Approximating Annotated Corpora with Finite-State Transductions: A Case Study in Part of Speech Tagging

Emmanuel Roche, Yves Schabes

TR95-10   December 1995

Abstract

There is a natural correspondence between annotated corpora and functions: a corpus can be seen as a collection of points and their images by a function that maps the raw input to the annotated output. We illustrate this point by considering a corpus of sentences annotated with their part-of-speech. We then show that the construction of a part-of-speech disambiguator from a training corpus is equivalent to approximating the function corresponding to the corpus. A good approximation can be computed within the space of finite-state functions. The inferred function is capable of generalizing the disambiguation process to unknown text with state of the art accuracy. Moreover, the resulting function to linear-time implementation. In a companion paper, the method has also been successfully applied to letter-to-sound conversion.

Revisions history.

1. Version 1.0, 95/01/05

1 Introduction

Finite-state devices have important applications to many areas of computer science, including pattern matching, databases and compiler technology. Although their linguistic adequacy to natural language processing has been questioned in the past (Chomsky, 1964), there has recently been a dramatic renewal of interest in the application of finite-state devices to several aspects of natural language processing. This renewal of interest is due to the speed and the compactness of finite-state representations. This efficiency is explained by two properties: finite-state devices can be made deterministic, and they can be turned into a minimal form. Such representations have been successfully applied to different aspects of natural language processing, such as morphological analysis and generation (Karttunen, Kaplan, and Zaenen, 1992; Clemenceau, 1993), parsing (Roche, 1993; Tapanainen and Voutilainen, 1993), phonology (Laporte, 1993; Kaplan and Kay, 1994) and speech recognition (Pereira, Riley, and Sproat, 1994). Although finite-state machines have been used for part-of-speech tagging (Tapanainen and Voutilainen, 1993; Silberztein, 1993), none of these approaches has the same flexibility as stochastic techniques. Unlike stochastic approaches to part-of-speech tagging (Church, 1988; Kupiec, 1992; Cutting et al., 1992; Merialdo, 1990; DeRose, 1988; Weischedel et al., 1993), up to now the knowledge found in finite-state taggers has been handcrafted and cannot be automatically acquired.

Our work relies on two central notions: the notion of a finite-state transducer and the notion of a subsequential transducer. Informally speaking, a finite-state transducer is a finite-state automaton whose transitions are labeled by pairs of symbols. The first symbol is an input token and the second is an output token. Applying a finite-state transducer to an input string consists of following a path according to the input tokens while storing the output tokens, the result being the sequence of output tokens stored. Finite-state transducers can be composed, intersected, merged with the union operation, sometimes made deterministic and minimized[1]. Basically, one can manipulate finite-state transducers as easily as finite-state automata.[2]

In this paper, we investigate the correspondence between an annotated corpus and a function that can be represented as a finite-state transducer. A corpus can be seen as a collection of points and their images by a function that maps the raw input to the annotated output. We illustrate this point by considering a corpus of sentences annotated with their part-of-speech. We then show that the construction of a part-of-speech disambiguator from a training corpus is equivalent to approximating the function corresponding to the corpus. This is achieved by recording and combining contiguous chunks of annotations within a finite-state transducer. The method result to a good approximation with finite-state functions. The inferred function is capable of generalizing to disambiguate unknown text with state of the art accuracy. Moreover, the resulting function to linear-time implementation. The method clarify the limits encountered by approaches that ignore sentence structures such as n-grams and hidden Markov models.

In a companion paper (Schabes and Roche, 1995), the method has also been successfully applied to letter-to-sound conversion and reached 98% letter accuracy and 84% word accuracy.

2 Viewing Corpora as Finite-State Transductions

We illustrate the correspondence between an annotated corpus and finite-state functions by considering the example of part-of-speech tagging. For experimental purposes, we use the

[1]See for example, Berstel (1979), Mohri (1994) and Roche and Schabes (1994).

[2]However, whereas every finite-state automaton is equivalent to some deterministic finite-state automaton, there are finite-state transducers that are not equivalent to any deterministic finite-state transducer. Transductions that can be computed by some deterministic finite-state transducer are called *subsequential functions*.

Brown Corpus (Francis and Kučera, 1982) (700,000 words and 37,000 sentences) in which each word is annotated with its correct part-of-speech. The following sentence

(1) The/at plan/nn does/doz not/* cover/vb doctor/nn bills/nns ./.

is for example found in the training corpus.[3] Note that the word "plan" which appears in the above sentence can also be verb, that "cover" and "bills" can also be verbs. The task of a part-of-speech tagger is to disambiguate each word in context.

