varepsilon}(z)$. In other words, each candidate tokenization z can be scored by $\mathcal{A}$ using essentially the unigram language model score [1] for z (i.e., the product of the occurrence probabilities for the tokens in z). To determine the best tokenization (i.e., the one with the highest probability) for the input w, we can now exploit the nice closure properties of the mappings generated by weighted string automata. More precisely, since we already have a model that can score tokenized text, it remains to construct a weighted string automaton representing the possible

[3] They actually use a weighted finite-state transducer. For each input it outputs the separation into tokens, the phonetic realization of the characters in Pinyin [12], and the part-of-speech tag of the recognized word. These outputs are necessary to deal with homographs (i.e., characters with several phonetic realisations potentially indicating different meanings) and similar phenomena. We drop the outputs since most of that information (except for the separation naturally) is irrelevant for our exposition.

tokenizations for the input string. To this end, the input string $w \in \Sigma^+$ is simply interspersed with □-symbols. Let $\mathcal{A}_w = (Q_w, \Sigma \cup \{\square\}, I_w, \mathrm{wt}_w, F_w)$ be the weighted string automaton such that

- $Q_w = \{\langle i, j\rangle \mid 0 \le i \le |w|,\ j \in \{1, 2\}\}$ with initial weights $I_w(\langle 0, 1\rangle) = 1$ and $I_w(q) = 0$ for all $q \in Q_w \setminus \{\langle 0, 1\rangle\}$,
- final weights $F_w(\langle |w|, 1\rangle) = 1$ and $F_w(q) = 0$ for all $q \in Q \setminus \{\langle |w|, 1\rangle\}$, and
- the transition weights are as follows:
  - $\mathrm{wt}_w(\langle i, j\rangle, w(i + 1), \langle i + 1, 1\rangle) = 1$ for all $0 \le i < |w|$ and $j \in \{1, 2\}$,
  - $\mathrm{wt}_w(\langle i, 1\rangle, \square, \langle i, 2\rangle) = 1$ for all $1 \le i < |w|$, and
  - all remaining transitions have weight 0.

The structure of this automaton for the input $w =$ 日文章鱼 is displayed in Figure 1. The Hadamard product $\mathcal{A} \cdot \mathcal{A}_w$ then trivially represents a scorer for all possible tokenizations of w. In the example of Figure 1 only the tokenizations 日□文章□鱼 and 日文□章鱼 are permitted by the lexicon. Their scores are $L(日) \cdot L(文章) \cdot L(鱼)$ and $L(日文) \cdot L(章鱼)$, respectively, which coincide with their traditional unigram language model scores. The automaton $\mathcal{A} \cdot \mathcal{A}_w$, which represents the Hadamard product, scores all tokenizations of w according to the lexicon model $\mathcal{A}$ and prunes out all other strings (i.e., other strings are scored with 0). This automaton is always acyclic (because the automaton $\mathcal{A}_w$ is acyclic) and typically unambigious. Since the automaton $\mathcal{A}_w$ only uses the units $\{0, 1\}$ of the semiring, which always commute with all other elements, the commutativity of the semiring is not actually needed, so also non-commutative semirings can be used to model the lexicon.[4] To extract the best (i.e., highest-scoring) tokenization from $\mathcal{A} \cdot \mathcal{A}_w$ we can use Dijkstra's algorithm [17], which runs in time $O(|\mathcal{A}| \cdot |w| + |Q| \cdot |w| \cdot (\log |Q| + \log |w|))$. As demonstrated in [15] the framework of weighted string automata lends itself perfectly to the inclusion of additional knowledge sources such as models for morphological derivation or proper name detection.

Later approaches for the Chinese word-segmentation problem (e.g., those derived from Xue's work [11]) reinterpret the problem as a character tagging problem assigning essentially word-start, word-internal, and word-ending tags (or even more tags [18]) to each character. We omit a detailed presentation of those approaches here, since we will cover the similar part-of-speeching tagging problem in the next section. The shared tasks organized by SIGHAN (the special interest group for all aspects of Chinese language processing) consistently demonstrate that these tagging models based on hidden Markov models or conditional random fields outperform the longest-match baseline and the approach presented here. In addition, they demonstrate that the main remaining problem concerns out-of-vocabulary items (i.e., characters not seen during training) and the recognition of named entities.

2.3. Unsupervised training and part-of-speech tagging

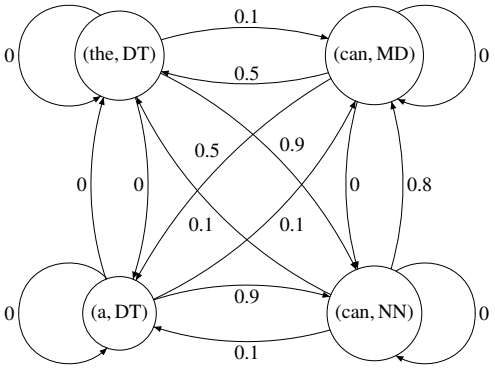
After the basic segmentation of the text is performed, and thus the common notions of 'sentence' and 'word' have been established, many applications require further processing of the token stream. For example, we might be interested in finding representative keywords for the token stream in order to provide an index into a text database [19] (also called terminology mining or term extraction). Such keywords are usually taken to be nouns, which requires us to detect noun occurrences. The goal of sentiment analysis [20] (or opinion mining) is detecting whether a text overall speaks positively or negatively about its subject. In this context, we are typically interested in content words (nouns, adjectives, verbs, negations, etc.) and might not be interested in function words (articles, pronouns, auxiliary verbs, etc.) since they often just serve a grammatical function and carry little lexical meaning. Also in this setup we would need a process that identifies occurrences of such word categories.

Part-of-speech tagging [21] is the process of assigning a word category (noun, adjective, verb, etc.) to each word of a sentence. The word categories are also called part-of-speech. Again, the most straightforward solution to this problem uses a lexicon, in which the possible word categories of a word might be listed (potentially with their frequencies). However, in many languages words can have several categories. For example, the English word 'well' has common uses in at least 5 word categories (adverb, adjective, interjection, noun, and verb) and the Wikipedia page for 'like' lists 10 categories (noun, verb, adverb, adjective, preposition, particle, conjunction, hedge, filler, and quotative). This ambiguity makes the task non-trivial because especially common words often have several associated categories. In fact we can form (artificial) sentences [22] containing the same word multiple times.

$$\underbrace{\text{Buffalo buffalo}}_{\text{Bison from Buffalo}} \quad \underbrace{\text{Buffalo buffalo buffalo}}_{\text{that are bullied by bison from Buffalo}} \quad \underbrace{\text{buffalo Buffalo buffalo.}}_{\text{themselves bully bison from Buffalo.}}$$

[4] Non-commutative semirings can be used to simulate weighted finite-state transducer behavior [15]. If we additionally want to output part-of-speech tags or the Pinyin romanization, then the multiplicative operation will use some form of concatenation and will thus be non-commutative.

	⟨the, DT⟩	⟨a, DT⟩	⟨can, MD⟩	⟨can, NN⟩
⟨the, DT⟩	0	0	0.1	0.9
⟨a, DT⟩	0	0	0.1	0.9
⟨can, MD⟩	0.5	0.5	0	0
⟨can, NN⟩	0.1	0.1	0.8	0



$$\begin{pmatrix} 0.3 \\ 0.4 \\ 0.2 \\ 0.1 \end{pmatrix} \quad \begin{matrix} \langle \text{the, DT} \rangle \\ \langle \text{a, DT} \rangle \\ \langle \text{can, MD} \rangle \\ \langle \text{can, NN} \rangle \end{matrix}$$

Figure 2: Transition matrix and initial state vector of a Markov model.

In this case, the associated sequence of assigned word categories would be

NNP (proper noun), NN (noun), NNP, NP, VBP (verb, non-3rd person singular present), VBP, NNP, NN,

where we used the word category abbreviations (or tags) from the Penn tree bank [23]. Similarly, the sentence

$$\underbrace{\text{He}}_{\text{PRP}} \; \underbrace{\text{lived}}_{\text{VBD}} \; \underbrace{\text{a}}_{\text{DT}} \; \underbrace{\text{long}}_{\text{JJ}} \; \underbrace{\text{and}}_{\text{CC}} \; \underbrace{\text{fruitful}}_{\text{JJ}} \; \underbrace{\text{life}}_{\text{NN}}$$

demonstrates some other standard word categories. We represent such annotated sentences as sentences over the alphabet $L \times S$, where L is the finite word inventory (i.e., lexicon) and S is the finite set of word categories. Both sets L and S can, for example, be obtained from the Penn tree bank. In this representation, the annotated sentence above, would be

$$\langle \text{He, PRP} \rangle \; \langle \text{lived, VBD} \rangle \; \langle \text{a, DT} \rangle \; \langle \text{long, JJ} \rangle \; \langle \text{and, CC} \rangle \; \langle \text{fruitful, JJ} \rangle \; \langle \text{life, NN} \rangle \; .$$

