Abstract

We present a new part-of-speech tagger that demonstrates the following ideas: (i) explicit use of both preceding and following tag contexts via a dependency network representation, (ii) broad use of lexical features, including jointly conditioning on multiple consecutive words, (iii) effective use of priors in loglinear models, and (iv) fine-grained modeling of linguistic and unknown word features. Using these ideas together, the resulting tagger gives a 97.21% accuracy on the Penn Treebank WSJ, an error reduction of 4% on the best previous single automatically learned tagging result.

1 Introduction

Almost all approaches to sequence problems such as part-of-speech tagging take a unidirectional approach to conditioning inference along the sequence. Regardless of whether one is using HMMs, maximum entropy conditional sequence models, or other techniques like decision trees, most systems work in one direction through the sequence (normally left to right, but occasionally right to left, e.g., Church (1988)). There are a few exceptions, such as Brill’s transformation-based learning (Brill, 1995), but most of the best known and most successful approaches of recent years have been unidirectional.

Most sequence models can be seen as chaining together the scores or decisions from successive local models to form a global model for an entire sequence. Clearly the identity of a tag is correlated with both past and future tags’ identities. However, in the unidirectional (causal) case, only one direction of influence is explicitly considered at each local point. For example, in a left-to-right first-order HMM, the current tag $t_0$ is predicted based on the previous tag $t_{-1}$ (and the current word). The backward interaction between $t_0$ and the next tag $t_{+1}$ shows up implicitly later, when $t_{+1}$ is generated in turn. While unidirectional models are therefore able to capture both directions of influence, there are good reasons for suspecting that it would be advantageous to make information from both directions explicitly available for conditioning at each local point in the model: (i) because of smoothing or and interactions with other modeled variables, features, terms like $P(t_0|t_{+1}, \ldots)$ might give a sharp estimate of $t_0$ even when terms like $P(t_{+1}|t_0, \ldots)$ do not, and (ii) jointly considering the left and right context together might be especially revealing. In this paper we exploit this idea, using dependency networks, with a series of local conditional loglinear (aka maximum entropy or multiclass logistic regression) models as one way of providing efficient bidirectional inference.

Secondly, while all taggers use lexical information, and, indeed, it is well-known that lexical probabilities are much more revealing than tag sequence probabilities (Charniak et al., 1993), most taggers make quite limited use of lexical probabilities (compared with, for example, the bilexical probabilities commonly used in current statistical parsers). While modern taggers may be more principled than the classic CLAWS tagger (Marshall, 1987), they are in some respects inferior in their use of lexical information: CLAWS, through its IDIOMTAG module,

1Rather than subscripting all variables with a position index, we use a hopefully clearer relative notation, where $t_0$ denotes the current position and $t_{-n}$ and $t_{+n}$ are left and right context tags, and similarly for words.
categorically captured many important, correct taggings of frequent idiomatic word sequences. In this work, we incorporate appropriate multiword feature templates so that such facts can be learned and used automatically by the model.

Having expressive templates leads to a large number of features, but we show that by suitable use of a prior (i.e., regularization) in the conditional loglinear model – something not used by previous maximum entropy taggers – many such features can be added with an overall positive effect on the model. Indeed, as for the voted perceptron of Collins (2002), we can get performance gains by reducing the support threshold for features to be included in the model. Combining all these ideas, together with a few additional handcrafted unknown word and linguistic features, gives us a part-of-speech tagger with a per-position tag accuracy of x%, and a whole-sentence correct rate of y% on Penn Treebank WSJ data. This is the best automatically learned part-of-speech tagging result known to us, representing an error reduction of y% on the model presented in Collins (2002), using the same data splits, and a larger error reduction of 11% from the more similar best previous loglinear model in Toutanova and Manning (2000).

2 Bidirectional Dependency Network Tagging Models

When building probabilistic models for tag sequences, we often decompose the global probability of sequences using a directed graphical model (e.g., an HMM (Brants, 2000) or a conditional Markov model (CMM) (Ratnaparkhi, 1996)). In such models, the probability assigned to a tagged sequence of words \( x = \langle t_i, w_i \rangle \) is the product of a sequence of local portions of the graphical model, one from each time slice. For example, in the left-to-right CMM shown in figure 1(a),

\[
P(t, w) = \prod_i P(t_i | t_{i-1}, w_i)
\]

where the replicated structure is a local model of \( P(t_0 | t_{-1}, w_0) \).\(^2\) Of course, if there are too many

\[\text{(a) Left-to-Right CMM}\]
\[
\begin{array}{c}
  t_1 \\
  \vdots \\
  t_n \\
\end{array}
\]
\[
\begin{array}{c}
  w_1 \\
  \vdots \\
  w_n \\
\end{array}
\]

\[\text{(b) Right-to-Left CMM}\]
\[
\begin{array}{c}
  t_1 \\
  \vdots \\
  t_n \\
\end{array}
\]
\[
\begin{array}{c}
  w_1 \\
  \vdots \\
  w_n \\
\end{array}
\]

\[\text{(c) Bidirectional Dependency Network}\]

Figure 1: Dependency networks: (a) the (standard) left-to-right first-order CMM, (b) the (reversed) right-to-left CMM, and (c) the bidirectional dependency network.

conditioned quantities, these local models may have to be estimated in some sophisticated way; it is typical in tagging to populate these models with little maximum entropy models. For example, we might populate a model for \( P(t_0 | t_{-1}, w_0) \) with a maxent model of the form:

\[
P(t_0 | t_{-1}, w_0) = \frac{\exp(\lambda_{t_0,t_{-1}} + \lambda_{t_0,w_0})}{\sum_{t'_{-1}} \exp(\lambda_{t'_{-1},t_{-1}} + \lambda_{t'_0,w_0})}
\]

In this case, the \( w_0 \) and \( t_{-1} \) can have joint effects on \( t_0 \), but there are not joint features involving all three variables (though there could have been such features). We say that this model uses the feature templates \( \langle t_0, t_{-1} \rangle \) (previous tag features) and \( \langle t_0, w_0 \rangle \) (current word features).

