

Chapter 1

Part-of-Speech Tagging

1.1 Introduction

Part-of-speech (POS) tagging is the task of determining the correct parts of speech for a sequence of words. Words are often ambiguous wrt. part of speech. The word *back* e.g. is a noun in the sentence *They stabbed him in the back*, a finite verb in the sentence *The senators back the tax reduction*, an infinitive verb in the sentence *They will back the proposal*, an adjective in the sentence *The charity owes \$400,000 in back taxes*, and an adverb in the sentence *It's too late to put the genie back in the bottle*.

POS tagging is useful for a large number of applications: It is the first analysis step in many syntactic parsers. It is required for the correct lemmatization of words like *saw* which is ambiguous between the noun *saw* and the past tense form of the verb *to see*, and it is used in information extraction, speech synthesis, lexicographic research, term extraction, and many other applications.

The set of tags assigned by a part of speech tagger may contain just a dozen tags or many hundred tags. A typical tagset size is somewhere between 50 and 150 tags. Larger tagsets with several hundred tags are sometimes used with morphologically rich languages like German where the number, gender and case features of nouns, adjectives, and determiners lead to an explosion in the number of tags. However, POS tagging with such large tagsets is difficult for most taggers due to sparse data problems and because some ambiguities are hard to resolve without global context. Compare the two German sentences *Er hat seine Frau gesprochen.* (He has his wife talked. – He has talked to his wife.) and *Dann hat seine Frau gesprochen.* (Then has his wife talked. – Then, his wife has talked.). The noun phrase *die Frau* is the accusative object in the first sentence, but the nominative subject in the second sentence. The disambiguation of the case is hardly feasible without a complete analysis of the sentence, which is the task of a parser (see chapter ?? of this book).

POS tags only represent morphosyntactic distinctions. A POS tagger is therefore able to distinguish between the noun reading and the verb reading of the word *accent*, but

not between the *word accent* reading and the *foreign accent* reading of the noun. The latter task is called *word sense disambiguation* and requires other techniques than POS tagging.

A large number of methods have been applied to POS tagging over the years. Among them are Hidden Markov Models (Church, 1988; Cutting et al., 1992; Brants, 2000), transformation-based learning (Brill, 1992), memory-based learning (Daelemans et al., 1996), maximum-entropy modeling (Ratnaparkhi, 1996), support vector machines (Giménez and Màrquez, 2004), neural networks (Benello et al., 1989; Nakamura et al., 1990; Schmid, 1994a), decision trees (Black et al., 1992; Màrquez and Padró, 1997) and manually written disambiguation rules (Greene and Rubin, 1971; Tapanainen and Voutilainen, 1994; Klatt, 2002). In principle, would also be possible to use a statistical parser for this purpose, but standard POS taggers are usually much faster than parsers and tend to be more accurate.

The typical accuracy of POS taggers is between 95 % and 98 % depending on the tagset, the size of the training corpus, the coverage of the lexicon, and the similarity between training and test data. This is impressive, but an accuracy of 96 % means that a 20-word sentence is correctly tagged with a probability of just $0.96^{20} \approx 44\%$.

The rest of the chapter is organized as follows. The next section introduces POS tagging with Hidden Markov Models, which is probably the most widely used approach to POS tagging. The following sections describe further tagging methods. Section 1.3 presents Log-Linear Models including Maximum Entropy taggers. Section 1.4 describes POS taggers which are based on classification algorithms such as neural networks, support vector machines, decision trees, or memory-based learning. Section 1.5 is devoted to the Brill tagger and its transformation-based learning strategy. POS taggers which use manually developed disambiguation rules are described in section 1.6, and the last section discusses the adaptation of POS taggers to other languages.

1.2 Hidden Markov Models

From a statistical point of view, the task of a POS tagger is to find the most likely POS sequence $\hat{t}_1, \hat{t}_2, \dots, \hat{t}_n = \hat{t}_1^n$ for a given word sequence $w_1, w_2, \dots, w_n = w_1^n$. In other words, the tagger has to maximize the conditional probability $p(t_1^n | w_1^n)$ of the tag sequence given the word sequence over all possible tag sequences t_1^n :

$$\hat{t}_1^n = \arg \max_{t_1^n} p(t_1^n | w_1^n) = \arg \max_{t_1^n} \frac{p(t_1^n, w_1^n)}{p(w_1^n)} = \arg \max_{t_1^n} p(t_1^n, w_1^n)$$

