A Hybrid Approach to Part-of-Speech Tagging

Bryan Jurish
jurish@ling.uni-potsdam.de

Final Report
Berlin-Brandenburgische Akademie der Wissenschaften
Project "Kollokationen im Wörterbuch"[1]
December 2003

Abstract

Part-of-Speech (PoS) Tagging – the automatic annotation of lexical categories – is a widely used early stage of linguistic text analysis. One approach, *rule-based morphological anaylsis*, employs linguistic knowledge in the form of hand-coded rules to derive a set of possible analyses for each input token, but is known to produce highly ambiguous results. Stochastic tagging techniques such as *Hidden Markov Models (HMMs)* make use of both lexical and bigram probabilities estimated from a tagged training corpus in order to compute the most likely PoS tag sequence for each input sentence, but provide no allowance for prior linguistic knowledge. In this report, I describe the `dwdst`[2] PoS tagging library, which makes use of a rule-based morphological component to extend traditional HMM techniques by the inclusion of *lexical class probabilities* and theoretically motivated *search space reduction*.

Contents

[1]Funding for this project was provided by the Alexander von Humboldt Stiftung and by the Zukunftsinvenstionsprogramm of the German federal government.

[2]Pronounced as "dude's tea".

List of Tables

List of Figures

1 Introduction

Part-of-speech tagging is used as an early stage of linguistic text analysis in many applications, including subcategorization acquisition (Manning, 1993; Ushioda et al., 1993), text-to-speech synthesis (Dutoit, 1997; Allen et al., 1987), and corpus indexing (Geyken et al., 2002). Two prominent distinct approaches to be found in previous work are *rule-based morphological analysis* (Allen et al., 1987; Hanneforth, 2002b) on the one hand, and *stochastic models* such as Hidden Markov Models (HMMs) (Church, 1988; DeRose, 1988; Cutting et al., 1992) on the other.

Rule-based morphological analyzers rely on hand-crafted rules to decompose input tokens into their morphological components, computing the resultant lexical category as a function of those components. Such systems incorporate the linguistic competence of their human authors, to the extent that such competence can be and is expressed in the systems' rule sets. Unfortunately, the construction of a hand-crafted rule set for unrestricted input tokens of a given language is a time-consuming and labor-intensive task. Another common problem for token-wise rule-based approaches is that of ambiguity – in order to determine which of multiple possible analyses for a single token is the correct one, some reference to the context in which that token occurs is usually required.

Stochastic tagging techniques such as Hidden Markov Models rely on both lexical and bigram probabilities estimated from a tagged training corpus in order to compute the most likely PoS tag sequence for each sequence of input tokens. The existence of hand-tagged training corpora for many languages and the robustness of the resulting models have made stochastic taggers quite popular. Disadvantages for HMM taggers include the large amount of training data required to achieve high levels of accuracy, as well as the fact that no clear allowance is made in traditional HMM tagging architectures for prior linguistic knowledge.

Some previous work has focussed on integrating rule-based linguistically motivated morphological analysis with robust context-dependent disambiguation, notably work by Brill (1992, 1994) and Brants (2000). In my opinion, neither of these approaches represents an optimal integration of prior linguistic knowledge with robust context-dependent tagging techniques.

Brill's tagger uses a two-stage architecture, initially tagging input tokens with their most likely tag (disregarding context), employing an automatically acquired set of lexical rules to identify unknown words by relating them to words found in the training corpus based on the presence (or absence) of affixes (suffixes or prefixes) of some prespecified maximal length.[3] In the second stage, a set of automatically acquired context-dependent transformations are applied to alter initial tag assignments based on contextual features. While human editing of the lexical rule set could arguably provide a place for prior linguistic knowledge, the hard limit on affix lengths imposed by Brill's design makes his system

[3]For Brill (1994), the maximum allowed affix length is 4.

impractical for languages such as German which exhibit highly productive noun compounding phenomena.

Brants' stochastic HMM tagger TnT uses a suffix analysis technique attributed to Samuelsson (1993) to estimate lexical probabilities for unknown tokens based on properties of words in the training corpus which share the same suffixes. As in Brill's approach, the length of suffixes considered is constrained by a hard limit,[4] and the string suffixes considered need not correspond to any linguistically meaningful units.

The approach taken in the `dwdst` PoS tagging library developed at the Berlin-Brandenburgische Akademie der Wissenschaften attempts to integrate a body of prior linguistic knowledge realized as a finite-state morphological analyzer (Hanneforth, 2002b) with robust Hidden Markov Model tagging techniques by modifications both to the underlying model parameters and to the *Viterbi algorithm* (Viterbi, 1967), which is used by HMM taggers to compute the most likely tag sequence for a sequence of input tokens.

2 Formal Background

In this Section, I present some formal prerequisites for an adequate formal description of the `dwdst` system.

2.1 Text Data

Definition 2.1 (Input Alphabet Σ)
Σ is a finite set of characters, the *input alphabet*, and Σ^* is the set of all strings over Σ, including the *empty word*, ε.

Definition 2.2 (Separators)
$\oplus \in \Sigma$ is a conventional *token separator*. $\odot \in \Sigma$ is a conventional *sentence separator*.

Informally, separators can be seen as conventions which allow the system to identify and access linguistically meaningful boundaries in an input text.

Definition 2.3 (Token Alphabet Σ_T)
$\Sigma_T = \Sigma - \{\oplus, \odot\}$ is a finite set of characters, the *token alphabet*. Separators are explicitly excluded from the token alphabet to ensure that they are used solely as delimiters.

Definition 2.4 (Tokens $Toks$)
$Toks$ is a set of delimited token strings: $Toks \subseteq \Sigma_T^*\{\oplus\}$.

[4] It should be noted that Brants' system allows the user to specify his or her own suffix-length limit. By default, the maximum suffix length considered by TnT is 10 characters.

Definition 2.5 (Sentences *Sents*)
Sents is a set of delimited sentences: $Sents \subseteq Toks^*\{\odot\}$.

Without loss of generality, I will write $s = w_1, \ldots, w_n \in Sents$, or simply $s = w_{1..n} \in Sents$ as shorthand for the delimited sentence notation $s = w_1 \oplus \cdots w_n \oplus \odot \in Sents$, for token-strings $w_1, \ldots, w_n \in \Sigma_T^*$. When delimiters are critical, I will use the notation $s = \langle w_1, \ldots, w_n \rangle$.

Definition 2.6 (Analysis Alphabet)
Σ_A is a finite *analysis alphabet*. In practice, it is useful to define an interface level in which $\Sigma_A = \Sigma_T$, but this is not strictly necessary.

Definition 2.7 (Tag Alphabet)
$Tags \subset \Sigma_A^*$ is a set of *part-of-speech tags*. I follow tradition in requiring that that the tagset $Tags$ be finite.

2.2 Finite State Devices

This Section lays out some basic definitions concerning finite state devices. For more complete discussions, see Aho and Ullman (1972) and Roche and Schabes (1997).