In a first step, each word is tagged with its most likely part-of-speech tag, estimated by examining the tagged corpus, without regard to context. For example, assuming that *vb* is the most likely tag for the word "plan" *nn* for "cover" and "doctor", and "nns" for bills, the first step might assign the following part-of-speech tags:

(2) The/at plan/vb does/doz not/* cover/nn doctor/nn bills/nns ./.

Since this step does not use any contextual information, many words can be tagged incorrectly. In (2), the word "plan" is erroneously tagged as a verb and "cover" is erroneously tagged as noun.

This first step can be trivially seen as a finite-state transduction from words to parts-of-speech tags.[4]

The mapping from this first initial sequence to the correct sequence of part-of-speech tags can similarly be represented by a finite-state transducer representing the initial alignment. For example, if the corpus consists only of (1), and the initial tagging is the one given in (2), the following transducer represents the initial alignment between the guessed and the correct part-of-speech sequences:

```
  at/at  vb/nn  doz/doz  */*   nn/vb   nn/nn  nns/nns  ./.
o-->---o-->---o--->----o-->---o-->---o-->---o--->----o-->---o
```

However, this transducer is useless since it can only applied on the exact training corpus. We will show in the following section, how a general method for approximating a transducer can be applied on this problem.

3 Approximation of the Finite-State Transduction

We illustrate the methods by applying them to the problem of part-of-speech disambiguation, where the input and the output are sequences of tags as illustrated in the previous section.

The methods generalize the data found in the initial alignment corpus using all aligned substrings of different length of all initial alignment corpus. Substrings of n tags (*n-grams*) aligned with n tags are called *n-maps*. Given an aligned corpus, the set of *n-maps* are trivially computed.

For example, all *4-maps* for the above example are

[3]The notation for part-of-speech tags is adapted from the one used in the Brown Corpus (Francis and Kučera, 1982): *at* stands for determiner, *nn* for singular noun, *nns* for a plural noun, *doz* for the word "does", * for the word "not", *vb* for a verb in its base form, *vbn* for verb in past participle form, *bedz* for the word "was" and . for the end of sentence.

[4]The transducer has a single state, and each word is represented by a looping arc labeled by the word and its most likely tag.

```
at,vb,doz,*/at,nn,doz,*
vb,doz,*,nn/nn,doz,*,vb
doz,*,nn,nns/doz,*,vb,nns,
*,nn,nns,./*,vb,nns,.
```

For a given length n, the method builds a dictionary of *n-maps* that corresponds to the most likely transcription of each *n-gram*. For example, if the *2-map* `nn,nn/vb,nn` occurs five times in the dictionary and the *2-map* `nn,nn/nn,nn` occurs three times in the dictionary, the most likely *2-map* for the *bi-gram* `nn,nn` is `nn,nn/vb,nn` and only this one will be stored in the *2-map* dictionary.

The construction and the use of these dictionaries is now described.

The method is better illustrated by an example. Consider the following sample dictionary obtained by merging *1-map*, *2-map* and *3-map* dictionaries:[5]

```
at/at
vb/vb
vb,doz/nn,doz
,nn,nn/*,vb,nn
nns/nns
./.
```

Using the above dictionary, the part-of-speech sequence `at vb doz * nn nn nns .` is transcribed left to right as follows.

```
at   vb doz   * nn nn   nns   .
at   nn doz   * vb nn   nns   .
```

Looking at the first tag, the longest *n-map* defined in the dictionary that matches what follows is `at/at`. `at` is emitted and we move to the second tag (`vb`). At that point, two *n-maps* match, `vb/vb` and `vb,doz/nn,doz`. The longest one, `vb,doz/nn,doz`, is chosen and we move to the fourth tag while emitting `nn doz`. At that point, `*,nn,nn/*,vb,nn` is the longest match. And so on.

Note that in the above *n-phon* dictionary, the *2-map* `nns,./nns,.` does not need to be stored since it can be obtained from this dictionary (with `nns/nns` and `./.`) with the same procedure.

More precisely, the method builds a series of *n-map* dictionaries. The *1-map* dictionary is first built.[6] Dictionaries of increasing lengths of *n-maps* are inductively built. Assuming that all dictionaries up to a given length $n-1$ have been built, we build the dictionary for length n by removing from the set of most likely transcriptions of length n the ones that can correctly be derived from the dictionaries of length up to $n-1$.