The prior probability $p(w_1^n)$ of the word sequence is positive and independent of the POS sequence. Therefore, it has no influence on the ranking of the different POS sequences and can be ignored. The joint probability $p(t_1^n, w_1^n)$ of the tag sequence and the word sequence is decomposed into a product of conditional probabilities:

$$p(t_1^n, w_1^n) = p(t_1, w_1, t_2, w_2, \dots, t_n, w_n) = \prod_{i=1}^n p(t_i | t_1^{i-1}, w_1^{i-1}) p(w_i | t_1^i, w_1^{i-1})$$

Assuming that the next part of speech depends only on the k preceding POS tags and that the next word depends only on its part of speech, the following formula is obtained:

$$p(t_1^n, w_1^n) = \prod_{i=1}^n p(t_i | t_{i-k}^{i-1}) p(w_i | t_i)$$

These assumptions are called *Markov assumptions* and the resulting statistical model is a Hidden Markov Model (HMM) (Rabiner, 1989; Manning and Schütze, 1999). Each state of the Hidden Markov Model corresponds to a k -tuple $\langle t_1, t_2, \dots, t_k \rangle$ of POS tags. The values of k used in POS tagging are usually 1 (bigram tagger) or 2 (trigram tagger). Because of data sparseness problems, a context larger than 2 rarely improves the performance. The *transition* probability $a_{s,s'}$ between two states $s = \langle t, t' \rangle$ and $s' = \langle t'', t''' \rangle$ of a trigram tagger ($k = 2$) is $p(t''' | t, t')$ if $t' = t''$ and 0 otherwise. The *emission* probabilities of a trigram tagger are defined as $b_{s,w} = p(w | t')$ if $s = \langle t, t' \rangle$.

Viterbi Algorithm The best POS tag sequence for the word sequence w_1^n is efficiently computed by the *Viterbi* algorithm (Manning and Schütze, 1999). The Viterbi algorithm applies dynamic programming to compute the probability $\delta_s(k)$ of the most probable state sequence $s_0, s_1, \dots, s_{k-1}, s$ which generates w_1, \dots, w_k and ends with state s , for increasing length k ($0 \leq k \leq n$).

$$\delta_s(k) = \max_{s_0^k: s_k=s} \pi_{s_0} \prod_{i=1}^k a_{s_{i-1}s_i} b_{s_i w_i}$$

The Viterbi probabilities $\delta_s(k)$ are efficiently computed as follows:

$$\begin{aligned} \delta_s(0) &= \pi_s \\ \delta_s(k) &= \max_{s'} \delta_{s'}(k-1) a_{s',s} b_{s,w_k} \text{ for } 0 < k \leq n \\ \psi_s(k) &= \arg \max_{s'} \delta_{s'}(k-1) a_{s',s} b_{s,w_k} \text{ for } 1 < k \leq n \end{aligned}$$

The Viterbi probability of state s for $k = 0$ is the initial probability π_s . The initial probability distribution solves the problem that $p(t_i | t_{i-2}, t_{i-1})$ is undefined for $i < 2$. The other Viterbi probabilities are computed by maximizing the product of the Viterbi probability of the preceding state and the transition probability and the emission probability over all possible preceding states. The best tag sequence is obtained by computing $\arg \max_s \delta_s(n)$. The other tags of the most probable tag sequence are recovered by means of the ψ -variable which records the best predecessor state for each state and word.

The parameters of the Hidden Markov Model have to be estimated from training data. Given a corpus which is annotated with POS tags, the *Maximum Likelihood estimate* (MLE) of the transition and emission probabilities of a trigram tagger are computed from the POS trigram frequencies $f(t, t', t'')$, the POS bigram frequencies $f(t, t')$, the

POS unigram frequencies $f(t)$, and the word/POS pair frequencies $f(w, t)$ in the training corpus:

$$p(t''|t, t') = \frac{f(t, t', t'')}{f(t, t')}$$

$$p(w|t) = \frac{f(w, t)}{f(t)}$$

Finally, the initial state probabilities π_s have to be estimated. Assuming that the first word starts a new sentence, the start state distribution should reflect the distribution of the states of the HMM immediately before it starts to process a new sentence. If the sentence boundaries are marked in the training corpus, it is possible to estimate the initial probabilities of a trigram tagger from the occurrence frequencies $f_F(t, t')$ of tag pairs immediately before a sentence boundary:

$$\pi(\langle t, t' \rangle) = \frac{f_F(t, t')}{\sum_{t'', t'''} f_F(t'', t''')}$$

An alternative strategy which is easy to implement if the input is processed sentence by sentence is the following: Two special tokens ;S_i which are tagged with the special tag t_S are inserted before any sentence in the training corpus. The parameters are then estimated as usual, and the start state of the HMM is the state $\langle t_S, t_S \rangle$ with probability 1. The tagger appends the special token ;S_i to each sentence and computes the Viterbi tag sequence as described above. A major difference between the two methods for the definition of the start states lies in the treatment of cross-sentential dependencies: Unlike the first method, the latter method prevents that the tagging of a sentence is influenced by the preceding sentence.