2.2.1 Finite State Automata

Definition 2.8 (Finite State Automaton (FSA))
A *finite state automaton* is a five-tuple $A = (\Sigma, Q, q_0, \delta, F)$ where:

- Σ is a finite *alphabet*;
- Q is a finite set of automaton *states*;
- $q_0 \in Q$ is the distinguished *initial state*;
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is a *transition function*; and
- $F \subseteq Q$ is a set of *final states*.

I will write $\mathcal{FSA}_\Sigma$ to refer to the collection of all finite state automata over the alphabet Σ.

Definition 2.9 (Extended Transition Function δ^*)
Given a finite state automaton $A = (\Sigma, Q, q_0, \delta, F)$, the *extended transition function* $\delta^* : Q \times \Sigma^* \to 2^Q$ is defined as follows for all $q \in Q$, for all $w \in \Sigma^*$, and for all $a \in \Sigma \cup \{\varepsilon\}$:

$$\begin{aligned} \delta^*(q, \varepsilon) &= \{q\} \\ \delta^*(q, wa) &= \bigcup_{q' \in \delta^*(q,w)} \delta(q, a) \end{aligned}$$

Definition 2.10 (Automaton Language)
The *language recognized by* a finite state automaton $A = (\Sigma, Q, q_0, \delta, F)$ is written $\mathcal{L}(A)$, and defined:

$$\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta^*(w) \cap F \neq \emptyset\}$$

A string language L is said to be a *regular language* just in case there is a finite state automaton A such that $L = \mathcal{L}(A)$.

Definition 2.11 (Automaton Constructor *Rec*)
From a finite alphabet Σ and a string $w \in \Sigma^*$, an automaton $Rec(w) \in \mathcal{FSA}_\Sigma$ can be constructed such that $L(Rec(w)) = \{w\}$. The existence of such an automaton follows directly from the closure of regular languages with respect to concatenation. The construction itself is quite simple to define for $w = a_1 \cdots a_n$, where $a_i \in \Sigma$ for $1 \leq i \leq n$:

$$Rec(w) = (\Sigma, \bigcup_{i=0}^{n} \{i\}, 0, \bigcup_{i=0}^{n-1} \{(i, a_i, \{i+1\})\}, \{n\})$$

2.2.2 Finite State Transducers

Definition 2.12 (Finite State Transducer (FST))
A *finite state transducer* is a six-tuple $T = (\Sigma_1, \Sigma_2, Q, q_0, \delta, F)$ where:

- Σ_1 is a finite *input alphabet*;
- Σ_2 is a finite *output alphabet*;
- Q is a finite set of *states*;
- $q_0 \in Q$ is the distinguished *initial state*;
- $\delta : Q \times (\Sigma_1 \cup \{\varepsilon\}) \to 2^{(Q \times (\Sigma_2 \cup \{\varepsilon\}))}$ is a *transition function*; and
- $F \subseteq Q$ is a set of *final states*.

I will write $\mathcal{FST}_{\Sigma_1,\Sigma_2}$ to refer to the collection of all finite state transducers $(\Sigma_1, \Sigma_2, Q, q_0, \delta, F)$ with input and output alphabets Σ_1 and Σ_2, respectively.

Definition 2.13 (Underlying Automaton)
The *underlying automaton* for a finite state transducer $T = (\Sigma_1, \Sigma_2, Q, q_0, \delta, F)$ is a finite state automaton $\mathcal{A}(T) = (\Sigma_1 \times \Sigma_2, Q, q_0, \delta', F)$ where for all $q_1, q_2 \in Q$, for all $a_1 \in \Sigma_1$ and all $a_2 \in \Sigma_2$:

$$\delta'(q_1, (a_1, a_2)) = \{q_2 \mid (q_2, a_2) \in \delta(q_1, a_1)\}$$

Definition 2.14 (Transducer Relation)
The *relation realized by* a finite state transducer $T = (\Sigma_1, \Sigma_2, Q, q_0, \delta, F)$ is written $\mathcal{R}(T)$, and defined as a binary relation over Σ_1 and Σ_2 by reference to the underlying automaton $\mathcal{A}(T)$ as: $\mathcal{R}(\mathrm{T}) = \mathcal{L}(\mathcal{A}(\mathrm{T}))$.

Similarly, T may be seen to induce a mapping $|T| : \Sigma_1^* \to 2^{\Sigma_2^*}$ from input strings to sets of output strings by defining for all $w_1 \in \Sigma_1^*$:

$$|T|(w_1) = \{w_2 \in \Sigma_2^* \mid (w_1, w_2) \in \mathcal{R}(T)\}$$

Definition 2.15 (Identity Transducer)
A finite state transducer $T \in \mathcal{FST}_{\Sigma,\Sigma}$ is called an *identity transducer* for the finite state automaton $A \in \mathcal{FSA}_\Sigma$ just in case $\mathcal{R}(T) = \{(w, w) \mid w \in \mathcal{L}(A)\}$. Given the automaton $A = (\Sigma, Q, q_0, \delta, F)$, an identity transducer $Id(A)$ may be constructed by setting $Id(A) = (\Sigma, \Sigma, Q, q_0, \delta', F)$, where for all $q_1 \in Q$ and for all $a \in \Sigma \cup \{\varepsilon\}$,

$$\delta'(q_1, a) = \bigcup_{q_2 \in \delta(q_1, a)} \{(q_2, a)\}$$

Definition 2.16 (Transducer Projections)
For a finite state transducer $T = (\Sigma_1, \Sigma_2, Q, q_0, \delta, F)$, $Proj_1(T)$ and $Proj_2(T)$ are finite state automata, the *first* and *second projections* of T, respectively, defined as:

$$\begin{aligned} Proj_1(T) &= (\Sigma_1, Q, q_0, \delta_1, F) \\ Proj_2(T) &= (\Sigma_2, Q, q_0, \delta_2, F) \end{aligned}$$

where for all $q \in Q$, for all $a_1 \in (\Sigma_1 \cup \{\varepsilon\})$, and for all $a_2 \in (\Sigma_2 \cup \{\varepsilon\})$:

$$\begin{aligned} \delta_1(q, a_1) &= \{q' \mid \exists a_2' \in (\Sigma_2 \cup \{\varepsilon\}).(q\prime, a_2') \in \delta(q, a_1)\} \\ \delta_2(q, a_2) &= \{q' \mid \exists a_1' \in (\Sigma_1 \cup \{\varepsilon\}).(q\prime, a_2) \in \delta(q, a_1')\} \end{aligned}$$

2.3 Hidden Markov Models

This Section lays out some basic definitions concerning discrete Hidden-Markov Models. For more complete discussions, see Manning and Schütze (1999), Charniak (1993), or Rabiner (1989). Stochastic part-of-speech taggers can be described briefly as those taggers which compute the tagging function $\tau : Toks^n \to Tags^n$ for finite $n \in \mathbb{N}$ as:

$$\tau(w_{1..n}) = \arg \max_{t_{1..n} \in Tags^n} P(t_{1..n}|w_{1..n}) \quad (1)$$

where:

$$P(w_{1..n}, t_{1..n}) = \prod_{i=1}^{n} P(t_i|w_{1..i-1}, t_{1..i-1}) P(w_i|w_{1..i-1}, t_{1..i}) \quad (2)$$

Definition 2.17 (Hidden Markov Model)
A *Hidden Markov Model (HMM)* is a five-tuple $H = (\Sigma, Q, q_0, A, B)$ where:

- Σ is a finite *observation alphabet*;
- Q is a finite set of *states*;
- $q_0 \in Q$ is the distinguished *initial state*;
- $A : Q \times Q \rightarrow [0..1]$ is a probability distribution on state transitions: $A(q_1, q_2)$ is the probability of a transition to state q_2 from state q_1; and
- $B : Q \times \Sigma \rightarrow [0..1]$ is a probability distribution on state symbol emissions: $B(q, a)$ is the probability of observing the symbol a when in state q.