In our example, the *two-map* `nns,./nns,.` is not stored since it can be derived from the *uni-map* `nns/nns` and `./.`.

We could carry this operation up to the length of the longest of the corpus, or, obviously, stop at a shorter length k.

So far, we have described the method in terms of dictionaries (*n-map*). The method however operates on finite-state transducers representing these dictionaries. Each *n-map* dictionary can

[5]In practice it is useful to add begin and end markers to each sentence. For sake of simplicity those markers are ignored in this paper.

[6]As previously said, this dictionary contains the most likely transcription of each tag.

n	# `n-maps`	# remaining `n-maps`
1	97	97
2	3867	759
3	37993	7291
4	137518	20217
5	269294	25268
6	365021	19594
7	406000	11709

Figure 1: Different levels of *n-maps* applied to the whole training corpus.

be trivially represented as a finite-state transducer where each arc is labeled with a pair of letter and phoneme.

We build inductively a single finite state transducer F_k that computes the part-of-speech alignment of an initial part-of-speech guess. The transcriptions according to the *one-map* dictionary is obviously a deterministic finite-state transducer. Inductively, assume that we have built the machine operating with the $1-$ to *n-map* dictionaries. Suppose also that the *n+1-map* dictionary is represented as deterministic finite-state transducer, then the algorithm described in Appendix A computes the combined deterministic transducer. This operation is repeated until the *n-maps* of length k are included.

By construction, the final deterministic finite-state transducer generalizes the function to any part-of-speech string. The constructed *n-map* dictionaries can be seen as transcription rules inferred from the dictionary. Therefore by combining the initial dictionary lookup and this approximation function, we can assign a part-of-speech sequence to any input sentence.

3.1 Experiments

We used 90% of the Brown Corpus for training purposes and used the remaining 10% for testing. Figure 1 shows the sizes of the `n-map` dictionaries obtained by the transducer F_n constructed for increasing values of n (as described in the previous Section). The second column gives the number of `n-maps` found in the training. The third column gives the number of `n-maps` remaining after considering the shorter `p-maps` ($p < n$). The third column shows the power of combining shorter `n-maps` to form cover large ones.

Figure 2 shows the performance in term of percentage of correctly tagged words obtained by the transducer on test sentences.

The experiments show that an optimal performance is obtained for length 3. State-of-the-art performance is achieved at that point(Church, 1988; Kupiec, 1992; Cutting et al., 1992; Merialdo, 1990; DeRose, 1988; Weischedel et al., 1993; Brill, 1992). After this point, over-training occurs. The method implicitly shows the limits encountered by approaches that ignore sentence structures such as n-grams and hidden Markov models. One can conclude that sentence structure analysis would be required if longer contexts are used.

Since the algorithm described in Appendix A builds a deterministic finite-state transducer, using an appropriate implementation for finite-state transducers (Roche, 1993; Roche and Schabes, 1994), the resulting part-of-speech tagger operates in linear time, independent of the number of rules and the length of the context. The new tagger therefore operates in optimal time in the sense that the time to assign tags to a sentence corresponds to the time required to follow a single path in a deterministic finite-state machine. This very method yields a different method for tagging with deterministic finite-state transducer than the one suggested in