These sequences of word and assigned category pairs follow certain regularities [21] that can be modelled statistically. Some combinations are extremely infrequent (e.g., adjective followed by a determiner as in "as tall a tree as"), whereas others are very frequent (e.g., determiner followed by an adjective as in "a tall tree"). The stochastic process, which generates the sequences, can be modelled as a (homogeneous) Markov chain, which was originally devised by Markov [24] for sequences of letters in Russian texts. Note that we will again start in the supervised setting, where we follow the general strategy of building a model for the fully annotated data, which we will then modify to our needs using simple transformations. We first formally model the likelihood of a particular sequence $t = w_1 \cdots w_n$ with $w_i \in W = L \times S$ for all $1 \leq i \leq n$. Additionally, let X_i be a random variable associated to position i. Using the general product rule, we obtain the probability of t as

$$p(t) = p(X_1 = w_1, \ldots, X_n = w_n) = \prod_{i=1}^{n} p(X_i = w_i \mid X_1 = w_1, \ldots, X_{i-1} = w_{i-1}) \; .$$

Thus, we see that, in general, the probability of an element in the sequence depends on all previous positions. We could now assume that the variables $X_1, \ldots, X_n$ are statistically independent, but we have already seen that this is an unrealistic assumption. Instead Markov assumed that the probability of an element only depends on the previous element. This assumption is the defining feature of Markov chains and demands that there exists an $|W| \times |W|$-matrix $\mathcal{M}$ such that

$$p(X_i = w_i \mid X_1 = w_1, \ldots, X_{i=1} = w_{i-1}) = p(X_i = w_i \mid X_{i-1} = w_{i-1}) = \mathcal{M}_{w_{i-1}, w_i}$$

for all $2 \leq i \leq n$. An example illustration of such a matrix is presented in Figure 2. Under this assumption, the probability of t simplifies to $p(t) = \vec{v}_{w_1} \cdot \prod_{i=2}^{n} \mathcal{M}_{w_{i-1}, w_i}$ for a stochastic vector $\vec{v}$ of length $|W|$ with $\vec{v}_w = p(X_1 = w)$ for every $w \in W$. A Markov chain can easily be modelled as a weighted string automaton as follows: Given the

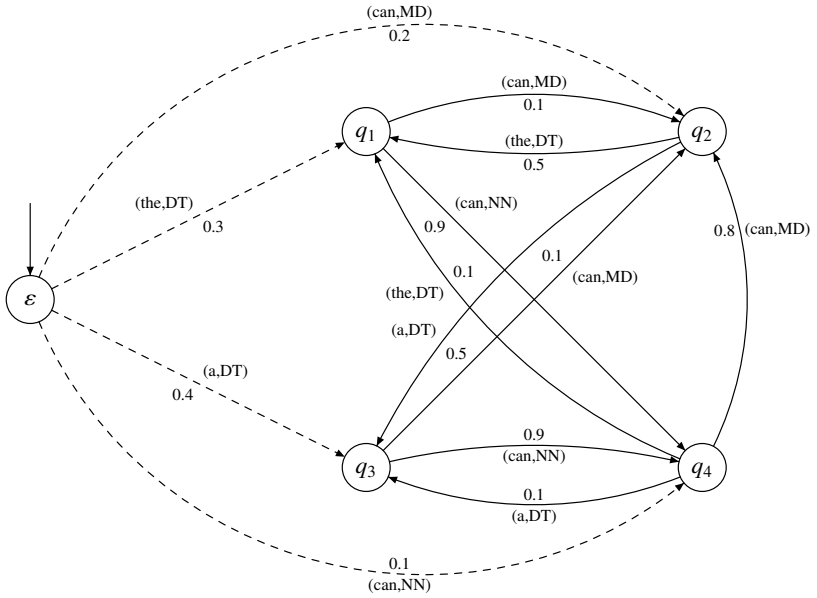


Figure 3: Illustration of the weighted string automaton corresponding to a Markov chain.

defining parameters W, $\mathcal{M}$, and $\vec{\nu}$ of the Markov chain, we construct the automaton $\mathcal{A} = (W \cup \{\varepsilon\}, W, I, \mathrm{wt}, F)$ over $(\mathbb{Q}_{\geq 0}, +, \cdot, 0, 1)$, where $I(\varepsilon) = F(\varepsilon) = 1$, $I(w) = 0$, and $F(w) = 1$ for all $w \in W$, and

$$\mathrm{wt}(\varepsilon, w, w'') = \begin{cases} \vec{\nu}_w & \text{if } w = w'' \\ 0 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \mathrm{wt}(w', w, w'') = \begin{cases} \mathcal{M}_{w,w'} & \text{if } w = w'' \\ 0 & \text{otherwise} \end{cases}$$

for all $w, w' \in W$ and $w'' \in W \cup \{\varepsilon\}$. The general structure is illustrated in Figure 3, which shows the weighted string automaton for the Markov chain of Figure 2. Note that the automaton $\mathcal{A}$ is deterministic. Clearly, $\mathcal{A}(t) = p(t)$ for all $t \in W^*$. We have arrived at a visible Markov model for the annotated sentences. It is called 'visible' because the input string actually dictates the state behavior of the automaton. If the i-th letter of the input string is $w \in W$, then the automaton $\mathcal{A}$ needs to switch into state w during the i-th transition (since the weight of all other transitions is 0). Moreover, given annotated sentences as training data (e.g., the Penn tree bank), we can easily determine W and estimate the parameters $\vec{\nu}$ and $\mathcal{M}$ by maximum likelihood estimation [25]. Assuming that the training data $(t_1, \ldots, t_k)$ of sequences $t_1, \ldots, t_k \in W^*$ is representative for the general distribution, we have that $\vec{\nu}_w \approx \frac{c_1(w)}{k}$ and $\mathcal{M}_{w,w'} \approx \frac{c(ww')}{c(w') - c_1(w')}$ for every $w, w' \in W$, where $c_1(w'')$ is the number of occurrences of w'' at the first position and $c(t)$ is the number of occurrences of the sequence $t \in W^*$ (at any position) in the training examples $t_1, \ldots, t_k$. An excellent and more detailed introduction into these models can be found in the book [1].

At this point, we managed to extract a weighted string automaton from fully annotated data. The automaton runs on (and scores essentially the bigrams in) sentences that are already annotated with word categories, but in the standard use-case outside training those categories are not available. To determine the best annotation for an unannotated string, we can again construct a weighted string automaton $\mathcal{A}_u$ for the given input string $u \in L^*$ that generates all possible word category annotations for u. Then the Hadamard product $\mathcal{A} \cdot \mathcal{A}_u$ scores all word category annotations and extracting the best derivation (using, for example, the Dijkstra algorithm) yields the best annotation. Alternatively, the Viterbi algorithm [26] essentially combines these two steps and can directly be employed. With these approaches we can determine that the most likely annotation for "can the" under the model of Figure 3 is $\langle \mathrm{can}, \mathrm{MD} \rangle \langle \mathrm{the}, \mathrm{DT} \rangle$ (probability: $0.2 \cdot 0.5$), whereas $\langle \mathrm{can}, \mathrm{NN} \rangle \langle \mathrm{the}, \mathrm{DT} \rangle$ is scored only $0.1 \cdot 0.1$. Since we already demonstrated this approach, we omit a detailed exposition here, but note that the Hadamard product can also be used to introduce additional knowledge into the model. For example, in English certain word endings (e.g., "-ly", "-ing", "-ize", and "-al") are strongly indicative for some word categories (adverbs, nouns, verbs, and adjectives, respectively). Another model that encodes such additional information can easily be combined with $\mathcal{A}$ with the help of the Hadamard product.

To showcase the power of the finite-state approach, we switch to the unsupervised setting and demonstrate how we can tune (or train) our model to additional training material that is not annotated (i.e., tokenized English text). The annotated training data (e.g., Penn tree bank) might not be representative for the texts that we want to use our model on. In such setups, we would like to adjust our model to accommodate the particularities of our evaluation texts. The special variant used here is typically called weight training, and we will illustrate it on the forward-backward (or Baum-Welch) algorithm [27].

The next goal is to fine-tune our model $\mathcal{A}$ to a finite sequence $(u_1, \ldots, u_n)$ with $u_i \in L^*$ of unannotated training examples (i.e., tokenized English sentences). To this end, we first note that our model $\mathcal{A}$ also assigns a probability to unannotated sentences. More precisely, the probability of a sentence $u \in L^*$ is given by summing over all the possible annotations for u; i.e., $p(u) = \sum_{t \in W^*, \overline{\pi_1}(t)=u} p(t) = \sum_{t \in W^*, \overline{\pi_1}(t)=u} \mathcal{A}(t)$, where $\pi_1 \colon W \to L$ is the projection on the first component (i.e., $\pi_1(\langle \ell, s \rangle) = \ell$ for all $\ell \in L$ and $s \in S$) and $\overline{\pi_1} \colon W^* \to L^*$ is its homomorphic extension to strings. Fortunately, regular weighted string languages are closed under alphabet change, so we can easily compute a weighted string automaton $\mathcal{B}$ from $\mathcal{A}$ that computes the probability for each unannotated string. For the particular automaton of Figure 3, the corresponding automaton after the projection looks exactly alike with the exception that all input labels only contain the first component. Note that this automaton is typically no longer deterministic. With it, we can efficiently compute the probability $\mathcal{B}$(can the) of the unannotated string "can the", which is 0.11 ($0.1 = 0.2 \cdot 0.5$ along the states $q_2 q_1$ and $0.01 = 0.1 \cdot 0.1$ along the states $q_4 q_1$), but the internal state changes are no longer observable, which justifies the name 'hidden Markov model'. The guiding principle of the forward-backward algorithm is the maximization of the training data $(u_1, \ldots, u_n)$. Consequently, for each $\mathrm{wt} \colon Q \times L \times Q \to \mathbb{Q}_{\geq 0}$, let $\mathcal{B}_{\mathrm{wt}} = (Q, L, I, \mathrm{wt}, F)$ be the weighted string automaton $\mathcal{B}$ with transition weights 'wt'. To keep the presentation simple, we assume that there is a unique initial state q_0 (i.e., there exists a unique $q \in Q$ with $I(q) \neq 0$), a unique final state q_f (i.e., there exists a unique state $q \in Q$ with $F(q) \neq 0$). Finally, we assume that the training data $(u_1, \ldots, u_n)$ is independently identically distributed. Given the last assumption, we want to determine