Sparse Data Problem Given a tagset with 100 tags, a trigram tagger needs to estimate $100^3 = 1$ million different transition probabilities. The number of parameters is therefore in the same range as the size of the largest manually annotated corpora available for training like the Penn treebank (Marcus et al., 1993) for English or the Negra corpus (Skut et al., 1998) for German. It is therefore likely that some perfectly well-formed, but less probable POS trigram sequences do not show up in the training data and therefore receive a zero probability. Any tag sequence which contains such an unobserved trigram will also get a zero probability and will only be output if the tagger finds no other alternative. This *sparse data problem* is solved by parameter smoothing which redistributes a small fraction of the probability mass from observed events to unobserved events. A description and comparison of different smoothing techniques is provided e.g. in (Chen and Goodman, 1996).

Unknown words Another problem arises with words which do not occur in the training corpus. If a dictionary with POS information is available, the tagger can use parameter smoothing to assign a small probability to words which occur in the lexicon but not

in the training corpus. However, this doesn't solve the problem with words which do not occur in the dictionary, either. One option is to put unknown and rare words into a class which is treated like a single word. To this end, any word occurring only once (or less frequently than some threshold) in the training data is replaced with a special token “unknown_i”. The parameters are then estimated as usual. During tagging, any word which does not occur in the modified training corpus, is also replaced with “unknown_i” and tagged as usual.

This simple approach ignores the information which the unknown word form provides about its possible parts of speech. A word like *pancying* e.g. is either a gerund, an adjective or a noun. The word *Wenyng* on the other hand is a proper name unless it appears at the beginning of a sentence. In order to improve the tagging of unknown word forms, it is necessary to take this information into account. The first step is the replacement of $p(w|t)$, the probability of the word given the tag, with $p(t|w)$, the probability of the tag given the word, in the HMM formula by applying Bayes' theorem:

$$\begin{aligned} \hat{t}_1^n &= \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_{i-k}^{i-1}) p(w_i | t_i) \\ &= \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_{i-k}^{i-1}) \frac{p(t_i | w_i) p(w_i)}{p(t_i)} \\ &= \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_{i-k}^{i-1}) \frac{p(t_i | w_i)}{p(t_i)} \end{aligned}$$

$p(t|w)$ is easier to estimate than $p(w|t)$ because it requires no information about the probability of the unknown word. After applying Bayes' theorem, this prior probability $p(w_i)$ appears as a separate factor which is positive and independent of the tag sequence and can therefore be ignored. The prior probability $p(t)$ of the POS tags is easily estimated from the training corpus by dividing the tag frequency with the corpus size.

Brants (2000) then estimates the tag probabilities $p(t|w)$ for unknown words based on the suffix c_{n-k}^n of the unknown word $w = c_1^n$.

$$p(t|c_1^n) = p(t | \text{word ends in } c_{n-k}^n)$$

The probability of the tag t given the word suffix c_{n-k}^n is estimated by dividing the number of words in the corpus with the given suffix and POS tag by the total number of words with the given suffix. The probabilities are smoothed by interpolation with the tag probabilities for all shorter suffixes. Brants uses separate models for lower-case words and capitalized words in order to capture capitalization information. Similar strategies have been presented in (Schmid, 1994b) and (Weischedel et al., 1993).

Ambiguous Output Sometimes a HMM POS tagger has not enough information to reliably disambiguate between two POS tags of a word. The Viterbi algorithm is nevertheless forced to return a single analysis. For some applications like parsing, it

might be better to obtain both tags when the tagger is unsure what the correct tag is. Ideally, the tagger would return for each word the list of possible tags and their probability given the input sentence. The application could then decide which tags to consider. The computation of these probabilities is possible with the Forward-Backward algorithm (Manning and Schütze, 1999). It computes the sum of the probabilities of all the tag sequences where the i -th tag is t , divided by the sum of the probabilities of all tag sequences.