For a tagset $Tags$ and a finite set of input tokens $Toks$, it is customary to define a bigram HMM part-of-speech tagger $H = (Toks, Tags, \odot, A, B)$, where the probability functions A and B are estimated from a tagged training corpus. Under such a model, part-of-speech tags are represented as states of the model, and the task of finding the most likely tag sequence $t_{1..n} \in Tags^n$ for an input token sequence $w_{1..n} \in Toks^n$ can be formulated as a search for the most likely sequence of HMM states given the observation sequence $w_{1..n}$. Underlying such a technique are the following two Markov assumptions:

$$P(t_i|w_{1..i-1}, t_{1..i-1}) = P(t_i|t_{i-1}) \quad (3)$$

$$P(w_i|w_{1..i-1}, t_{1..i}) = P(w_i|t_{1..i}) \quad (4)$$

Under these assumptions, and with the addition of a special boundary tag t_0, computation of the most probable tag sequence $\tau(w_{1..n})$ for an input sequence $w_{1..n}$ can be simplified[5] to:

$$\tau(w_{1..n}) = \arg \max_{t_{1..n} \in Tags^n} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) \quad (5)$$

Definition 2.18 (Viterbi Algorithm)
The *Viterbi Algorithm* (Viterbi, 1967) is a dynamic programming method which efficiently computes for an HMM $H = (\Sigma, Q, q_0, A, B)$ and a given observation sequence $o_{1..n} \in \Sigma^n$ the state sequence $q_{1..n} \in Q^n$ most likely to generate $o_{1..n}$ according to the model parameters.

Formally, the Viterbi algorithm computes: $\arg\max_{q_{1..n}} P(o_{1..n}, q_{1..n})$ by incrementally computing variables $\delta_q(i)$, which store for each time increment i and each state q the probability of the most likely path resulting in state q at time i, as well as variables $\psi_q(i)$, which store for each time increment i and state q the most likely path leading to q at time i, which is completely specified by the most likely predecessor for q at time i:

[5]Here and elsewhere, I assume an "initial tag" t_0, use of which will be made explicit in Section 3.

$$\begin{aligned}\delta_q(i) &= \max_{q_{1..i-1}} P(o_{1..i-1}, q_{1..i-1}, q_i = q)\\ \psi_q(i) &= \arg_{q_{i-1}} \max_{q_{1..i-1}} P(o_{1..i-1}, q_{1..i-1}, q_i = q)\end{aligned}$$

The computation proceeds as follows:

1. **Initialize:**
$$\delta_q(0) = \begin{cases} 1 & \text{if } q = q_0 \\ 0 & \text{otherwise} \end{cases}$$
2. **Repeat:** for $1 \leq i \leq n$
$$\begin{aligned}\delta_q(i) &= \max_{q' \in Q} \delta_{q'}(i-1) A(q', q) B(q, o_i)\\ \psi_q(i) &= \arg\max_{q' \in Q} \delta_{q'}(i-1) A(q', q) B(q, o_i)\end{aligned}$$
3. **Terminate:**
$$\begin{aligned}q_n &= \arg\max_{q \in Q} \delta_q(n)\\ q_{i-1} &= \psi_{q_i}(i), \text{ for } n \geq i > 1\end{aligned}$$

3 The dwdst System

The dwdst part-of-speech tagger is composed of five runtime main modules: the *preprocessor*, the *morphological analyzer*, the *lexical classifier*, the *HMM disambiguator*, and the *analysis restrictor*. An additional module – the *HMM trainer* – is used to estimate model parameters from a tagged training corpus. A simplified graphical representation of the functional relations between these modules is given in Figure 1. In the remainder of this Section, I present some details on each of these modules in turn.

3.1 Preprocessor

The preprocessor reads and tokenizes a raw input text, performing basic end-of-sentence recognition and abbreviation expansion, using a set of hard-coded language-specific heuristics. For purposes of portability, the preprocessor is currently implemented as an independent process, dwdspp, whose formatted output may be piped to the tagger proper. Formally, the task of the preprocessor may be described as follows:

Definition 3.1 (Preprocessor *PreProc*)
The *preprocessor* is a function $PreProc : \Sigma^* \rightarrow Sents$ which maps raw input strings to delimited sentences – strings of delimited tokens.

The preprocessor dwdspp is implemented in C++ as a flex++ (Coëtmeur, 1993) lexical analyzer using an 8-bit input alphabet.[6] The newline character (char-

[6] By default, the ISO-8859-1 character set is used.