$$\widehat{\mathrm{wt}} = \operatorname*{arg\,max}_{\mathrm{wt} \colon Q \times L \times Q \to \mathbb{Q}_{\geq 0}} \left(\prod_{i=1}^{n} \mathcal{B}_{\mathrm{wt}}(u_i) \right) , \tag{2}$$

which is the transition weight assignment that maximizes the probability of the training data U (maximum likelihood estimation). Unfortunately, no analytic method is known that solves this problem in general, but the forward-backward algorithm will improve a given transition weight assignment 'wt' such that the probability of the data does not decrease. Iterating the algorithm then yields a (classical hill-climbing) approximation procedure for $\widehat{\mathrm{wt}}$ that might get stuck in local maxima.[5] We let $\mathcal{B} = \mathcal{B}_{\mathrm{wt}}$ be the automaton with the current transition weight assignment. The algorithm consists of two steps since it is an instance of the famous expectation-maximization (EM) algorithm [32]: (a) an expectation step, in which the contribution of each transition is calculated, and (b) a maximization step. In the expectation step, we compute the "contribution" p_τ of each transition $\tau = \langle q, \ell, q' \rangle \in Q \times L \times Q$ in the product of (2) as

$$p_\tau = \sum_{i=1}^{n} \Bigg(\sum_{\substack{u_i = v \ell w \\ v, w \in L^*}} \mathcal{B}(v)_{q_0, q} \cdot \mathrm{wt}(q, \ell, q') \cdot \mathcal{B}(w)_{q', q_\mathrm{f}} \Bigg) .$$

The name 'forward-backward algorithm' is derived from the efficient calculation of the probabilities $\mathcal{B}(v)_{q_0,q}$ [forward step] and $B(w)_{q',q_\mathrm{f}}$ [backward step] for the prefixes v and suffixes w of each sentence u_i of the training data $(u_1, \ldots, u_n)$. We note that $\mathcal{B}(v)_{q_0,q}$ and $\mathcal{B}(w)_{q',q_\mathrm{f}}$ can be efficiently computed using (1) by storing the intermediate results. Next, in the maximization step, we set the transition probabilities relative to their contribution. We let

$$\mathrm{wt}'(q, \ell, q') = \frac{p_{\langle q, \ell, q' \rangle}}{\sum_{q'' \in Q,\, \ell' \in L} p_{\langle q, \ell', q'' \rangle}}$$

for all $q, q' \in Q$ and $\ell \in L$. It was shown [33] that this procedure works in the sense that the probability of the training data improves; i.e., $\prod_{i=1}^{n} \mathcal{B}_{\mathrm{wt}}(u_i) \leq \prod_{i=1}^{n} \mathcal{B}_{\mathrm{wt}'}(u_i)$. Iterating the two steps, yields better and better transition weights

[5] Due to this behavior, the optimization should be started on several initial weight assignments and techniques like simulated annealing [28, 29] should be used to avoid getting stuck and potentially identify a global maximum. There is a wealth of literature [30, 31] on the selection of initial weights as those can be chosen randomly, can be obtained by other methods (such as our initial weights), or can be obtained based on knowledge of the search space.

converging towards a local maximum. Applied to our example model of Figure 3 and the training sentences "can the" and "the can", we obtain the contributions

$$p_{\langle\varepsilon,\text{can},q_2\rangle} = 0.2 \cdot 0.5 \qquad \text{and} \qquad p_{\langle\varepsilon,\text{can},q_4\rangle} = 0.1 \cdot 0.1 \qquad \text{and} \qquad p_{\langle\varepsilon,\text{the},q_1\rangle} = 0.3 \cdot (0.1 + 0.9) \ .$$

With these contributions, the new transition weights are

$$\text{wt}'(\varepsilon, \text{can}, q_2) = \frac{0.1}{0.1 + 0.01 + 0.3} = 0.24 \qquad \text{and} \qquad \text{wt}'(\varepsilon, \text{can}, q_4) = \frac{0.01}{0.1 + 0.01 + 0.3} = 0.02 \ .$$

Clearly, their weights increase because we have evidence in the training data for sentences starting with "can". Similarly, the weight of the transition $\langle\varepsilon, \text{the}, q_1\rangle$ will increase from 0.3 to 0.73. To compensate, the transition weight for $\langle\varepsilon, \text{a}, q_3\rangle$ is set to 0 since this transition was not useful in the small training sentences "can the" and "the can". This "removal" of the transition $\langle\varepsilon, \text{a}, q_3\rangle$ already indicates a potential generalization problem of the approach to maximize the probability of the training data, but we omit a detailed discussion.[6]

Whereas the states in the modelling of the tokenization problem were mostly tools to represent a graph and were necessary to model long strings inside an automaton, the states in Markov models have a much more pronounced role. They again serve a clear representation role in visible Markov models, but in a hidden Markov model the states and the transitions indeed encode the unknown behavior of a system (or process). Optimization algorithms like the forward-backward algorithm assign transition weights and in this sense assign some "meaning" to the states. This use of states as a model of unknown behavior of an ill-understood process that generates the desired distribution is a major approach in many application areas (e.g., speech processing [34]), physics [35], and biology [36, 37]. Modern approaches to part-of-speech tagging also use refinements of the presented approach such as conditional random fields [38], which are strongly related to hidden Markov models [39].

3. Context-free grammars and tree automata

3.1. Basic notions

From now on, we assume that $(S, +, \cdot, 0, 1)$ is a commutative semiring. A weighted context-free grammar [39, 40] is a tuple $\mathcal{G} = (N, T, S, P, \text{wt})$, where N is a finite set of nonterminals, T is an alphabet of terminal symbols such that $N \cap T = \emptyset$, $S \subseteq N$ is a set of initial nonterminals, $P \subseteq N \times (N \cup T)^*$ is a finite set of productions, and $\text{wt}\colon P \to S$. Each production $(n \to w) \in P$ induces a rewrite relation $\overset{n\to w}{\Longrightarrow} \subseteq (N \cup T)^* \times (N \cup T)^*$, which is given by

$$\overset{n\to w}{\Longrightarrow} = \{(unv, uwv) \mid u \in T^*,\ v \in (N \cup T)^*\} \ .$$

We let $\Longrightarrow_{\mathcal{G}} = \bigcup_{\rho\in P} \overset{\rho}{\Longrightarrow}$, and to keep the presentation simple, we additionally assume that for every $n \in N$, the statement $n \Longrightarrow_{\mathcal{G}}^{+} w$ implies that (i) $w \neq \varepsilon$ (no ε-productions) and (ii) $w \neq n$ (no cyclic chains). A (ξ_0, ξ_k)-derivation of $\mathcal{G}$ is a sequence $(\xi_0, \rho_1, \xi_1, \ldots, \rho_k, \xi_k)$ of sentential forms $\xi_0, \ldots, \xi_k \in (N \cup T)^*$ and productions $\rho_1, \ldots, \rho_k \in P$ such that $\xi_{i-1} \overset{\rho_i}{\Longrightarrow} \xi_i$ for each $1 \leq i \leq k$. The set of all (ξ_0, ξ_k)-derivations of $\mathcal{G}$ is denoted by $\text{Der}_{\mathcal{G}}(\xi_0, \xi_k)$. The weighted context-free grammar $\mathcal{G}$ generates the weighted language $\mathcal{G}\colon T^* \to S$, which is defined for every $w \in T^*$ by

$$\mathcal{G}(w) = \sum_{n\in S} \Bigg(\sum_{(n,\rho_1,\xi_1,\ldots,\rho_k,w)\in \text{Der}_{\mathcal{G}}(n,w)} \text{wt}(\rho_1) \cdot \ldots \cdot \text{wt}(\rho_k) \Bigg) \ .$$

Let Σ be an alphabet and X be a set such that $X \cap \Sigma = \emptyset$. The set of Σ-trees indexed by X, denoted by $T_\Sigma(X)$, is the smallest set T such that $X \subseteq T$ and $\sigma(t_1, \ldots, t_k) \in T$ for all $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T$. Instead of $T_\Sigma(\emptyset)$ we also write T_Σ, and $\sigma()$ will also be written just σ for all $\sigma \in \Sigma$. Given a tree $t \in T_\Sigma(X)$, its (rank-extended) positions $\text{pos}(t) \subseteq \mathbb{N}^* \times \mathbb{N}$ are inductively defined by $\text{pos}(x) = \{(\varepsilon, 0)\}$ for every $x \in X$ and $\text{pos}(\sigma(t_1, \ldots, t_k)) = \{(\varepsilon, k)\} \cup \{(i.w, j) \mid 1 \leq i \leq k, (w, j) \in \text{pos}(t_i)\}$ for all $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T_\Sigma(X)$.