$$p(t_i = t | w_1^n) = \frac{\sum_{t_1^n: t_i = t} p(t_1^n, w_1^n)}{\sum_{t_1^n} p(t_1^n, w_1^n)}$$

In case of a bigram tagger, each POS tag t corresponds to a state s_t and the sum of the probabilities of all tag sequences with $t_i = t$ is efficiently computed as the product of the forward probability $\alpha_{s_t}(i)$ and the backward probability $\beta_{s_t}(i)$ which are defined as follows:

$$\begin{aligned} \alpha_s(0) &= \pi_s \\ \alpha_s(i) &= \sum_{s'} \alpha_{s'}(i-1) a_{s',s} b_{s,w_i} \text{ for } 0 < i \leq n \\ \beta_s(n) &= 1 \\ \beta_s(i) &= \sum_{s'} \beta_{s'}(i+1) a_{s,s'} b_{s',w_{i+1}} \text{ for } 0 \leq i < n \end{aligned}$$

The computation of the forward probabilities $\alpha_s(i)$ is identical to the computation of the Viterbi probabilities $\delta_s(i)$, except that maximization has to be replaced by summation. In case of a trigram tagger, it is necessary to sum the product of forward and backward probabilities over all states $s = \langle t', t'' \rangle$ with $t'' = t$. The sum of the probabilities of all possible tag sequences is obtained by summing the forward probabilities of the last word over all states. For a trigram tagger, the tag probabilities are therefore computed as follows:

$$p(t_i = t | w_1^n) = \frac{\sum_{s=\langle t', t \rangle} \alpha_s(i) \beta_s(i)}{\sum_s \alpha_s(n)}$$

Unsupervised Training Given a dictionary with the set of possible POS tags for each word, it is also possible to train a HMM tagger in unsupervised mode on unlabeled training data (Cutting et al., 1992). The basic idea is to start with a HMM tagger whose parameters have been somehow initialized and to use this tagger to annotate data. The parameters of the tagger are then reestimated based on the annotated data. This process is repeated for a number of iterations. This *Expectation-Maximization* algorithm (Baum, 1972) is known to monotonically increase the probability assigned to the training data by the HMM in each iteration, if the forward-backward algorithm is used to compute estimated event frequencies. However, the local optimum approached by the EM training is not guaranteed to be also a global maximum. Which local optimum is reached, depends on the start state. Therefore it is important to properly initialize the

model, e.g. by estimating parameters from a small manually annotated training corpus. Another problem is that increasing the probability of the training data not necessarily improves the tagging accuracy. The two criteria are too different. If the initial model is already quite good, EM training will actually decrease the performance. The results obtained with unsupervised training are therefore usually worse than those obtained with smoothed parameter estimates from manually labeled training data (Elworthy, 1994). Nevertheless, EM training is an option if a dictionary and a raw text are available, but no annotated training corpus.

Sentence-Initial Capitalization The fact that sentence-initial words are always capitalized poses two problems to POS taggers: Words at the beginning of a sentence are unknown to the tagger if they are normally written in lower-case and if they only appear in the training corpus in non-initial position. The other problem is that a word like *Bacon* might be tagged as a regular noun rather than a proper name in the middle of a sentence if it appeared frequently enough as a regular noun in sentence-initial position in the training corpus.

These problems could be solved if the training corpus is modified such that words in sentence-initial position which would normally be written in lower-case, are decapitalized. Furthermore, the tagger needs to consider both the capitalized and the decapitalized form of words in sentence-initial position and should combine the sets of possible POS tags by weighting them according to the relative frequencies of the word forms (Schmid, 1995).

Ambiguous Input Thus far, it has been assumed that word segmentation is unambiguous. This is not always the case: In ideographic languages like Chinese, a sentence is just a sequence of characters without word boundaries, and there are usually several word sequences yielding the same character sequence (see chapter ?? of this book). The same situation arises when a speech recognizer produces ambiguous output, or when the tokenizer leaves the disambiguation between an abbreviation period and a full stop undecided.