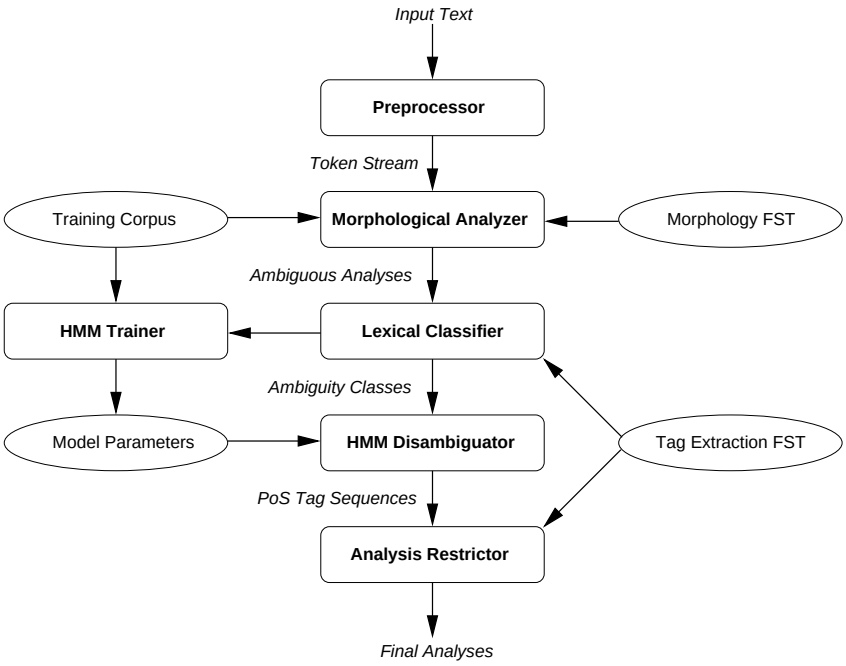

Figure 1: Simple functional diagram of the `dwdst` tagger

Input:	Linda *Linda*	Nakew *Nakew*	wird *will*	die *the*	Mannschaft *team*	verstärken *join*	. .
Morphological Analysis:	{NE.*first*, NE.*last*}	∅	{VAFIN.3*rd.sg.pres*, VVFIN.3*rd.sg.pres*, VVIMP.*sg*}	{ART.*sg.nom.fem*, ..., PDS.*nom.sg.fem*, ..., PRELS.*acc.pl*}	{NN.*masc.sg.nom*, ..., NN.*fem.sg.**}	{VVFIN.1*st.pl.pres*, ..., VVINF}	{$.}
Tag Extraction:	{NE}	{ART, ..., XY}	{VVFIN, VVIMP, VAFIN}	{ART, PDS, PRELS}	{NN}	{VVFIN, VVINF}	{$.}
Disambiguation:	NE	NE	VAFIN	ART	NN	VVINF	$.
Analysis Restriction:	{NE.*first*, NE.*last*}	{NE}	{VAFIN.3*rd.sg.pres*}	{ART.*sg.nom.fem*, ART.*sg.acc.fem*, ART.*pl.nom.**, ART.*pl.acc.**}	{NN.*masc.sg.nom*, ..., NN.*fem.sg.**}	{VVINF}	{$.}

Figure 2: Example of runtime `dwdst` module output

acter code 0x16) is used as a token delimiter, and two sequential newline characters (a blank line) indicate a sentence boundary, which effectively eliminates the empty token ε from the set of allowable tokens $Toks$.

All implemented heuristics for the tokenization of German are hard-coded into the preprocessor in the form of POSIX regular expressions. While such an implementational strategy is itself far from portable, the fact that the preprocessor runs independently of the rest of the tagging system, together with the existence of a well-defined Application Program Interface (API) for tokenized text makes up somewhat for the lack of a truly portable preprocessor.

3.2 Morphological Analyzer

The *morphological analyzer* segments and analyzes incoming tokens according to a set of decomposition rules represented as a finite-state transducer. The decomposition rules encoded in the morphological transducer are assumed to implement a body of prior linguistic knowledge regarding the behavior of the target language with respect to phenomena such as affixation, inflection, orthographic stem transformations, and compounding. For maximum portability and language independence, the morphological transducer must be specified by the user at runtime.

A complete discussion of the use of finite-state transducers for morphological analysis is beyond the scope of this document, but see Aho and Ullman (1972) and Roche and Schabes (1997) for background on finite-state devices, see Koskenniemi (1983) and Karttunnen et al. (1987) for details on the use of finite-state technology for morphological analysis, and see Hanneforth (2002b) for a discussion of the finite-state morphological analysis transducer for German used by the `dwdst` tagger at the Berlin-Brandenburgische Akademie der Wissenschaften.

Definition 3.2 (Morphological Transducer T_{Morph})
$T_{Morph} \in \mathcal{FST}_{\Sigma_T,\Sigma_A}$ is a finite state transducer with input alphabet Σ_T and with output alphabet Σ_A which relates token strings $w \in \Sigma_T^*$ to analyses $A \subseteq \Sigma_A^*$ by the regular function $|T_{Morph}| : \Sigma_T^* \to 2^{\Sigma_A^*}$.

In practice, the underlying finite-state device library (Hanneforth, 2002a) computes and returns the set of analyses $|T_{Morph}|(w)$ for a given token $w \in \Sigma_T^*$ as a finite-state automaton $M_w \in \mathcal{FSA}_{\Sigma_A}$ which recognizes the range of valid analyses. M_w is computed by converting the string w to an identity-transducer, composing it with T_{Morph}, and projecting the output tape of the resulting analysis transducer. The language recognized by the analysis automaton M_w is the set of valid analyses for w according to the morphological transducer: $\mathcal{L}(M_w) = |T_{Morph}|(w)$.

Crucial for the approach taken by the `dwdst` library is that for any token string $w \in \Sigma_T^*$, the set of analyses $|T_{Morph}|(w)$ is finite. Stated in terms of analysis

automata, this is equivalent[7] to requiring that any analysis automaton M_w returned by the morphological component is acyclic.

Definition 3.3 (Morphological Tagger Component)
The morphological tagger component $Morph : \Sigma_T^* \rightarrow \mathcal{FSA}^*_{\Sigma_A}$ itself is defined for token strings $w \in \Sigma_T^*$ as:[8]

$$Morph(w) = M_w = Proj_2(Id(Rec(w)) \circ T_{Morph})$$

As stated above, it is required that all M_w returned by the morphological component contain no complete cyclic paths.

The morphological component of the implemented `dwdst` tagging system may be called on its own as the executable program `dwdsm` for development and testing purposes, or it may be implicitly used by the high-level tagging program `dwdst` itself. In both cases, the user must specifiy a finite state transducer and alphabet specification to be used for analysis.

3.3 Lexical Classifier

The analyses returned by the morphological component may contain much more information than is required for part-of-speech tagging, including information about an input token's canonical form (lemma), segmentation boundaries, and/or morphosyntactic features such as case and number. Incorporating such features into the notion of a "part-of-speech tag" is undesirable for two reasons: first, it increases the size of the tagset, which makes the task of a part-of-speech tagger harder to accomplish and consequently reduces tagger accuracy. Second, currently available training corpora rarely provide the full range of information output by a highly informative morphological component, which all but eliminates the utility of such corpora for later estimation of HMM transition and emission probabilities.

The usual approach to such interface incompatibilities involves one or more conversion stages during the development phase. In this approach, incompatible morphological and corpus conventions are forced into a single set of application-specific conventions, either by explicit editing performed by human developers or by automatic means. Such conversion strategies have the desired effect of eliminating incompatibilities, but are highly dependent on the (host) application, and generally do not adapt well to changes on either side of the interface in question.

In the case of the German tagging system developed at the Berlin-Brandenburgische Akademie der Wissenschaften, the feature sets and conventions used by the

[7] I assume that M_w contains no useless transitions: that all transitions in M_w occur as part of some valid path from the initial state to a final state. Such a condition is easy to enforce.

[8] Here, $\circ$ indicates transducer composition, $|T_1 \circ T_2|(w) = |T_2|(|T_1|(w))$.

training corpus differed from those used by the morphological component. Also, the morphlogical transducer provides information regarding canonical form(s) and segmentation boundaries for each recognized token, in addition to part(s) of speech. One potential host application, a document indexing system, is capable of handling the degree of ambiguity induced by the presence of non-category features, but would benefit from the ambiguity reduction provided by the selection of a univocal "best" part of speech tag, while other applications require only the part of speech tag itself. It is difficult to see how a development-phase forced conversion strategy could accomodate the needs of diverse host applications in a sufficiently flexible manner.

The *lexical classifier* tagger component was introduced into the runtime `dwdst` part-of-speech tagging system to provide a well-defined abstract interface to varying conventions of a highly informative morphological component, and to allow sufficient flexibility in the interpretation of what exactly constitutes a part-of-speech tag and what does not – for instance, whether to exclude syntactic features from the notion of a "tag", while still retaining information regarding which tags of an ambiguous analysis are associated with which features.