[6] Adding the unannotated sentences from our annotated training data would mitigate the problem in our setup.

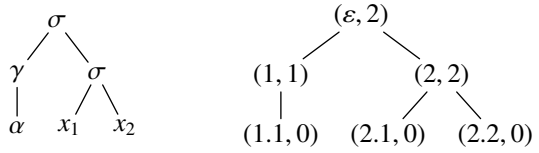


Figure 4: Tree t [left] and the same tree, in which each label was replaced by the rank-extended position of the node. The subtree $t|_2$ is $\sigma(x_1, x_2)$.

Each position $(w, k) \in \mathrm{pos}(t)$ in a tree $t \in T_\Sigma(X)$ has a label, denoted by $t(w)$, and induces a subtree, denoted by $t|_w$. Formally, these notions can be defined by $x(\varepsilon) = x|_\varepsilon = x$ for all $x \in X$ and

$$(\sigma(t_1, \ldots, t_k))(w) = \begin{cases} \sigma & \text{if } w = \varepsilon \\ t_i(v) & \text{if } w = i.v \text{ for } 1 \leq i \leq k \text{ and } v \in \mathrm{pos}(t_i) \end{cases}$$

$$(\sigma(t_1, \ldots, t_k))|_w = \begin{cases} \sigma(t_1, \ldots, t_k) & \text{if } w = \varepsilon \\ t_i|_v & \text{if } w = i.v \text{ for } 1 \leq i \leq k \text{ and } v \in \mathrm{pos}(t_i) \end{cases}$$

for all $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T_\Sigma(X)$. These notions are illustrated in Figure 4. The weighted context-free grammar $\mathcal{G} = (N, T, S, P, \mathrm{wt})$ also generates derivation trees. More formally, the set of derivation trees generated by $\mathcal{G}$ is

$$\mathrm{Der}(\mathcal{G}) = \{t \in T_N(T) \mid t(\varepsilon) \in S,\ \forall (w, k) \in \mathrm{pos}(t) \colon t(w) \in T \text{ or } (t(w) \to t(w.1) \cdots t(w.k)) \in P\} \ .$$

In other words, each derivation tree t has a root label $t(\varepsilon)$ that is from S, and at each position $(w, k) \in \mathrm{pos}(t)$ the label $t(w)$ is either a terminal symbol from T or the labels $t(w) \to t(w.1) \cdots t(w.k)$ form a production of P. The derivation tree $d \in \mathrm{Der}(\mathcal{G})$ is assigned the weight

$$\mathrm{wt}(d) = \prod_{\substack{(w,k) \in \mathrm{pos}(t) \\ t(w) \notin T}} \mathrm{wt}(t(w) \to t(w.1) \cdots t(w.k)) \ ;$$

i.e., the product of the weights of the productions occuring in the derivation tree, where each production (and its weight) is counted as often as it occurs.

A weighted tree automaton is a tuple $\mathcal{A} = (Q, \Sigma, Q_0, R, \mathrm{wt})$, where Q is the finite set of its states, Σ is its (terminal) alphabet, $Q_0 \subseteq Q$ is its set of initial states, R is its finite set of rules, where each is of the form $q \to \sigma(q_1, \ldots, q_k)$ for some $k \in \mathbb{N}$, $q, q_1, \ldots, q_k \in Q$, and $\sigma \in \Sigma$, and $\mathrm{wt} \colon R \to S$ is a weight assignment. For such a weighted tree automaton $\mathcal{A}$, we define a function $h_\mathcal{A} \colon T_\Sigma \times Q \to S$ recursively as follows:

$$h_\mathcal{A}(\sigma(t_1, \ldots, t_k), q) = \sum_{(q \to \sigma(q_1, \ldots, q_k)) \in R} \mathrm{wt}(q \to \sigma(q_1, \ldots, q_k)) \cdot \prod_{i=1}^{k} h_\mathcal{A}(t_i, q_i)$$

for all $q \in Q$, $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T_\Sigma$. A weighted tree language is simply a mapping $\varphi \colon T_\Delta \to S$ for some alphabet Δ. The weighted tree automaton $\mathcal{A}$ recognizes the weighted tree language $\mathcal{A} \colon T_\Sigma \to S$ such that $\mathcal{A}(t) = \sum_{q \in Q_0} h_\mathcal{A}(t, q)$ for all $t \in T_\Sigma$. The weighted tree language $\varphi \colon T_\Sigma \to S$ is regular [41] if there exists a weighted tree automaton $\mathcal{A}$ such that $\varphi = \mathcal{A}$. Let us provide a quick example using the commutative semiring $(\mathbb{Q}, +, \cdot, 0, 1)$ of rational numbers. Consider the weighted tree automaton $\mathcal{B} = (\{q, \top\}, \Sigma, \{q\}, R, \mathrm{wt})$ such that $\Sigma = \{\sigma, \alpha\}$, the set R of rules contains exactly the following rules

$$\begin{array}{llll} q \to \alpha & q \to \sigma(\top, \top) & q \to \sigma(q, \top) & q \to \sigma(\top, q) \\ \top \to \alpha & \top \to \sigma(\top, \top) \ , & & \end{array}$$

and $\mathrm{wt}(\rho) = 1$ for all $\rho \in R$. Then $h_\mathcal{B}(t, q) = |\mathrm{pos}(t)|$ and $h_\mathcal{B}(t, \top) = 1$ for all $t \in T_\Sigma$ such that for every position $(w, k) \in \mathrm{pos}(t)$ we have either (i) $t(w) = \sigma$ and $k = 2$ or (ii) $t(w) = \alpha$ and $k = 0$. Let $T \subseteq T_\Sigma$ be the set of trees fulfilling the previous condition, so for all other trees $t \in T_\Sigma \setminus T$ we have $h_\mathcal{B}(t, q) = h_\mathcal{B}(t, \top) = 0$. Consequently,

the automaton $\mathcal{B}$ recognizes the weighted tree language $\mathcal{B}$ that assigns the size to the trees only using binary 'σ' and nullary α and weight 0 to all other trees; i.e., for every $t \in T_\Sigma$

$$\mathcal{B}(t) = \begin{cases} |\mathrm{pos}(t)| & \text{if } t \in T \\ 0 & \text{otherwise.} \end{cases}$$

A detailed introduction into regular weighted tree languages can be found in the book chapter [42].

3.2. Supervised training and parsing

A number of applications needs more detailed information about the structure (syntax) of the sentence. For example, in the analysis of the semantics [43, 44] of a sentence the grammatical subject, verb, and its objects are of paramount importance. To detect and relate mentions [45] or to resolve pronouns, the noun phrases are typically identified with mentions. Finally, it was also demonstrated that syntax helps in certain machine translation tasks [46]. Let us start with an informal discussion. Given a tokenized sentence, which might already be part-of-speech tagged, the task of syntactical analysis (also called 'parsing') is to determine the grammatical parts (subject, verb with its objects, etc.) of the sentence and their relations. The principal grammatical parts and their potential relations are typically more or less agreed upon by linguists. The syntactical analysis shall identify the parts and their relation for the given sentence according to a fixed standard. Many different theories for the syntax of natural languages (e.g., constituent syntax [47] and dependency syntax [48]) exist, and we present an example analysis of the sentence "We must bear in mind the Community as a whole" as a constituent tree and a dependency tree in Figure 5. In the following, we will focus on the classic constituent syntax in the form of non-crossing trees. The Penn tree bank contains this annotation for roughly 50,000 English sentences.

Let us go into a little more detail on the constituent syntax of English. A sentence is typically built from a subject and a verbal complex and possibly modifiers. The verbal complex contains the verb and its objects. These grammatical parts are called constituents and the corresponding continuous substring is called a phrase. The main observation in constituent syntax is that constituents are hierarchically organized (larger constituents are formed from smaller ones), so that they can be organized in a tree. For example, in Figure 5 the substring "the Community" forms a noun phrase (NP) and the substring "the Community as a whole" forms another noun phrase that consists of the noun phrase "the Community" and the prepositional phrase (PP) "as a whole", which itself decomposes into the preposition (IN) "as" and the noun phrase "a whole". The phrases and their constituent types were hand-annotated in the Penn tree bank, so we again start in the supervised setting with fully annotated examples and extract a model directly from that. Following the historical development, we start with the extraction of a weighted context-free grammar $(N, T, S, P, \mathrm{wt})$, whose derivation trees will model the syntactical analysis. Let $(t_1, \dots, t_n)$ be the training data. First of all, the constituent types and part-of-speech tags will form the set N of nonterminals, of which those that occur (in some tree) at the root are made initial nonterminals S. Similarly, all the lexical items that occur in the training data will form the set T of terminals.[7] Consequently, we have $t_i \in T_N(T)$ and $t_i(\varepsilon) \in S$ for all $1 \le i \le n$. Since the trees in the training data are supposed to be derivation trees of the extracted grammar, also the used productions are evident and we set

$$P = \bigcup_{1 \le i \le n} \{t_i(w) \to t_i(w.1) \cdots t_i(w.k) \mid (w,k) \in \mathrm{pos}(t_i),\ k \neq 0\} \ .$$