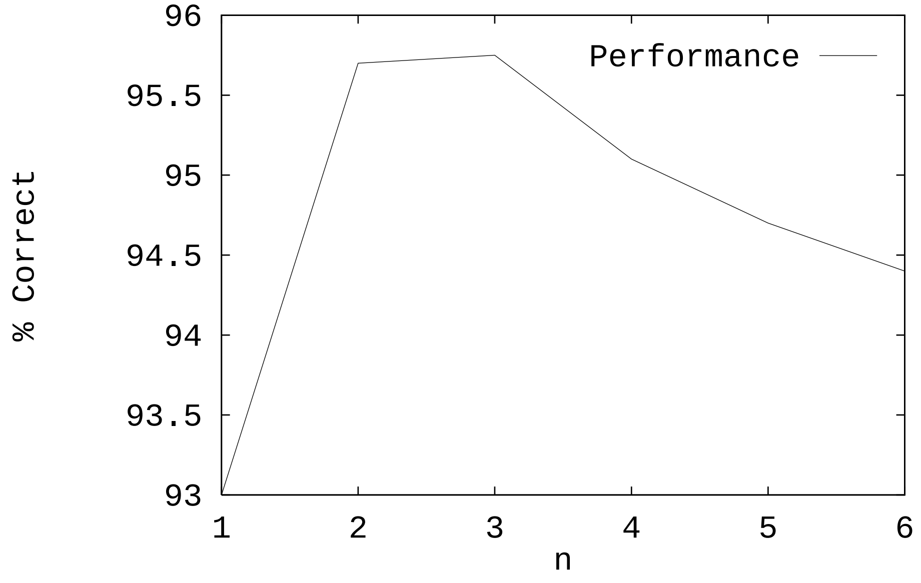

Figure 2: Performance Test Set

(Roche and Schabes, 1994) and is applicable to many other problems, such as letter-to-sound conversion (Schabes and Roche, 1995).

4 Conclusion

There is a natural correspondence between annotated corpora and functions: a corpus can be seen as a collection of points and their images of a function that maps the raw input to the annotated output. We have shown that finite-state transducers are very well suited for data-driven methods.

As a case study, we experimented with part-of-speech disambiguation using a 700,000 annotated corpus of English text. We have presented a method for building a deterministic finite-state transducer which is capable to tag any sentence with state-of-the-art performances. The resulting transducer is very compact compared to traditional stochastic approaches.

The method generalizes a finite-state function given value on a sample. We believe that they are useful for a wide range of data-oriented applications. In a companion paper (Schabes and Roche, 1995), the method has also been successfully applied to letter-to-sound conversion.

References

Berstel, Jean. 1979. *Transductions and Context-Free Languages.* Teubner, Stuttgart.

Brill, Eric. 1992. A simple rule-based part of speech tagger. In *Third Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy.

Chomsky, N. 1964. *Syntactic Structures.* Mouton and Co., The Hague.

Church, Kenneth Ward. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing*, Austin, Texas.

Clemenceau, David. 1993. *Structuration du Lexique et Reconnaissance de Mots Dérivés*. Ph.D. thesis, Université Paris 7.

Cutting, Doug, Julian Kupiec, Jan Pederson, and Penelope Sibun. 1992. A practical part-of-speech tagger. In *Third Conference on Applied Natural Language Processing*, pages 133–140, Trento, Italy.

DeRose, S.J. 1988. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14:31–39.

Francis, W. Nelson and Henry Kučera. 1982. *Frequency Analysis of English Usage*. Houghton Mifflin, Boston.

Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Karttunen, Lauri, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *Proceedings of the* 14th *International Conference on Computational Linguistics (COLING'92)*.

Kupiec, J. M. 1992. Robust part-of-speech tagging using a hidden markov model. *Computer Speech and Language*, 6:225–242.

Laporte, Eric. 1993. Phonétique et transducteurs. Technical report, Université Paris 7, June.

Merialdo, Bernard. 1990. Tagging text with a probabilistic model. Technical Report RC 15972, IBM Research Division.

Mohri, Mehryar. 1994. Compact representation by finite-state transducers. In 22nd *Meeting of the Association for Computational Linguistics (ACL'94)*, pages 204–209.

Pereira, Fernando C. N., Michael Riley, and Richard W. Sproat. 1994. Weighted rational transductions and their application to human language processing. In *ARPA Workshop on Human Language Technology*. Morgan Kaufmann.

Roche, Emmanuel and Yves Schabes. 1994. Deterministic part-of-speech tagging with finite-state transducers. Technical Report TR-94-07, Mitsubishi Electric Research Laboratories, Cambridge, USA, June.

Roche, Emmanuel. 1993. *Analyse Syntaxique Transformationelle du Français par Transducteurs et Lexique-Grammaire*. Ph.D. thesis, Université Paris 7, January.

Schabes, Yves and Emmanuel Roche. 1995. Exact generalization of finite-state transductions: Application to grapheme-to-phoneme transcription. Submitted to the 23rd Meeting of the Association for Computational Linguistics (ACL'95).

Silberztein, Max. 1993. *Dictionnaires électroniques et analyse lexicale du français— Le système INTEX*. Masson.

Tapanainen, Pasi and Atro Voutilainen. 1993. Ambiguity resolution in a reductionistic parser. In *Sixth Conference of the European Chapter of the ACL, Proceedings of the Conference*, Utrecht, April.

Weischedel, Ralph, Marie Meteer, Richard Schwartz, Lance Ramshaw, and Jeff Palmucci. 1993. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2):359–382, June.