The following modification of the Viterbi algorithm for ambiguous input assumes that s_w holds the start position of the word w and e_w its end position. If the end position of w is identical to the start position of w' , then w' is a possible successor of w . The Viterbi probabilities are now computed as follows:

$$\begin{aligned}\delta_s(0) &= \pi_s \\ \delta_s(i) &= \max_{s', w: e_w=i} \delta_{s'}(s_w) a_{s',s} b_{s,w} \text{ for } 0 < i \leq n \\ \psi_s(i) &= \arg \max_{s', w: e_w=i} \delta_{s'}(s_w) a_{s',s} b_{s,w} \text{ for } 0 < i \leq n\end{aligned}$$

Lexicalization Standard HMM POS taggers decompose the joint probability of the tag sequence and a word sequence into a product of conditional probabilities and assume

that the next POS tag depends only on the k preceding tags and that the next word depends only on its tag.

$$\begin{aligned}\hat{t}_1^n &= \arg \max_{t_1^n} p(t_1^n, w_1^n) \\ &= \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_1^{i-1}, w_1^{i-1}) p(w_i | t_1^i, w_1^{i-1}) \\ &= \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_{i-k}^{i-1}) p(w_i | t_i)\end{aligned}$$

The strong independence assumptions of the standard HMM are too strict for natural languages. In the Penn treebank, e.g., the word “nothing” never occurs after articles unlike most other words which are tagged as nouns. Another example is the preposition *than* which is far more probable if the word *more* precedes.

Lee et al. (2000) therefore weakened the independence assumptions and obtained a model which they called a *lexicalized* Hidden Markov Model:

$$\hat{t}_1^n = \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | t_{i-k}^{i-1}, w_{i-k}^{i-1}) p(w_i | t_{i-j}^i, w_{i-j}^{i-1})$$

In their model, the next tag depends on the k preceding tags *and* words; the next word depends on the j preceding tags and words and the current tag. The parameters of the lexicalized model are estimated with a back-off smoothing strategy.

Complex Tags The tagsets used for English usually contain between 30 and about 150 different tags. These tagsets are small compared to the tagsets used for language with a richer morphology like German or French. Due to the encoding of number, gender and case information, these tagsets often contain hundreds of tags. The German word *kleiner* e.g. might be tagged as *Adjective.attributive.nominative.singular.masculine.strong-inflection*. Standard HMM taggers have difficulties to deal with such tagsets because the number of parameters of a trigram tagger e.g. grows cubically with the size of the tagset.

Kempe (1993) therefore proposed to split complex POS tags t_i into feature bundles $f_{i,1}^N$ where N is the number of features. The part-of-speech tag of the word *kleiner* might be split into the feature-value pairs:

$$\left(\begin{array}{l} POS = adjective \\ Attributive = + \\ Case = nominative \\ Number = singular \\ Gender = masculine \\ Inflection = strong \end{array} \right)$$

The probability $p(f_1^N|C)$ of a feature bundle $f_{i,1}^N$ given the context C is decomposed into a product of conditional feature value probabilities $\prod_{k=1}^N p(f_k|C, f_1^{k-1})$. If the k -th feature is the *Number* feature, then the value f_k is either *singular*, or *plural*, or *undefined*. The conditional probabilities of the three values must sum to 1.

The probability of a tagged sequence of words is now defined as:

$$\prod_{i=1}^n p(t_i|t_{i-2}, t_{i-1}) p(w_i|t_i) = \prod_{i=1}^n \left(\prod_{k=1}^N p(f_{i,k}|f_{i-2,1}^N, f_{i-1,1}^N, f_{i,1}^{k-1}) \right) p(w_i|f_1^N)$$

The feature probabilities are smoothed by grouping the contexts $f_{i-2,1}^N, f_{i-1,1}^N, f_{i,1}^{k-1}$ into equivalence classes sharing the same parameter. The equivalence classes are defined by means of a binary decision tree (Quinlan, 1983; Breiman et al., 1984) which selects a subset of the contextual features which is found to be most relevant for predicting the value of the next feature. A similar smoothing technique is also used in the TreeTagger (Schmid, 1994b).

Why does the tag decomposition help with the sparse data problem? Consider a simple tagset with the two parts of speech *N* and *ADJ*, a Number feature with the values *sg* and *pl* and a Gender feature with the values *masc*, *fem*, and *neut*. The size of the tagset is $2 \cdot 2 \cdot 3 = 12$. A standard bigram tagger therefore has $12^2 = 144$ parameters. 12 of these parameters are fixed because the conditional probabilities $p(t|t')$ have to sum up to 1 for each context tag t' . If the decision tree decides (1) to predict the part of speech (*ADJ* or *N*) based on the preceding part of speech (ignoring the Number and Gender feature of the preceding tag) and (2) to predict the Number feature based on the current POS, the preceding POS and the preceding Number feature (but not the Gender feature), and (3) to predict the Gender feature based on the current POS, the preceding POS and the preceding Gender feature (but not the Number feature), then the number of parameters is $2 \cdot 2 + 2 \cdot 2 \cdot 2 + 2 \cdot 2 \cdot 3 \cdot 3 = 56$. Only 36 of these parameters are free compared with 132 for the bigram tagger.