Finally, we need to set the production weights. We again chose the maximum likelihood approach and simply count in $c(\rho)$ for each production $\rho \in P$, how often it occurs in the training data. More formally, for every production $(s \to s_1 \cdots s_k) \in P$ with $s \in N$ and $s_1, \dots, s_k \in N \cup T$, let

$$c(s \to s_1 \cdots s_k) = \sum_{1 \le i \le n} \left| \{(w,k) \in \mathrm{pos}(t_i) \mid t_i(w) = s,\ \forall 1 \le j \le k \colon t_i(w.j) = s_j\} \right| \ .$$

We can then normalize (e.g., by left-hand side) those counts to obtain probabilities and our production weights; namely, for every production $(s \to s_1 \cdots s_k) \in P$, we let

$$\mathrm{wt}(s \to s_1 \cdots s_k) = \frac{c(s \to s_1 \cdots s_k)}{\sum_{(s \to w) \in P} c(s \to w)} \ .$$

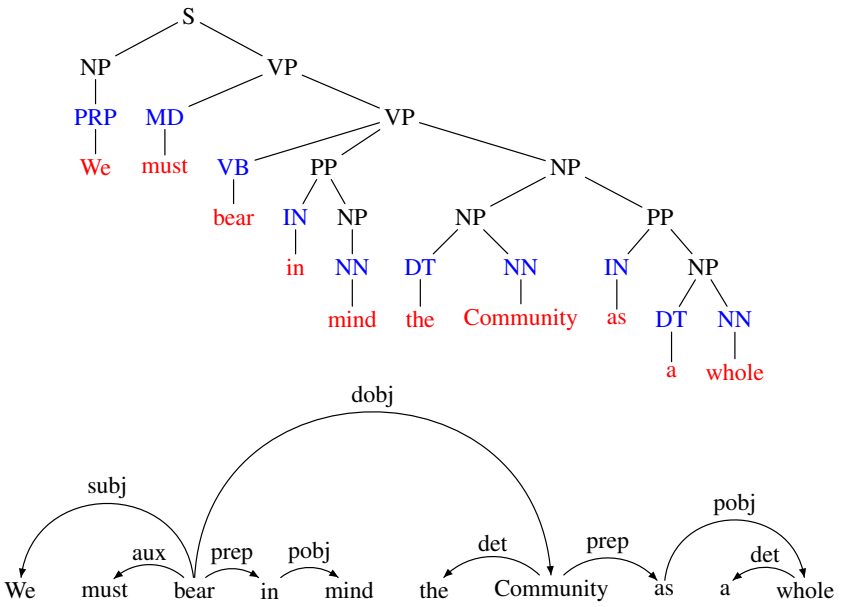


Figure 5: A constituency and dependency parse of an English sentence.

Before we proceed, let us explain ParseEval [49] quickly.[8] Note that in a tree $t \in T_N(T)$ each nonterminal occurrence w heads a certain substring, namely the string that is obtained by concatenating from left-to-right the lexical elements of the subtree $t|_w$. Moreover, constituents are the elements of N that are not part-of-speech tags. We indicated constituents in black in Figure 6. ParseEval essentially computes labeled precision p and recall r of constituents between two trees: a parse tree t and a reference tree u. Roughly speaking, labeled precision p is the number of correct constituents in t (i.e., the number of occurrences of constituents in t that head the same phrase as in the reference tree u) divided by the number of all constituent occurrences in t. Similarly, labeled recall r is the number of correct constituents in t divided by the number of all constituent occurrences in u. ParseEval is then the F_1-score $F_1 = \frac{2pr}{p+r}$ of p and r. This approach can trivially be extended to several trees forming a test set. To illustrate the notion, let us consider the two trees of Figure 6, in which the constituents are black. Given the right tree as reference, we obtain labeled precision $p = 0.9 = 90\%$ (because there are 10 occurrences of constituents in the parse tree and only 9 are correct; the NP heading "the Community as a whole" cannot be confirmed in the reference) and labeled recall $r = 1 = 100\%$ (because all 9 occurrences of constituents in the reference head the same substring in the parse tree). Overall, this yields $\frac{2pr}{p+r} = 0.95 = 95\%$ F_1-score.

If we extract a weighted context-free grammar $(N, T, S, P, \mathrm{wt})$ from the training data (Sections 2–21 of the Penn tree bank) as previously described, then with small modifications (unknown word handling and binarization) we already achieve 65–70% F_1-score (on Section 23) using ParseEval.[9] Clearly, 70% F_1-score does not compare favorably to scores of state-of-the-art parsers [54], which exceed 90%. The evaluation scores of the weighted context-free grammar approach presented above have been boosted by additional manual or automatic annotation of the training data. Many constituent types (like 'NP' and 'PP') have been subdivided into more specific types (like 'NP-SBJ' for the subject noun phrase or 'PP-TMP' for temporal prepositional phrases). This process is called subcategorization, and many subcategorizations have been proposed [55]. However, it is important to recall that in the evaluation (i.e., on the test set) the subcategorization is not performed. In other words, the parsers are still evaluated on the "base" categories

[7] We will not deal with unknown words in this survey.

[8] The actual description of ParseEval is slightly more involved. For example, ParseEval actually eliminates punctuation and unary nodes (i.e., nodes with exactly one child), and the original description does not take the node label into account. Although ParseEval received a lot of criticism [50, 51] over the years, it remains the dominant evaluation metric for statistical constituent parsers.

[9] Using the BitPar decoder [52, 53] with the Viterbi-parse option we obtain 69.84% F_1-score. Note that BitPar internally binarizes the grammar and adds a default unknown word handling.

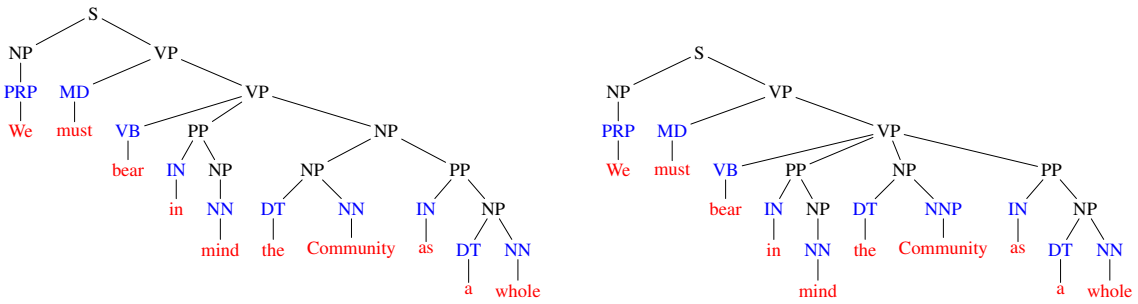


Figure 6: Parse tree (left) and reference tree (right).

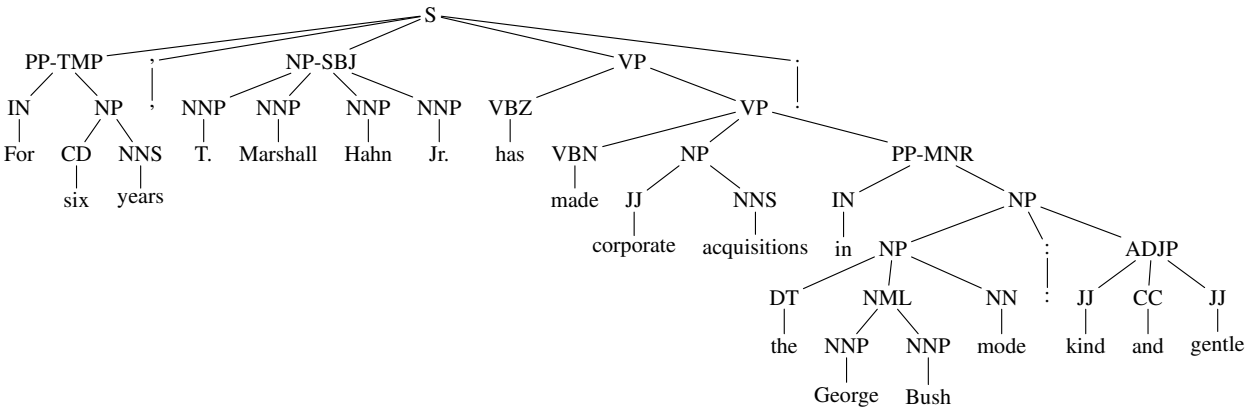


Figure 7: Actual parse tree from the PENN tree bank.