The task of the lexical classifier is twofold: first, to identify ambiguity classes (sets of part-of-speech tags) from the analysis automata output by the morphological component, and second, to maintain information on the sets of full analyses associated with each of the extracted tags. Both tasks are efficiently and flexibly accomplished by means of a user-specifiable *tag-extraction transducer* $T_{TagX} \in \mathcal{FST}_{\Sigma_A,\Sigma_A}$ which relates analysis-strings $a \in \Sigma_A^*$ to tags $t \in Tags$, formally:

Definition 3.4 (Tag Extraction Transducer T_{TagX})
$T_{TagX} \in \mathcal{FST}_{\Sigma_A,\Sigma_A}$ is a finite state transducer with input and output alphabet Σ_A which relates analysis strings $a \in \Sigma_A^*$ to tags $t \in Tags$ by the regular function $|T_{TagX}| : \Sigma_A^* \to 2^{Tags}$.

Similar to the case of the morphological component, the underlying finite state device library returns the set of extracted tags for an analysis automaton $M_w \in \mathcal{FSA}_{\Sigma_A}^*$ on the output tape of a finite state transducer $X_w = Id(M_w) \circ T_{TagX}$. The set of extracted tags itself is then given by $c_w = \mathcal{L}(Proj_2(X_w))$.

An additional task of the lexical classifier is the implementation of default rules for handling tokens unknown to the morphology. Such tokens w are referred to as "unrecognized", and are identifiable by checking whether $\mathcal{L}(Morph(w)) = \emptyset$, which is efficient to compute. If such a token is found, then M_w, X_w, and c_w are computed as above for the special token[9] **u** which allows the user to encode the default analyses for unrecognized tokens directly into the morphological component. As a final fallback solution, the system considers all tags as empirically determined from a training corpus as possible analyses for w, behaving

[9] In the implemented `dwdst` system, the "unknown" token **u** is represented by the string `@UNKNOWN`.

$X_{M_w} = Id(\bigcup_{t \in Tags} Rec(t))$.[10]

For the simplest case where the morphological component outputs only part-of-speech tags, the user may choose not to specify T_{TagX}, in which case the system behaves as if T_{TagX} were an identity transducer for Σ_A^*, so that $X_{M_w} = Id(M_w)$ and $C_{M_w} = \mathcal{L}(M_w)$.

Definition 3.5 (Lexical Classification Tagger Component)
The lexical classification tagger component $Lex : \Sigma_T^* \to 2^{Tags}$ itself is defined for a morphological component $Morph$ as described above, an input token string $w \in \Sigma_T^*$, and a designated "unknown" token $\mathbf{u} \in \Sigma_T^*$ as:

$$Morph'(w) \quad = M'_w = \begin{cases} Morph(w) & \text{if } \mathcal{L}(Morph(w)) \neq \emptyset \\ Morph(\mathbf{u}) & \text{if } \mathcal{L}(Morph(w)) = \emptyset \text{ and } \mathcal{L}(Morph(\mathbf{u})) \neq \emptyset \\ \bigcup_{t \in Tags} Rec(t) & \text{otherwise} \end{cases}$$

$$TagX(w) \quad = X_w = \quad Id(Morph'(w)) \circ T_{TagX}$$

$$Lex(w) \quad = c_w = \quad \mathcal{L}(Proj_2(TagX(w)))$$

As for the morphological component, it is required that all M'_w and X_w returned by the lexical classification component contain no complete cyclic paths.

3.4 HMM Trainer

The *HMM trainer* module estimates model paramaters from a tagged training corpus for later use by the disambiguator module (Section 3.5) – formally, the task of the HMM trainer is the construction of an HMM $(\Sigma_D, Q_D, q_0, A_D, B_D)$ as described in Section 2.3.

Definition 3.6 (Training Corpus)
A *training corpus* $C \subset ((\Sigma_T^* \times Tags) \cup \{\odot\})^*$ is a finite sequence of delimited pairs $\odot_0(w_1^1, t_n^1) \cdots (w_{n_1}^1, t_{n_1}^1) \odot_1 \cdots \odot_{m-1} (w_1^m, t_1^m) \cdots (w_{n_m}^m, t_{n_m}^m) \odot_m$ where $t_j^i \in Tags$ is the tag associated with the token string $w_j^i \in \Sigma_T^*$ at index j of the i^{th} sentence in the corpus, for $1 \leq i \leq m$ and for $1 \leq j \leq n_i$.

Given a training corpus as above, the first step in the construction of a disambiguation HMM $Dis = (\Sigma_D, Q_D, q_D^0, A_D, B_D)$ is the empirical determination

[10] Disjunction of finite state automata is meant here by $\bigcup$, which is equivalent to union of their languages.

of the HMM tagset[11] $Tags_D$, and the set of known tokens $Toks_D$, where $\mathbf{u}$ is a distinguished placeholder for unknown runtime input tokens, and $\odot$ is the distinguished sentential boundary tag:

$$Toks_D = \bigcup_{i=1}^{m}\bigcup_{j=1}^{n_i}\{w_j^i\} \cup \{\mathbf{u}\} \tag{6}$$

$$Tags_D = \bigcup_{i=1}^{m}\bigcup_{j=1}^{n_i}\{t_j^i\} \cup \{\odot\} \tag{7}$$

Similarly, a set $Classes_D$ of all ambiguity classes occurring in the training corpus can be extracted by consulting the lexical classification module:

$$Classes_D = \bigcup_{i=1}^{m}\bigcup_{j=1}^{n_i}\{Lex(w_j^i)\} \tag{8}$$

Of course it is the case that given a finite tagset $Tags$, the set of all possible ambiguity classes 2^{Tags} is also finite. Exponential growth of both time and memory requirements would be the result of any direct use of such a definition, however, and was therefore deemed infeasible for actual implementation. In the case of the German morphlogical component using a 64-tag tagset, it was found that out of over 1.8×10^{19} possible ambiguity classes, only 467 were actually ever assigned to an input token in a training corpus of about 300,000 running words. Similarly, for a 712-tag tagset including morphological features, only 3153 distinct classes out of a total of 2.15×10^{214} theoretically possible were actually assigned. Due to these results, the `dwdst` system uses an internal enumeration of actually occurring ambiguity classes $Classes_D$, rather than the theoretically pure but computationally expensive powerset 2^{Tags}.

Definition 3.7 (Corpus Frequency)
Given a training corpus C as above, for a token-string $w \in \Sigma_T^*$ and tags $t, t_1, t_2 \in Tags$, the following *frequency functions* are defined:

- the tag-unigram frequency function $F_{ug}^C : Tags_D \to \mathbb{N}$;

$$F_{ug}^C(t) = Card(\bigcup_{i=1}^{m}\bigcup_{j=1}^{n_i}\{(i,j) \mid t_j^i = t\}) \tag{9}$$

- the tag-bigram frequency function $F_{bg}^C : Tags_D \times Tags_D \to \mathbb{N}$;

$$F_{bg}^C(t_1,t_2) = Card(\bigcup_{i=1}^{m}\bigcup_{j=1}^{n+1}\{(i,j) \mid t_{j-1}^i = t_1 \;\&\; t_j^i = t_2\} \tag{10}$$

[11] The HMM tagset $Tags_D$ can be understood as the empirically determined pendant of the formal tagset $Tags$ as described in Section 2.1. The need to construct such a relation forces me to adopt an interface level for which $\Sigma_A = \Sigma_T \cup \{\odot\}$ – this interface level is largely omitted here for purposes of clarity.