1.3 Log-Linear Models

POS taggers based on Hidden Markov Models decompose the probability of a tag sequence into a product of contextual probabilities $p(t_i|t_{i-k}^{i-1})$ and lexical probabilities $p(w_i|t_i)$. The Maximum Entropy tagger presented in (Ratnaparkhi, 1996) uses a different decomposition into a product of the conditional probabilities of the tags t_i given the history h_i which consists of the k preceding tags, the current word, and the k preceding and following words:

$$\begin{aligned} \hat{t}_1^n &= \arg \max_{t_1^n} p(t_1^n|w_1^n) \\ &= \arg \max_{t_1^n} \prod_{i=1}^n p(t_i|w_1^n, t_1^{i-1}) \end{aligned}$$

$$\begin{aligned} &\approx \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | w_{i-k}^{i+k}, t_{i-k}^{i-1}) \\ &= \arg \max_{t_1^n} \prod_{i=1}^n p(t_i | h_i) \end{aligned}$$

The conditional probabilities $p(t_i | w_{i-k}^{i+k}, t_{i-k}^{i-1})$ are too specific to be directly estimated from training data: Given a context size $k = 2$, a tagset with 50 tags, and a lexicon with just 10,000 words, there are $10,000^5 * 50^3 = 1.25 * 10^{25}$ parameters to be estimated. Instead, the parameters are estimated with a Maximum Entropy Model which defines the probability as a normalized product of feature weights:

$$\begin{aligned} p(t|h) &= \frac{1}{Z_h} \prod_{j=1}^K \alpha_j^{f_j(t,h)} \\ Z_h &= \sum_t \prod_{j=1}^K \alpha_j^{f_j(t,h)} \end{aligned}$$

The normalization constant Z_h guarantees that $p(t|h)$ is a probability distribution. Models which define a probability distribution in this form are called *log-linear models* due to the transformation:

$$\prod_{j=1}^K \alpha_j^{f_j(t,h)} = e^{\sum_{j=1}^K \beta_j f_j(t,h)} \quad \text{with } \beta_j = \log \alpha_j$$

The features f_j are binary valued functions. The value is 1 if the feature condition is satisfied by the tag-history pair and 0 otherwise. An example is the following feature which checks whether the current word ends with the suffix *ing*. (“VBG” is the Penn treebank tag for gerund verbs.)

$$f_j(t_i, h_i) = \begin{cases} 1 & \text{if suffix}(w_i) = \text{“ing”} \ \& \ t_i = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

The features used by Ratnaparkhi’s tagger check the preceding POS tag t_{i-1} , the preceding POS bigram $t_{i-2}t_{i-1}$, the preceding word w_{i-1} , the following word w_{i+1} and the words two positions away (w_{i-2} and w_{i+2}). There are also features for the current word w_i , if w_i occurs 5 times or more in the training data. For words occurring less than 5 times, the model contains features which examine the prefixes and suffixes of length up to 5. Features occurring less than 10 times in the training data are eliminated because they are too unreliable. The features weights α_j are trained with *Generalized Iterative Scaling* (Darroch and Ratcliff, 1972).

Compared to unlexicalized Hidden Markov Models, the maximum entropy taggers use more information, in particular information about the surrounding words, and compared with lexicalized HMMs, the number of parameters is much smaller. At least for English, MaxEnt taggers seem to perform better than HMM taggers. Other POS taggers which are based on log-linear models have been presented in (Lafferty et al., 2001) and (Collins, 2002).

1.4 Classification-Based Approaches

Classification-based POS taggers compute the most likely POS tag for each word individually instead of optimizing the likelihood of the whole tag sequence, as HMM and MaxEnt taggers do. The advantage is that standard classification algorithms can be applied. Giménez and Màrquez (2004) presented a POS tagger which uses support vector machines (SVMs) to assign POS tags. The task of the SVM is to predict the POS tag (= class) of a word based on a set of features describing the word and the context. These features are similar to the features used by Ratnaparkhi's tagger. They include (a) the words, word bigrams, and word trigrams within a window of \pm three words around the currently tagged word, (b) the POS tags, POS bigrams, and POS trigrams in the current window, and (c) information about suffixes, prefixes, capitalization, hyphenation etc. of unknown words. The disambiguated part-of-speech tags of the current and the following words are not available, yet (unless the tagger operates on top of a baseline tagger). Therefore, the POS features encode the set of possible tags rather than a single tag for these words. Giménez and Màrquez (2004) compared their SVM tagger with the state-of-the-art trigram tagger of Brants (2000) and report a higher accuracy.