(constituent types). However, since the parser now operates with more fine-grained nonterminals, the derivation trees also contain the subcategorized nonterminals. Thus, a relabeling is applied to the derivation tree, which removes the additional information before evaluation. Let $t \in T_\Sigma(X)$ be a tree. Since the lexical elements are typically expressed as index X and not relabeled, a deterministic relabeling is a mapping $\chi\colon \Sigma \to \Delta$. It is extended to trees $\chi\colon T_\Sigma(X) \to T_\Delta(X)$ as follows: $\chi(x) = x$ for all $x \in X$ and $\chi(\sigma(t_1, \ldots, t_k)) = \chi(\sigma)(\chi(t_1), \ldots, \chi(t_k))$ for all $k \in \mathbb{N}$, $\sigma \in \Sigma$, and $t_1, \ldots, t_k \in T_\Sigma(X)$. We can extend it once more to weighted tree languages $\chi\colon S^{T_\Sigma(X)} \to S^{T_\Delta(X)}$, where as usual A^B denotes the set of all mappings $f\colon B \to A$, for every $\varphi\colon T_\Sigma(X) \to S$ and $u \in T_\Delta(X)$ by $(\chi(\varphi))(u) = \sum_{t \in T_\Sigma(X), \chi(t) = u} \varphi(t)$. Note that the sum in the previous sentence is always finite. In the parsing domain, the additional annotation is typically separated by a hyphen '-' or equals sign '=' from the base category (see Figure 7). The standard scorer [56] implements the relabeling that removes all information behind those symbols and the hyphen or equals sign itself. It is well-known [42] that relabeling the weighted derivations of a weighted context-free grammar yields a regular weighted tree language, and moreover, each regular weighted tree language can be obtained in this fashion. More formally, let $\mathcal{G} = (N, T, S, P, \mathrm{wt})$ be a weighted context-free grammar and $\chi\colon N \to N'$ be a relabeling. The weighted derivation language is $\mathrm{Der}_\mathcal{G}\colon T_N(T) \to S$ given by $\mathrm{Der}_\mathcal{G}(d) = \mathrm{wt}(d)$ for all $d \in \mathrm{Der}(\mathcal{G})$. Then $\chi(\mathrm{Der}_\mathcal{G})\colon T_{N'}(T) \to S$ is a regular weighted tree language. Vice versa, every regular weighted tree language $\varphi\colon T_\Sigma(X) \to S$ can be presented as $\varphi = \chi'(\mathrm{Der}_{\mathcal{G}'})$ for some relabeling χ' and weighted context-free grammar $\mathcal{G}'$. Consequently, irrespective of the chosen subcategorization, the model can actually be expressed as a weighted tree automaton. This even applies more generally to, for example, weighted tree substitution grammars and arbitrary subcategorizations [54].

Let us now model the relabeled derivations of a weighted context-free grammar with the help of a weighted tree automaton. We assume that all productions generating lexical items have a part-of-speech tag on the left-hand side. Moreover, we allow weighted tree automata rules of the form $q \to \sigma(w)$ for a state q, a part-of-speech tag σ, and lexical item w. This can easily be simulated by two "proper" rules $q \to \sigma(q_w)$ and $q_w \to w$ for some new state q_w at

the expense of an additional state per lexical item. Let $\mathcal{G} = (N, T, S, P, \mathrm{wt})$ be a weighted context-free grammar and $\chi\colon N \to N'$ be a deterministic relabeling. We construct the weighted tree automaton $\mathcal{A} = (N, N \cup T, S, P', \mathrm{wt}')$, where

$$P' = \{s \to \chi(s)(s_1, \dots, s_k) \mid (s \to s_1 \cdots s_k) \in P,\ \forall 1 \le i \le k\colon\ s_i \in N \cup T\}$$

and $\mathrm{wt}(s \to \chi(s)(s_1, \dots, s_k)) = \mathrm{wt}(s \to s_1 \cdots s_k)$ for all $(s \to s_1 \cdots s_k) \in P$ with $s_i \in N \cup T$ for all $1 \le i \le k$. It is routine to check that $\mathcal{A} = \chi(\mathrm{Der}_{\mathcal{G}})$. So far, this supervised approach mirrors the approach used for part-of-speech taggers: The states encode the constituent types (part-of-speech tags in part-of-speech tagging), which is the information that is generally not available outside the annotated training examples. The rules (or productions) and their weights encode how the states can be combined. Note that the parsing task also solves the part-of-speech tagging task since the part-of-speech tags are also derived, although they are excluded from the evaluation. Indeed the best part-of-speech taggers are actually parsers [57], which comes at the expense of a more potent and computationally more expensive model (weighted finite-state automaton vs. weighted context-free grammar).

3.3. Unsupervised training and the Berkeley parser

The introduction of fully automatic subcategorization [58–60] without external resources (i.e., unsupervised training) marked a fundamental change in direction. Beforehand most of the proposed subcategorizations were either manually annotated or automatically annotated with the help of other resources such as head information. The fully automatic approaches were later refined by Petrov [61, 62], whose approach we will present here. Essentially, we split each state into two almost identical copies and retrain the rule weights of the obtained weighted tree automaton using, for example, expectation maximization. To this end, we will use the original training material (as opposed to the approach presented for part-of-speech tagging), in which the desired subcategorization is naturally not present. Let us present the approach on an example. Assume that the two trees of Figure 8 are given as training data. From them we can extract the weighted tree automaton $\mathcal{A} = (N, N \cup T, S, R, \mathrm{wt})$, where

- $N = \{\mathrm{S}, \mathrm{NP}, \mathrm{VB}, \mathrm{DT}, \mathrm{NN}\}$ and $T = \{\mathrm{the}, \mathrm{a}, \mathrm{dragon}, \mathrm{sleeps}\}$ and $S = \{\mathrm{S}\}$, and
- the following rules are in R with their weight 'wt' indicated above the arrow:

$$\begin{array}{lll} \mathrm{S} \xrightarrow{1} \mathrm{S}(\mathrm{NP}, \mathrm{VB}) & \mathrm{VB} \xrightarrow{1} \mathrm{VB}(\mathrm{sleeps}) & \mathrm{DT} \xrightarrow{.5} \mathrm{DT}(\mathrm{the}) \\ \mathrm{NP} \xrightarrow{1} \mathrm{NP}(\mathrm{DT}, \mathrm{NN}) & \mathrm{NN} \xrightarrow{1} \mathrm{NN}(\mathrm{dragon}) & \mathrm{DT} \xrightarrow{.5} \mathrm{DT}(\mathrm{a})\ . \end{array}$$

Now we split every state into two copies. For the two rules $\mathrm{DT} \xrightarrow{.5} \mathrm{DT}(\mathrm{the})$ and $\mathrm{DT} \xrightarrow{.5} \mathrm{DT}(\mathrm{a})$, we create the four rules

$$\mathrm{DT\text{-}1} \xrightarrow{.51} \mathrm{DT}(\mathrm{the}) \qquad \mathrm{DT\text{-}1} \xrightarrow{.49} \mathrm{DT}(\mathrm{a}) \qquad \mathrm{DT\text{-}2} \xrightarrow{.49} \mathrm{DT}(\mathrm{the}) \qquad \mathrm{DT\text{-}2} \xrightarrow{.51} \mathrm{DT}(\mathrm{a})\ ,$$

where the weights were chosen slightly different to break the symmetry.[10] Mind that all states are split into two states at the same time. Now suppose that we arrived at the following weighted tree automaton $\mathcal{A}' = (N', N' \cup T, S, R', \mathrm{wt}')$, where

- $N' = \{\mathrm{S\text{-}1}, \mathrm{NP\text{-}1}, \mathrm{NP\text{-}2}, \mathrm{VB\text{-}1}, \mathrm{DT\text{-}1}, \mathrm{DT\text{-}2}, \mathrm{NN\text{-}1}\}$ and $T = \{\mathrm{the}, \mathrm{a}, \mathrm{dragon}, \mathrm{sleeps}\}$,
- $S = \{\mathrm{S\text{-}1}\}$, and
- the following rules with their weight indicated above the arrow:

$$\begin{array}{llll} \mathrm{S\text{-}1} \xrightarrow{.25} \mathrm{S}(\mathrm{NP\text{-}1}, \mathrm{VB\text{-}1}) & \mathrm{NP\text{-}1} \xrightarrow{1} \mathrm{NP}(\mathrm{DT\text{-}1}, \mathrm{NN\text{-}1}) & \mathrm{VB\text{-}1} \xrightarrow{1} \mathrm{VB}(\mathrm{sleeps}) & \mathrm{DT\text{-}1} \xrightarrow{.9} \mathrm{DT}(\mathrm{the}) \\ \mathrm{S\text{-}1} \xrightarrow{.75} \mathrm{S}(\mathrm{NP\text{-}2}, \mathrm{VB\text{-}1}) & \mathrm{NP\text{-}2} \xrightarrow{1} \mathrm{NP}(\mathrm{DT\text{-}2}, \mathrm{NN\text{-}1}) & \mathrm{NN\text{-}1} \xrightarrow{1} \mathrm{NN}(\mathrm{dragon}) & \mathrm{DT\text{-}2} \xrightarrow{.2} \mathrm{DT}(\mathrm{the}) \\ & & & \mathrm{DT\text{-}2} \xrightarrow{.8} \mathrm{DT}(\mathrm{a})\ . \end{array}$$

Using the provided training trees, we train the rule weights using the expectation maximization algorithm in much the same fashion as in Section 2.3. First, we consider all derivations of the training trees with any additional annotation,

[10] Setting the weights to equal values makes it difficult (if not impossible) for the learning procedure to distinguish the states.