- the joint word-tag frequency function $F^C_{wd} : Toks_D \times Tags_D \to \mathbb{N}$;

$$F^C_{wd}(w,t) = Card(\bigcup_{i=1}^{m}\bigcup_{j=1}^{n_i}\{(i,j) \mid w^i_j = w \;\&\; t^i_j = t\}) \tag{11}$$

- and the joint class-tag frequency function $F^C_{cl} : Classes_D \times Tags_D \to \mathbb{N}$

$$F^C_{cl}(c,t) = Card(\bigcup_{i=1}^{m}\bigcup_{j=1}^{n_i}\{(i,j) \mid Lex(w^i_j) = c \;\&\; t^i_j = t\}) \tag{12}$$

Note that under- and overflow of indices for tag bigram frequencies are handled by setting $t^i_0 = t^i_{n_i+1} = \odot$ for $1 \le i \le m$.

3.4.1 Maximum Likelihood Estimates

Maximum likelihood estimates are computed for tag uni- and bigram probabilities, as well as for conditional token- and class-probabilities for $w \in Toks_D - \{\mathbf{u}\}$, $c \in Classes_D$, and $t, t_1, t_2 \in Tags_D$. For the purposes of the following definitions, zero divided by zero is considered to be zero.

$$\hat{P}_{ug}(t) = \frac{F^C_{ug}(t)}{\sum_{i=1}^{m} n_i} \tag{13}$$

$$\hat{P}_{bg}(t_2|t_1) = \frac{F^C_{bg}(t_1,t_2)}{F^C_{ug}(t_1)} \tag{14}$$

$$\hat{P}_{wd}(w|t) = \frac{F^C_{wd}(w,t)}{F^C_{ug}(t)} \tag{15}$$

$$\hat{P}_{cl}(c|t) = \frac{F^C_{cl}(c,t)}{F^C_{ug}(t)} \tag{16}$$

3.4.2 Smoothing

The maximum likelihood estimates are augmented by uni- and bigram smoothing constants λ_1 and λ_2, respectively, as well as by discriminating selector functions κ_1 and κ_2, and a small quantity $\hat{\varepsilon}$, which is used to prevent propagation of zero probabilities.

The n-gram smoothing constants $\lambda_1, \lambda_2 \in [0..1]$ must be such that $\lambda_1 + \lambda_2 + \hat{\varepsilon} = 1$, and are used to define the adjusted bigram probability function P_{bg} for the disambiguation HMM, for all $t_1, t_2 \in Tags_D$:

$$P_{bg}(t_2|t_1) = \lambda_1 \hat{P}_{ug}(t_2) + \lambda_2 \hat{P}_{bg}(t_2|t_1) + \hat{\varepsilon} \tag{17}$$

The `dwdst` trainer estimates values for λ_1 and λ_2 by the deleted interpolation algorithm described in Brants (2000), but also allows the user to override these estimated values.

The discriminating lexical selector functions $\kappa_1 : Toks_D \to \{0,1\}$ and $\kappa_2 : Toks_D \to \{0,1\}$ are used to exclusively select between tag-conditioned token and class MLE estimates for a token string $w \in Toks_D$:

$$\kappa_1(w) = \begin{cases} 0 & \text{if } w = \mathbf{u} \\ 1 & \text{otherwise} \end{cases} \tag{18}$$

The selector function κ_2 is simply the logical negation of κ_1:

$$\kappa_2(w) = \begin{cases} 1 & \text{if } w = \mathbf{u} \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

The selector functions κ_1 and κ_2 are used for the definition of an adjusted lexical probability function P_{lx} for the disambiguation HMM, for all $w \in Toks_D$, $c \in Classes_D$, and $t \in Tags_D$:

$$P_{lx}((w,c)|t) = (1-\hat{\varepsilon})(\kappa_1(w)\hat{P}_{wd}(w|t) + \kappa_2(w)\hat{P}_{cl}(c|t)) + \hat{\varepsilon} \tag{20}$$

Of course, given a constant morphological component and tag extractor, the ambiguity class associated with an input token string w is completely determined by the string w, so that the vast majority of token-class pairs (w,c) will never be needed. Indeed, the implemented `dwdst` trainer stores only either raw frequencies (text format) or MLE estimates with smoothing parameters (binary format) in the model parameter files it generates, in order to avoid memory bloating. The implemented system can thus be said to use an entirely different notion of what exactly constitutes a Hidden Markov Model – the definitions here are meant to formally describe the behavior of the implemented system in terms of a more traditional HMM definition.

3.4.3 Disambiguator Parameters

Finally, the parameters of the disambiguator HMM $H_D = (\Sigma_D, Q_D, q_0, A_D, B_D)$ can be defined for $w \in Toks_D$, $c \in Classes_D$, $t, t_1, t_2 \in Tags_D$:

$$\Sigma_D = Toks_D \times Classes_D \tag{21}$$
$$Q_D = Tags_D \tag{22}$$
$$q_0 = \odot \tag{23}$$
$$A_D(t_1,t_2) = P_{bg}(t_2|t_1) \tag{24}$$
$$B_D(t,(w,c)) = P_{lx}((w,c)|t) \tag{25}$$

In the implemented `dwdst` system, known token strings $w \in Toks_D$, as well as tags $t \in Tags_D$ and ambiguity classes $c \in Classes_D$ are represented by

natural number identifiers, for efficient storage and retrieval. Also, as mentioned above, the observation probability table B_D is split into subcomponents for token strings $B_{D_{tok}}$ and lexical classes $B_{D_{class}}$, for data reduction purposes. Undefined entries in these tables are treated as zero probabilities.

3.5 HMM Disambiguator

The *HMM disambiguator* module is based on a modified version of the Viterbi algorithm (Definition 2.18), tailored for the availability of ambiguity class information for each observed token. Specifically, the modified Viterbi algorithm computes the disambiguation function $\tau_D : (Toks_D \times Classes_D)^n \to Tags_D^n$ for finite $n \in \mathbb{N}$:

$$\tau_D(w_{1..n}) = \arg \max_{t_{1..n} \in Tags_D^n} \prod_{i=1}^{n} P((w_i, Lex(w_i))|t_i) P(t_i|t_{i-1}) \tag{26}$$

Definition 3.8 (Modified Viterbi Algorithm)
The modified version of the Viterbi algorithm used by the `dwdst` system makes use of the ambiguity classes encoded as components of the observation alphabet Σ_D in order to reduce the search space and execution time.