Other classification algorithms applied to POS tagging are neural networks (Benello et al., 1989; Nakamura et al., 1990; Schmid, 1994a), decision trees (Black et al., 1992; Màrquez and Padró, 1997), the Winnow algorithm (Roth and Zelenko, 1998), Boosting (Abney et al., 1999), and memory-based learning (Daelemans et al., 1996). Classification-based POS taggers can also be used to correct the errors of a baseline tagger, similarly to the Brill tagger which is described in the next section.

1.5 Transformation-Based Learning

Brill (1992) presented a POS tagger which learns symbolic correction rules from a training corpus in order to improve the performance of a baseline tagger. The rules are of the form: Change tag X to Y if condition C is satisfied. The rule conditions may check (a) the POS tags of the two preceding and the two following words, (b) tag pairs consisting of the two preceding tags, or the two following tags, or the preceding and the following tag, (c) whether the current word or the preceding word is capitalized, (d) whether one of the two (or three) preceding (or following) tags is some tag t .

The learning algorithm computes the performance gain of each possible rule by counting how often the rule leads to an error correction in the baseline tagging of the training corpus and how often it introduces a new error. The rule with the largest difference between the number of corrections and error insertions is selected. The output of this rule becomes the new baseline and the training algorithm is repeated to select the next rule. The learning algorithm terminates when the performance gain is below some threshold. The tagger applies the rules one after the other. The initial baseline tagger simply assigns the most likely tag to each word without considering the context. Unknown words are tagged as proper names if they are capitalized. The tag of an unknown lower-case

word depends on the last three characters. A word ending in *ous* e.g. would be tagged as an adjective.

1.6 Rule-Based Tagging

All POS taggers considered so far, learn the tagging task directly from training data. In contrast to them, the very first part-of-speech taggers, which were developed at a time when manually annotated training data was not available, applied manually developed disambiguation rules (Klein and Simmons, 1963; Greene and Rubin, 1971). A more recent approach along these lines is described in (Tapanainen and Voutilainen, 1994). The tagger is based on the constraint grammar framework (Karlsson, 1990). The basic idea is to start with a sentence which was ambiguously tagged with tag candidates by looking up the possible part-of-speech tags for each word in a dictionary. Incorrect and very unlikely tag candidates are then eliminated from the candidate sets by a set of rules of the form “Delete tag X if it occurs in the context C ”. The context is specified with a regular expression over POS sequences. Using 1,300 disambiguation rules, the average tag ambiguity is reduced to about 1.02 - 1.04 while retaining about 99.5 % of the correct tags. In order to produce fully disambiguated output, the remaining ambiguity can be eliminated with a statistical tagger.

1.7 Multilinguality

Most part-of-speech taggers have been developed for English tagging, but many of them have also been applied to other languages. Rule-based taggers require the development of a set of disambiguation rules for each new language. They also need a dictionary containing the set of possible part-of-speech tags for the known word forms and POS guessing rules for unknown words.

The unsupervised EM training of HMM taggers requires at least a dictionary and an unlabeled training corpus. The other taggers which are trained on labeled data need a manually annotated training corpus. How much data the tagger needs for training depends on the size of the tagset, the tagging algorithm, the required accuracy, and other factors. If a dictionary is available, it usually helps to use this information to restrict the set of possible tags considered by the tagger. The developer of the tagset has to take care that the number of tags is not too large, that the individual tags are frequent enough to allow reliable parameter estimation, and that it is possible to disambiguate the tags based on lexical information and the local context.

It is possible to speed up the annotation of a training corpus by pre-annotating the data with a part-of-speech tagger. The part-of-speech tagger is trained on the data which is already annotated and is retrained in regular intervals. As the size of the training corpus grows, the pre-tagging becomes increasingly accurate and the effort for manual correction decreases. A disadvantage of this method is the bias towards the tags

assigned by the tagger. Data created with this method should not be used for evaluation purposes.