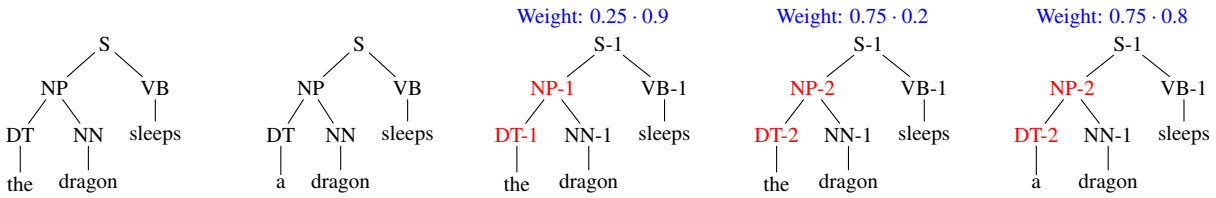


Figure 8: Two simple training trees (left) and three derivations of them (right).

which we display in Figure 8 together with their weights. Consequently, we obtain the following "expected" counts for two rules:

$$c(\text{S-1} \to \text{S(NP-1, VB-1)}) = 0.25 \cdot 0.9 = 0.225 \qquad \text{and} \qquad c(\text{S-1} \to \text{S(NP-2, VB-1)}) = 0.75 \cdot 0.2 + 0.75 \cdot 0.8 = 0.75$$

In the maximization step, we again normalize the rule weight, so we obtain the new rule weights

$$\text{wt}''(\text{S-1} \to \text{S(NP-1, VB-1)}) = \frac{0.225}{0.225 + 0.75} = 0.23 \qquad \text{and} \qquad \text{wt}''(\text{S-1} \to \text{S(NP-2, VB-1)}) = \frac{0.75}{0.225 + 0.75} = 0.77$$

for the mentioned rules. Intuitively, the second rule was more useful in the training data, so its weight was increased with the goal to increase the overall likelihood of the training data. At this point we can re-evaluate the probability of the training corpus, and if we do not observe significant improvements, then we remerge certain splits. In fact, a simple and direct way to evaluate the approximate gain from an individual split has been obtained [61], but we refer to the thesis [62] for the details. This completes one iteration, and we can now either stop the training process or continue with another iteration by splitting all remaining states again.

It was demonstrated [61, 62] that the automatic splits capture interesting linguistic phenomena. Moreover, the Berkeley parser, which is trained in the described manner, obtains an F_1-score of 90.7% on the standard data from the Penn tree bank. We note that the states here capture completely unobserved information (i.e., information that is not even present in the fully annotated training data). The result also demonstrates that even hand-crafted subcategorizations are outperformed by the automatic approach, so the automatically aquired finite-state information has a higher utility than the manual or human-designed subcategorization. A case in point is the subcategorization manually annotated in the Penn tree bank. As demonstrated [55] these annotations typically have negative utility for parsing the test set when used directly. This shows that the utility of subcategorizations is hard to predict, and given sufficiently large training resources it seems best to acquire them automatically [54].

4. Conclusion and outlook to machine translation

We have demonstrated the usefulness of the finite-state approach in several natural language processing tasks. The finitely many states can often nicely and compactly encode the desired annotation (as in the tokenization and part-of-speech tagging tasks) or even completely unknown fine-grained distinctions that go well beyond the annotation provided in the supervised setting (as in the Berkeley parser scenario). The availability of toolkits [63, 64] that implement finite-state models together with efficient manipulation procedures allows us to focus on the modelling task, in which we derive a finite-state model that directly or partially encodes the desired solution. The final representation of the solution can then be extracted from the built model using generic algorithms provided by the toolkits as demonstrated by the examples in this contribution. In addition, the finite-state approach is also ideally suited to compactly represent n-best lists or lattices of solutions, which allows to efficiently provide access to several weighted solutions to further post-processing tasks. Finally, the availability of composition procedures [65] for finite-state models with output allows the modular development of a pipeline for the task at hand without the disadvantages usually encountered when evaluating such pipelines. The latter approach was extremely successful in speech recognition and is used in most commercially available systems.

Finally, let us provide a quick outlook on finite-state technology in statistical machine translation. The task of machine translation is to transfer an input sentence given in some natural language into another natural language with

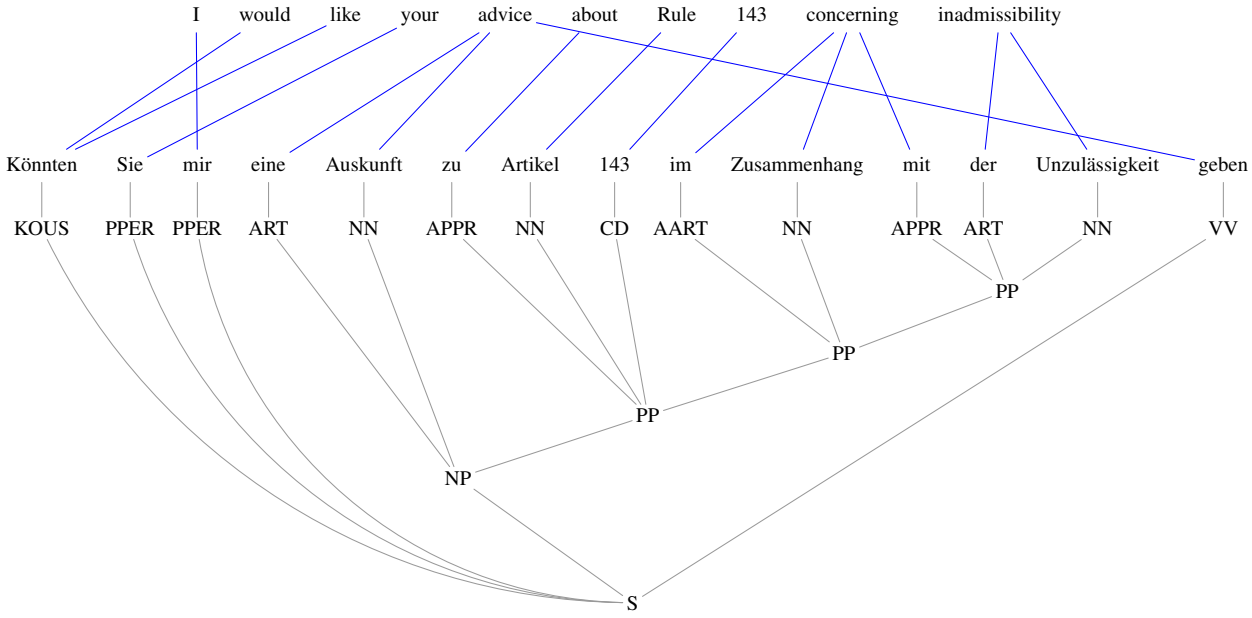


Figure 9: Source string and translation with annotated syntax tree, which are linked via the word alignment.

the constraint that the same meaning should be conveyed by the sentences. Most current statistical machine translation systems [66] again use supervised training and thus annotated training data to derive the rules used in the derivation process in addition to maximum likelihood estimates as rule weights.[11] Most modern translation systems are trained on a word-aligned parallel corpus, which is a resource that contains sentence-by-sentence translations in the two languages in question together with a bipartite relation on the sentence positions, called the word alignment [67]. Roughly speaking, the word alignment relates the words of the sentences and ideally indicates the translation on the word level. A typical training example for a system that uses syntactic annotations on the target side is provided in Figure 9. Synchronous grammars [68] are often used as models that perform the translation. The rules extracted from the training data typically consist of segments (substrings in case of strings and subtrees in the case of trees) of the annotated training example. The number of extracted rules is often extremely large and modern systems have several hundred million rules, so adding even more "hidden" information (e.g., by state splitting) seems difficult. Although finite-state information was successfully used in many different areas of natural language processing, it remains open whether similar approaches can be made tractable for statistical machine translation. The finite-state information could either be automatically derived as in the BERKELEY parser or imported from processes that perform annotations (e.g., the parser annotating the syntax tree or the aligner yielding the word alignment). Currently, finite-state constructions are used to translate weighted tree languages representing all or a selection of the likely parses of an input sentence instead of just the best syntax tree [69] or similarly extract rules from weighted tree languages annotated to an input sentence in the training data [70]. Whether automatically derived finite-state information can be used successfully also these highly complex systems with millions of surprisingly simple individual rules remains a key challenge, but recent successes [71] with deep neural networks seem to indicate that hidden finite-state information only waits to be discovered.

References

[1] C. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, MIT Press, 1999.

[11] Modern systems use additional features besides the pure translation weights to encourage fluent translations and offset certain particularities of the translation model.