1. **Input:** $tok_{1..n} \in Sents$
2. **Observations:** for $1 \leq i \leq n$:
   (a) Filter unknown tokens from the input sentence by setting:
   $$w_i := \begin{cases} tok_i & \text{if } tok_i \in Toks_D \\ \mathbf{u} & \text{otherwise} \end{cases}$$
   (b) Classify token-strings by consulting the lexical classification module, setting:
   $c_i := Lex(w_i)$
3. **State Variables:**
   Time-dependent variables $\delta_t(i)$ and $\psi_t(i)$ are defined as for the traditional Viterbi algorithm.
4. **Initialization:**
   $$\delta_t(0) = \begin{cases} 1 & \text{if } t = \odot \\ 0 & \text{otherwise} \end{cases}$$
5. **Repeat:** for $1 \leq i \leq n$
   $$\delta_t(i) = \max_{t' \in c_i} \delta_{t'}(i-1) A(t', t) B(t, (w_i, c_i))$$
   $$\psi_t(i) = \arg\max_{t' \in c_i} \delta_{t'}(i-1) A(t', t) B(t, (w_i, c_i))$$

6. **Terminate:**

$$\begin{aligned} t_n &= \arg\max_{t \in c_n} \delta_t(n) A(t, \odot) \\ t_{i-1} &= \psi_{t_i}(i), \text{ for } n \geq i > 1 \end{aligned}$$

7. **Output:** Tag sequence $t_{1..n} \in Tags_D^n$.

This modified Viterbi algorithm uses part-of-speech ambiguity classes as lexical equivalence classes for purposes of n-gram based stochastic sequence optimization, a technique implemented by DeRose (1988), who noted that lexical equivalence classes can also be used as a means of data reduction, since experience shows there to be fewer actually ocurring part-of-speech ambiguity classes than there are literal text tokens. Use of literal tokens as input symbols provides more accurate results, however – the `dwdst` system allows users to specify a minimum absolute frequency which a literal token must exceed in the training corpus before becoming eligible to act as a "known" token, thus being passed unchanged through step (2a) above.

An additional novelty of the modifified algorithm is the restriction of the algorithm's search space based on the constitution of the lexical ambiguity classes – only those tags are considered possible analyses at a given input index i which appear as elements of the ambiguity class c_i associated with that index. In this way, the body of prior linguistic knowledge assumed to be encoded in the morphology transducer is exploited by the HMM disambiguation module. A traditional and formally equivalent implementation of this technique is setting those output probabilities $B(t, (w, c))$ to zero whenever $t \notin c$. Despite its formal correctness, the traditional description – if implemented as such – still requires at least $O\left(n^{Card(Tags)^2}\right)$ execution time, since each tag must still be considered at each time increment. The modified algorithm requires on average $O\left(n^{Avg(Card(c_i))^2}\right)$ execution time – proportional to the average size of a lexical ambiguity class c_i. For the German morphological component, $Avg(Card(c_i)) \approx 2.2$ for $Card(Tags) = 64$, and $Avg(Card(c_i)) \approx 5.2$ for $Card(Tags) = 712$, so the efficiency improvements provided by such search space reduction are indeed non-trivial.

3.6 Analysis Restrictor

The *analysis restrictor* is responsible for pruning out those analyses returned by the morphological transducer which do not correspond to the most likely part-of-speech tag as determined by the HMM disambiguator. This task is efficiently accomplished for each input token-string w by reference to the tag-extracted analysis transducer $TagX(w)$ returned by the lexical classification module,[12] which directly encodes the relation between part-of-speech tags and

[12] See Definition 3.5 for the definition of the tag-extracted analysis transducer $TagX(w)$.

Corpus	Training Size: 355096 tok Test Size: 35332 tok Tagset Size: 55 tags
Morphology	Recognition: 97.21 % Coverage: 95.13 % Tagset Size: 64 tags
Disambiguation	Saves: 61.13 % Success: 97.67 %
Global	Accuracy: 95.19 % Throughput: 25047 tok/sec

Table 1: Results for the implemented `dwdst` tagger

full analyses, and is temporarily stored by the `dwdst` system as a component of the internal sentence buffer.

Definition 3.9 (Analysis Restriction Tagger Component)
The analysis restriction tagger component $Restrict : \Sigma_T^* \times Tags \rightarrow 2^{\Sigma_A^*}$ is defined for a an input token string $w \in \Sigma_T^*$ and a univocal part-of-speech tag $t \in Tags$ as:

$$Restrict(w,t) = \mathcal{L}\left(Proj_1(TagX(w) \circ Id(Rec(t)))\right)$$

Note that in the trivial case of a one-to-one mapping between morphological analyses and part-of-speech tags, $Restrict(w,t) = \{t\}$, by Definition 3.5. For efficiency reasons, the `dwdst` system treats the case of a one-to-one mapping as a special case in which the user specifies no tag-extraction transducer T_{TagX}. In this case, no tag-extracted transducer $TagX(w)$ is actually constructed, in order to avoid unecessary calls to the underlying finite state device library.

4 Results

The system described in Section 3 was implemented in C++ and tested for German with the finite-state morphological component described in Hanneforth (2002b), using the NEGRA corpus (Skut et al., 1997) for training and evaluation. 90% of the NEGRA corpus was used for training, and the remaining 10% was reserved for evaluation and testing. All tests were performed on a machine with an Athlon XP 2000+ (1.66GHz) CPU running Linux. A summary of the results is given in Table 1.

Some explanation of the data presented in Table 1 is in order. The *morphological recognition rate* is defined as the number of input tokens for which the morpho-

[12]Reported throughput includes disk I/O.

logical component returned a non-empty analysis divided by the total number of input tokens. The *morphological coverage rate* is defined as the number of input tokens for which the correct tag (according to the evaulation/test corpus) was included in the set of analyses returned by the morphological component.

The *disambiguation save rate* is defined as the number of unrecognized tokens which were assigned the correct tag by the disambiguation module divided by the number of unrecognized tokens. The *disambiguation success rate* is defined as the number of covered tokens which were assigned the correct tag by the disambiguation module divided by the number of covered tokens. *Global accuracy* refers to the usual concept, defined as the number of correct tag assignments made by the tagging system divided by the total number of input tokens.

The `dwdst` tagger described here was also evaluated against a more traditional bigram HMM tagger using a limited number of lexical classes based on the presence/absence of punctuation and/or digits in the input tokens.[13] Tests showed that the `dwdst` tagger achieves higher accuracy rates from smaller training corpora than the traditional HMM approach – for the corpus configuration described above, `dwdst`'s use of lexical equivalence class probabilities in place of literal token probabilities when the latter are not available resulted in a 17.6% reduction in errors with respect to the baseline given by the traditional HMM tagger. These comparative results are plotted for against training-corpus size in Figure 3.

The interpretation of morphological analyses as a source of prior linguistic knowledge allows a theoretical speed improvement of *circa* 96% with respect to a traditional HMM tagger. In practice however, the `dwdst` tagger with a high-coverage morphological component is not significantly faster than a traditional HMM tagger using heuristic search-space reduction techniques such as *beam search*, due in a large part to the additional overhead introduced by the morphological lookup and classification routines. Additionally, the uncontestability of non-empty morphological analyses enforces a strict upper bound on tagger accuracy: the correct tag cannot be assigned by the disambiguator module if the ambiguity class for the token in question does not contain that tag as an element. High morphological coverage is therefore critical for application of the tagging technique described here. Indeed, it is better for overall tagger accuracy for the morphological module to return an empty analysis set than for it to return a non-empty analysis set which does not contain the correct tag.