A Combine Operation

In the following, a finite-state transducer T is formally defined as a 5-tuple (Σ, Q, i, F, E) where: Σ is a finite alphabet; Q is the set of states or vertices; $i \in Q$ is the initial state; $F \subseteq Q$ is the set of final states; $E \subseteq Q \times \Sigma \cup \{\epsilon\} \times \Sigma^* \times Q$ is the set of edges or transitions.

This leads to the definition of *deterministic* transducers called *subsequential* transducers: a subsequential transducer T is a 6-tuple $(\Sigma, Q, i, F, E, \rho)$ where: Σ, Q, i, F, E are defined as above, but E is such that $\forall q \in Q, \forall a \in \Sigma, |\{(q, a, b, q') \in E\}| \leq 1$; and the final emission function ρ maps F on Σ^*, one writes $\rho(q) = w$.

In addition, it is useful to define the *state transition function* d by $d(q, a) = q'$ s.t. $\exists (q, a, b, q') \in E$; and the *emission function* δ by $\delta(q, a, q') = b$ if $(q, a, b, q') \in E$.

For $w_1, w_2 \in \Sigma^*$, $w_1 \wedge w_2$ denotes the longest common prefix of w_1 and w_2.

The following algorithm combines two transducers $T_1 = (\Sigma, Q_1, i_1, F_1, E_1, \rho_1)$ and $T_2 = (\Sigma, Q_2, i_2, F_2, E_2, \rho_2)$ to form a third transducer $T = (\Sigma, Q, i, F, E, \rho)$ according to the method described in Section 3.

```
1. COMBINE_TRANSDUCERS($T_1 = (\Sigma, Q_1, i_1, F_1, E_1, \rho_1)$,$T_2 = (\Sigma, Q_2, i_2, F_2, E_2, \rho_2)$)
2. $q = 0; n = 1; i = 0; C[0] = ((0, \epsilon), (0, \epsilon)); F = \{0\}; \rho(0) = \epsilon; E = \emptyset;$
3.     do {
4.         $((x_1, u_1), (x_2, u_2)) = C[q];$
5.         if $(x_1 \in F_1)$
6.             $F = F \cup \{q\}$; $\rho(q) = u_1 \cdot \rho_1(x_1);$
7.         for each $a$ s.t. $d_1(x_1, a) \neq \emptyset$
8.             CASE 1 : $x_2 \neq$ OUT
9.                 CASE 1.1 : $d_2(x_2, a) \neq \emptyset$
10.                    $y_1 = d_1(x_1, a); y_2 = d_2(x_2, a);$
11.                    CASE 1.1.1 : $y_2 \notin F_2$
12.                        $\mu_1 = \delta_1(x_1, a, y_1); \mu_2 = \delta_2(x_2, a, y_2);$
13.                        $b = u_1 \cdot \mu_1 \wedge u_2 \cdot \mu_2;$
14.                        $S' = ((y_1, b^{-1} \cdot u_1 \cdot \mu_1), (y_2, b^{-1} \cdot u_2 \cdot \mu_2))$
15.                        if $\exists r \in [0, n-1]$ s.t. $C[r] == S'$
16.                            $e = r;$
17.                        else
18.                            $C[e = n++] = S';$
19.                        $E = E \cup \{(q, a, b, e)\};$
20.                    CASE 1.1.2 : $y_2 \in F_2$
21.                        $b = u_2 \cdot \delta_2(x_2, a, y_2) \cdot \rho_2(y_2);$
22.                        $S' = ((0, \epsilon), (0, \epsilon));$
23.                        if $\exists r \in [0, n-1]$ s.t. $C[r] == S'$
24.                            $e = r;$
25.                        else
26.                            $C[e = n++] = S';$
27.                        $E = E \cup \{(q, a, b, e)\};$
28.                CASE 1.2 : $d_2(x_2, a) = \emptyset$
29.                        Call AUX;
30.            CASE 2 : $x_2 =$ OUT
31.                Call AUX;
32.        $q++;$
33.    } while $(q < n)$;
34. Function AUX
35.    $y_1 = d_1(x_1, a); b = u_1 \cdot \delta_1(x_1, a, y_1);$
36.    if $y_1 = 0$ then $y_2 = 0$ else $y_2 =$ OUT;
37.    $S' = ((y_1, \epsilon), (y_2, \epsilon));$
38.    if $\exists r \in [0, n-1]$ s.t.$C[r] == S'$
39.        $e = r;$
40.    else
41.        $C[e = n++] = S';$
41.    $E = E \cup \{(q, a, b, e)\};$
```