Bibliography

- Abney, S., Schapire, R. E., and Singer, Y. (1999). Boosting applied to tagging and PP attachment. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, College Park, MD.
- Baum, L. E. (1972). An inequality and association maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3:1–8.
- Benello, J., Mackie, A. W., and Anderson, J. A. (1989). Syntactic category disambiguation with neural networks. *Computer Speech and Language*, 3:203–217.
- Black, E., Jelinek, F., Lafferty, J., Mercer, R., and Roukos, S. (1992). Decision tree models applied to the labeling of text with parts of speech. In *Proceedings of the DARPA Speech and Natural Language Workshop*, Arden House, NY.
- Brants, T. (2000). TnT - a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000*, Seattle, WA.
- Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Pacific Grove CA.
- Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, Trento, Italy.
- Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In Joshi, A. and Palmer, M., editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco. Morgan Kaufmann Publishers.
- Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, pages 136–143.
- Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with Perceptron algorithms. In *Proceedings of the Third Conference on Empirical Methods in Natural Language Processing*, pages 1–8, Philadelphia, PA.

- Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 133–140, Trento, Italy.
- Daelemans, W., Zavrel, J., Berck, P., and Gillis, S. (1996). Mbt: A memory-based part of speech tagger-generator. In Ejerhed, E. and Dagan, I., editors, *Proceedings of the Fourth Workshop on Very Large Corpora*, pages 14–27, Copenhagen, Denmark.
- Darroch, J. and Ratcliff, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480.
- Elworthy, D. (1994). Does Baum-Welch re-estimation help taggers? In *Proceedings of the 4th Conference on Applied Natural Language Processing*, pages 53–58, Stuttgart, Germany.
- Giménez, J. and Màrquez, L. (2004). SVMTool: A general POS tagger generator based on support vector machines. In *Proceedings of the IV International Conference on Language Resources and Evaluation (LREC'04)*, pages 43–46, Lisbon, Portugal.
- Greene, B. and Rubin, G. (1971). Automatic grammatical tagging of English. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island.
- Karlsson, F. (1990). Constraint grammar as a framework for parsing running text. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING 1990)*, pages 168–173, Helsinki, Finland.
- Kempe, A. (1993). A probabilistic tagger and an analysis of tagging errors. Technical report, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart.
- Klatt, S. (2002). Combining a rule-based tagger with a statistical tagger for annotating German texts. In Buseman, S., editor, *KONVENS 2002. 6. Konferenz zur Verarbeitung natürlicher Sprache*, Saarbrücken, Germany.
- Klein, S. and Simmons, R. (1963). A computational approach to grammatical coding of english words. *JACM*, 10:334–337.
- Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-01*, pages 282–289.
- Lee, S.-Z., ichi Tsujii, J., and Rim, H.-C. (2000). Lexicalized hidden Markov models for part-of-speech tagging. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, pages 481–487, Saarbrücken, Germany.
- Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. The MIT Press, Cambridge, Ma.

- Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.
- Màrquez, L. and Padró, L. (1997). A flexible POS tagger using an automatically acquired language model. In *Proceedings of the 35th Annual Meeting of the ACL*, pages 238–245, Madrid, Spain.
- Nakamura, M., Maruyama, K., Kawabata, T., and Shikano, K. (1990). Neural network approach to word category prediction for English texts. In Karlgren, H., editor, *Proceedings of the 13th International Conference on Computational Linguistics (COLING 1990)*, pages 213–218, Helsinki, Finland.
- Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An artificial intelligence approach*, pages 463–482. Morgan Kaufmann, San Mateo, CA.
- Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.
- Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*. University of Pennsylvania.
- Roth, D. and Zelenko, D. (1998). Part of speech tagging using a network of linear separators. In *Proceedings of ACL-COLING'98*, pages 1136–1142, Montreal, Canada.
- Schmid, H. (1994a). Part-of-speech tagging with neural networks. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING 1994)*, pages 172–176, Kyoto, Japan.
- Schmid, H. (1994b). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK.
- Schmid, H. (1995). Improvements in part-of-speech tagging with an application to German. In *Proceedings of the ACL SIGDAT-Workshop*, pages 47–50.
- Skut, W., Brants, T., Krenn, B., and Uszkoreit, H. (1998). A linguistically interpreted corpus of German newspaper text. In *Proceedings of the 10th European Summer School in Logic, Language and Information (ESSLLI'98), Workshop on Recent Advances in Corpus Annotation*.
- Tapanainen, P. and Voutilainen, A. (1994). Tagging accurately - don't guess if you know. In *Proceedings of the 4th Conference on Applied Natural Language Processing*, pages 47–52, Stuttgart, Germany. Association for Computational Linguistics, Morgan Kaufmann.

- Weischedel, R., Meteer, M., Schwartz, R., Ramshaw, L., and Palmucci, J. (1993). Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2):359–382.