5 Future Work

Although the initial results for the implemeneted `dwdst` system are encouraging, many improvements to the system as it stands could still be made. The

[13] The more traditional HMM tagger used for these tests is also implemented and distributed as part of the `dwdst` library as the executable program `dwdshmm`.

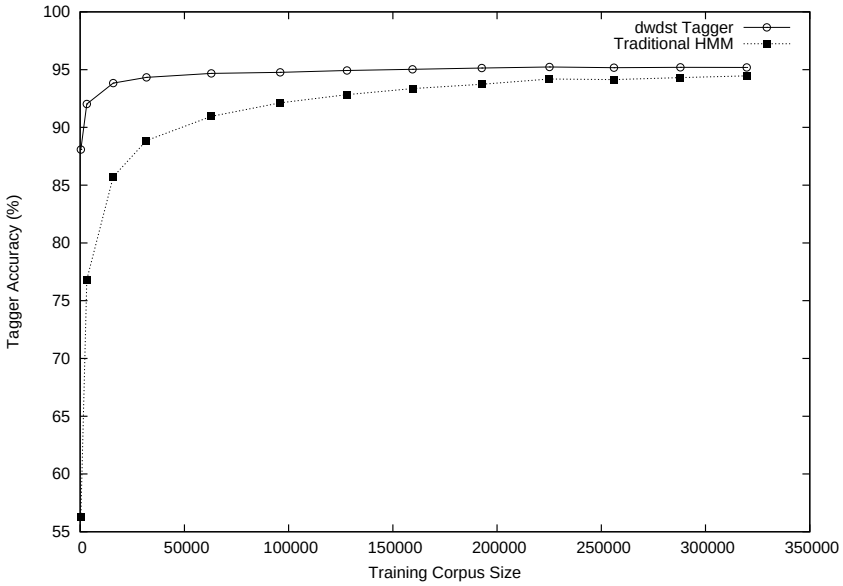

Figure 3: Comparison of `dwdst` *vs.* a traditional HMM tagger

use of tag trigrams – in addition to uni- and bigrams – to estimate state transition probabilities for the disambiguator module as described for a traditional HMM tagger in Brants (2000) could well improve overall overall accuracy. Also worth investigation would be a flexible method by which to incorporate segmentation information returned by the morphological module into the emission probabilities for the disambiguator, since it has been shown (Charniak et al., 1993; Samuelsson, 1993) that even a simple string-suffix analysis can improve accuracy for traditional HMM taggers. Along these lines, a revised morphological lookup routine might be implemented to provide a linguistically motivated suffix analysis even when the corresponding stem(s) are not encoded in the underlying transducer. With such improvements implemented, I speculate that overall tagger accuracy might reach or even exceed the level of more mature tagging systems.

6 Conclusion

In this paper, I have presented an approach to part-of-speech tagging based on the integration of a rule-based morphological analyser encoded as a finite-state transducer with a Hidden Markov model disambiguator. Inclusion of lexical equivalence classes as determined by the morphological analyser as sources of categorical information allowed a theoretically motivated search space reduction

for the modified Viterbi algorithm used by the disambiguator. Use of lexical classes as equivalence classes for HMM emission probability estimates was shown to improve overall tagger accuracy with respect to a more traditional approach, especially for small training corpora. Initial results for accuracy and throughput are encouraging, but a number of optimizations might be made to further improve the system.

References

A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation, and Compiling, Volume I: Parsing*. Prentice-Hall, Englewood Cliffs, NJ, 1972. ISBN 0-13-914556-7.

J. Allen, S. Hunnicut, and D. Klatt. *From Text to Speech: the MITalk system*. Cambridge University Press, 1987.

T. Brants. TnT – a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000*, Seattle, WA, April 29 – May 3 2000.

E. Brill. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing*, pages 152–155, Trento, IT, 1992. URL `http://acl.ldc.upenn.edu/A/A92/A92-1021.pdf`.

E. Brill. Some advances in transformation-based part of speech tagging. In *National Conference on Artificial Intelligence*, pages 722–727, 1994. URL `ftp://ftp.cs.columbia.eduq/pub/cs4999/brill94.ps`.

E. Charniak. *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts, 1993.

E. Charniak, C. Hendrickson, N. Jacobson, and M. Perkowitz. Equations for part-of-speech tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence, Menlo Park*. AAAI Press/MIT Press, 1993. URL `http://www.cs.brown.edu/people/ec`.

K. W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the 2nd Conference on Applied Natural Language Processing*, pages 136–143, 1988.

A. Coëtmeur. *flex++ Version 2.3.8-4*, 1993. URL `ftp://iecc.com/pub/file/bison++flex++`.

D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, 1992.

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

T. Dutoit. *An Introduction to Text-to-Speech Synthesis*. Kluwer, Dordrecht, 1997.

A. Geyken, T. Hanneforth, and A. Sokirko. Dialing-DWDS concordance (DDC). Executive Board Meeting ELSNET, 14–15 October 2002.

T. Hanneforth. FSMlib: C++ library for finite state device operations. Unreleased development version, 2002a.

T. Hanneforth. Finite-state morphology for German. Talk given at Universität Heidelberg, February 2002b.

B. Jurish. Part-of-speech tagging with finite-state morphology. Poster presented at the conference *Collocations and Idioms: linguistic, computational, and psycholinguistic perspectives*, Berlin, 18.–20. September, 2003. URL `http://www.ling.uni-potsdam.de/~moocow/pubs/kollok2003.pdf`.

L. Karttunnen, K. Koskenniemi, and R. M. Kaplan. A compiler for two-level phonological rules. In *Tools for Morphological Analysis*. Center for the Study of Language and Information, Stanford University, Palo Alto, CA, 1987.

K. Koskenniemi. *Two-level Morphology. A General Computational Model for Word-Form Recognition and Production*. PhD thesis, University of Helsinki, Department of General Linguistics, 1983.

C. Manning. Automatic acquisition of a large subcategorization dictionary from corpora. In *Proceedings of the 31st Annual Meeting of the ACL*, 1993.

C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, 1999.

L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

E. Roche and Y. Schabes, editors. *Finite–State Language Processing*. MIT Press, Cambridge, Massachusetts, 1997.

C. Samuelsson. Morphological tagging based entirely on Bayesian inference. In *9th Nordic Conference on Computational Linguistics NODALIDA-93*, Stockholm University, Stochkholm, Sweden, 1993.

W. Skut, B. Krenn, T. Brants, and H. Uszkoreit. An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing ANLP-97*, Washington, DC, 1997.

A. Ushioda, D. A. Evans, T. Gibson, and A. Waibel. The automatic acquisition of frequencies of verb subcategorization frames from tagged corpora. In B. Boguraev and J. Pustejovsky, editors, *Proceedings of the SIGLEX ACL Workshop on the Acquisition of Lexical Knowledge from Text*, pages 95–106, Columbus, Ohio, 1993.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transications on Information Theory*, pages 260–269, April 1967.