[2] D. Jurafsky, J. H. Martin, Speech and Language Processing, 2nd Edition, Prentice Hall, 2008.
[3] U. Hebisch, H. J. Weinert, Semirings — Algebraic Theory and Applications in Computer Science, World Scientific, 1998.
[4] J. S. Golan, Semirings and their Applications, Kluwer Academic, Dordrecht, 1999.
[5] S. Yu, Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 1, Springer, 1997, Ch. II, pp. 41–110.
[6] M. P. Schützenberger, On the definition of a family of automata, Inform. and Control 4 (2–3) (1961) 245–270.
[7] M. Droste, W. Kuich, H. Vogler (Eds.), Handbook of Weighted Automata, EATCS Monographs on Theoret. Comput. Sci., Springer, 2009.
[8] K. Knight, J. May, Applications of weighted automata in natural language processing, in: Droste et al. [7], Ch. XIV, pp. 571–596.
[9] J. Sakarovitch, Rational and recognisable power series, in: Droste et al. [7], Ch. IV, pp. 105–174.
[10] N. Wirth, Compiler Construction, Addison Wesley, 1996.
[11] N. Xue, Chinese word segmentation as character tagging, Computational Linguistics and Chinese Language Processing 8 (1) (2003) 29–48.
[12] ISO Technical Committee 46, Romanization of Chinese, Tech. Rep. ISO 7098:1991, International Organization for Standardization (1991).
[13] F. Xia, The segmentation guidelines for the Penn Chinese treebank (3.0), Tech. Rep. IRCS 00-06, University of Pennsylvania (2000).
[14] N. Xue, F. Xia, F.-D. Chiou, M. Palmer, The Penn Chinese treebank: Phrase structure annotation of a large corpus, J. Natur. Lang. Engrg. 11 (2) (2002) 207–238.
[15] R. Sproat, C. Shih, W. Gale, N. Chang, A stochastic finite-state word-segmentation algorithm for Chinese, Comput. Linguist. 22 (3) (1996) 377–404.
[16] R. Sproat, Lexical analysis, in: R. Dale, H. Moisl, H. Somers (Eds.), Handbook of Natural Language Processing, 2nd Edition, Marcel Dekker Inc., 2000, Ch. 3, pp. 37–57.
[17] E. W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1) (1959) 269–271.
[18] H. Zhao, C. Kit, Unsupervised segmentation helps supervised learning of character tagging for word segmentation and named entity recognition, in: Proc. SIGHAN Workshop on Chinese Language Processing, Association for Computational Linguistics, 2008, pp. 106–111.
[19] R. Navigli, P. Velardi, Learning domain ontologies from document warehouses and dedicated web sites, Comput. Linguist. 30 (2) (2004) 151–179.
[20] B. Pang, L. Lee, S. Vaithyanathan, Thumbs up? sentiment classification using machine learning techniques, in: Proc. EMNLP, Association for Computational Linguistics, 2002, pp. 79–86.
[21] S. J. DeRose, Grammatical category disambiguation by statistical optimization, Comput. Linguist. 14 (1) (1988) 31–39.
[22] D. A. Borgmann, Beyond Language — Adventures in Word and Thought, Charles Scribner's Sons, 1967.
[23] M. P. Marcus, B. Santorini, M. A. Marcinkiewicz, Building a large annotated corpus of English — the Penn treebank, Comput. Linguist. 19 (2) (1993) 313–330.
[24] A. A. Markov, An example of statistical investigation of the text of 'Eugene Onegin' concerning the connection of samples in chains, in: Proc. Royal Academy of Sciences, St. Petersburg, Vol. VI of Ser. 7, 1913, pp. 153–162.
[25] I. J. Myung, Tutorial on maximum likelihood estimation, J. Math. Psych. 47 (2003) 90–100.
[26] A. J. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Trans. Inform. Theory 13 (2) (1967) 260–269.
[27] L. E. Baum, T. Petrie, Statistical inference for probabilistic functions of finite state markov chains, Ann. Math. Stat. 37 (6) (1966) 1554–1563.
[28] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[29] V. Černý, Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm, J. Optim. Theory Appl. 45 (1) (1985) 41–51.
[30] S. Ravi, K. Knight, Minimized models for unsupervised part-of-speech tagging, in: Proc. ACL, Association for Computational Linguistics, 2009, pp. 504–512.
[31] D. Garrette, J. Baldridge, Learning a part-of-speech tagger from two hours of annotation, in: Proc. NAACL, Association for Computational Linguistics, 2013, pp. 138–147.
[32] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, J. R. Stat. Soc. Ser. B Stat. Methodol. 39 (1) (1977) 1–38.
[33] L. E. Baum, T. Petrie, G. Soules, N. Weiss, A maximization technique occurring in the statictical analysis of probabilistic functions of Markov chains, Ann. Math. Stat. 41 (1) (1970) 164–171.
[34] J. Baker, The DRAGON system — an overview, IEEE Trans. Acoust. Speech Signal Process. 23 (1) (1975) 24–29.
[35] C. Nicolai, Solving ion channel kinetics with the QuB software, Biophys. Rev. Letters 8 (03n04) (2013) 191–211.
[36] M. J. Bishop, E. A. Thompson, Maximum likelihood alignment of DNA sequences, J. Mol. Biol. 190 (2) (1986) 159–165.
[37] J. Stigler, F. Ziegler, A. Gieseke, J. C. M. Gebhardt, M. Rief, The complex folding network of single calmodulin molecules, Science 334 (6055) (2011) 512–516.
[38] J. Lafferty, A. McCallum, F. C. N. Pereira, Conditional random fields — probabilistic models for segmenting and labeling sequence data, in: Proc. ICML, 2001, pp. 282–289.
[39] N. A. Smith, M. Johnson, Weighted and probabilistic context-free grammars are equally expressive, Comput. Linguist. 33 (4) (2007) 477–491.
[40] A. Salomaa, Probabilistic and weighted grammars, Inform. and Control 15 (6) (1969) 529–544.
[41] J. Berstel, C. Reutenauer, Recognizable formal power series on trees, Theoret. Comput. Sci. 18 (1982) 115–148.
[42] Z. Fülöp, H. Vogler, Weighted tree automata and tree transducers, in: Droste et al. [7], Ch. 9, pp. 313–403.
[43] Y. Wilks, E. Charniak, Computational Semantics — An Introduction to Artificial Intelligence and Natural Language Understanding, North-Holland, 1976.
[44] P. Blackburn, J. Bos, Representation and Inference for Natural Language — A First Course in Computational Semantics, CSLI publications, 2005.
[45] A. Haghighi, D. Klein, Coreference resolution in a modular, entity-centered model, in: Proc. NAACL, Association for Computational Linguistics, 2010, pp. 385–393.
[46] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, A. Tamchyna, Findings of the 2014 workshop on statistical machine translation, in: Proc. WMT, Association for Computational Linguistics,

2014, pp. 12–58.
[47] A. Carnie, Constituent Structure, Oxford University, 2010.
[48] L. Tesnière, Elements of structural syntax, John Benjamins, 2015.
[49] E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, T. Strzalkowski, A procedure for quantitatively comparing the syntactic coverage of English grammars, in: Proc. HLT, Morgan-Kaufmann, 1991, pp. 306–311.
[50] J. Carroll, E. Briscoe, A. Sanfilippo, Parser evaluation: a survey and a new proposal, in: Proc. LREC, 1998, pp. 447–454.
[51] G. Sampson, A. Babarczy, A test of the leaf-ancestor metric for parse accuracy, J. Natur. Lang. Engrg. 9 (4) (2003) 365–380.
[52] H. Schmid, Efficient parsing of highly ambiguous context-free grammars with bit vectors, in: Proc. CoLing, Association for Computational Linguistics, 2004, pp. 162–168.
[53] H. Schmid, Trace prediction and recovery with unlexicalized pcfgs and slash features, in: Proc. ACL, Association for Computational Linguistics, 2006, pp. 177–184.
[54] H. Shindo, Y. Miyao, A. Fujino, M. Nagata, Bayesian symbol-refined tree substitution grammars for syntactic parsing, in: Proc. ACL, Association for Computational Linguistics, 2012, pp. 440–448.
[55] D. Klein, C. D. Manning, Accurate unlexicalized parsing, in: Proc. ACL, Association for Computational Linguistics, 2003, pp. 423–430.
[56] S. Sekine, Evalb — bracket scoring program, online at: `http://nlp.cs.nyu.edu/evalb/` (2013).
[57] B. Bohnet, J. Nivre, A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing, in: Proc. EMNLP, Association for Computational Linguistics, 2012, pp. 1455–1465.
[58] J. Henderson, Discriminative training of a neural network statistical parser, in: Proc. ACL, Association for Computational Linguistics, 2004, pp. 95–102.
[59] T. Matsuzaki, Y. Miyao, J. Tsujii, Probabilistic CFG with latent annotations, in: Proc. ACL, Association for Computational Linguistics, 2005, pp. 75–82.
[60] D. Prescher, Inducing head-driven PCFGs with latent heads — refining a tree-bank grammar for parsing, in: Proc. ECML, Association for Computational Linguistics, 2005, pp. 292–304.
[61] S. Petrov, L. Barrett, R. Thibaux, D. Klein, Learning accurate, compact, and interpretable tree annotation, in: Proc. ACL, Association for Computational Linguistics, 2006, pp. 433–440.
[62] S. Petrov, Coarse-to-fine natural language processing, Ph.D. thesis, University of California at Bekeley, Berkeley, CA, USA (2009).
[63] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, M. Mohri, Openfst: A general and efficient weighted finite-state transducer library, in: Proc. CIAA, Vol. 4783 of LNCS, Springer, 2007, pp. 11–23.
[64] J. May, K. Knight, Tiburon: A weighted tree automata toolkit, in: Proc. CIAA, Vol. 4094 of LNCS, Springer, 2006, pp. 102–113.
[65] M. Mohri, Finite-state transducers in language and speech processing, Comput. Linguist. 23 (2) (1997) 269–311.
[66] P. Koehn, Statistical Machine Translation, Cambridge University Press, 2010.
[67] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, R. L. Mercer, The mathematics of statistical machine translation — parameter estimation, Comput. Linguist. 19 (2) (1993) 263–311.
[68] D. Chiang, An introduction to synchronous grammars, Tech. rep., ISI, University of Southern California, part of a tutorial given with Kevin Knight at ACL (2006).
[69] H. Mi, L. Huang, Q. Liu, Forest-based translation, in: Proc. ACL, Association for Computational Linguistics, 2008, pp. 192–199.
[70] H. Mi, L. Huang, Forest-based translation rule extraction, in: Proc. EMNLP, Association for Computational Linguistics, 2008, pp. 206–214.
[71] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Proc. EMNLP, Association for Computational Linguistics, 2014, pp. 1724–1734.