Clearly, both the preceding tag \( t_{-1} \) and following tag \( t_{+1} \) carry useful information about a current tag \( t_0 \). Unidirectional models do not ignore this influence; in the case of a left-to-right CMM, the influence of \( t_{-1} \) on \( t_0 \) is explicit in the \( P(t_0 | t_{-1}, w_0) \) local model, while the influence of \( t_{+1} \) on \( t_0 \) is implicit in the local model at the next position (via \( P(t_{+1} | t_0, w_{+1}) \)). The situation is reversed for the right-to-left CMM in figure 1(b).

From a seat-of-the-pants machine learning perspective, when building a classifier to label the tag at a certain position, the obvious thing to do is to

\(^2\)Throughout this paper we assume that enough boundary symbols always exist that we can ignore the differences which would otherwise exist at the initial and final few positions.
explicitly include in the local model all predictive features, no matter on which side of the target position they lie. Moreover, there are two good formal reasons to expect that a model explicitly conditioning on both sides at each position, like figure 1(c) could be advantageous. First, because of smoothing effects and interaction with other conditioning features (like the words), left-to-right factors like $P(t_0|t_{-1}, w_0)$ do not always suffice when $t_0$ is implicitly needed to determine $t_{-1}$. For example, consider a case of observation bias (Klein and Manning, 2002) for a first-order left-to-right CMM. The word *to* has only one tag (TO) in the PTB tagset. The TO tag is often preceded by nouns, but rarely by modal tags (MD). In a sequence *will* *to* *fight*, that trend indicates that *will* should be a noun rather than a modal verb. However, that effect is completely lost in a CMM like (a): $P(t_{\text{will}}|\text{will}, \langle \text{start} \rangle)$ prefers the modal tagging, and $P(\text{TO}|t_0, t_{\text{will}})$ is roughly 1 regardless of $t_{\text{will}}$. While the model has an arrow between the two tag positions, that path of influence is severed.\(^3\) The same problem exists in the other direction. If we use the symmetric right-to-left model, *fight* will receive its more common noun tagging by symmetric reasoning. However, the bidirectional model (d) discussed in the next section allows both directions to be available for conditioning at all locations, using replicated models of $P(t_0|t_{-1}, t_{+1}, w_0)$, and will be able to get this example correct.\(^4\) Additionally, making both directions available simultaneously, as in (d), allows the preceding and following tags to be conditioned on jointly, which proves to be very useful.

### 2.1 Semantics of Dependency Networks

While the structures in figure 1(b) and (c) are well-understood graphical models with well-known semantics, figure 1(d) is not a standard Bayes’ net,

---

\(^3\)Despite names like “label bias” (Lafferty et al., 2001) or “observation bias”, these effects are really just unwanted explaining-away effects, where two nodes which are not actually in causal competition have been modeled as if they were.

\(^4\)The effect of indirect influence being weaker than direct influence is more pronounced for conditionally structured models, but is potentially an issue even with a simple HMM. The probabilistic models for basic left-to-right and right-to-left HMMs with emissions on their states can be shown to be equivalent using Bayes’ rule on the transitions, provided start and end symbols are modeled. However, this equivalence is violated in practice by the addition of smoothing.

---

![Figure 2: Simple dependency nets](image)

While the structures in figure 1(b) and (c) are well-understood graphical models with well-known semantics, figure 1(d) is not a standard Bayes’ net precisely because the graph has cycles. Rather, it is a more general dependency network (Heckerman et al., 2000). Each node represents a random variable along with a local conditional probability mode of that variable, conditioned on all incoming neighbors. In this sense, the semantics are the same as for standard Bayes’ nets. However, because the graph is cyclic, the net does not correspond to a proper factorization of a large joint probability estimate into local conditional factors. Consider the two-node cases shown in figure 2. Formally, for the net in (a), we can write:

$$P(a, b) = P(a) P(b|a)$$

For (b) we write

$$P(a, b) = P(b) P(a|b)$$

However, in (c), the nodes A and B carry the information $P(a|b)$ and $P(b|a)$ respectively. The chain rule doesn’t allow us to reconstruct $P(a, b)$ by multiplying these two quantities. Under appropriate conditions, we could reconstruct $P(a, b)$ from these quantities using Gibbs sampling, and, in general, that is the best we can do. However, while reconstructing the joint probabilities from these local conditional probabilities may be difficult, estimating the local probabilities themselves is no harder than it is for acyclic models: we take observations of the local environments and use any maximum likelihood estimation method we desire. In our experiments, we used local maxent models, but if the event space allows, (smoothed) relative counts would do.

### 2.2 Inference for Linear Dependency Networks

Cyclic or not, we can view the product of local probabilities from a dependency network as a score:

$$\text{score}(x) = \prod_i P(x_i|Pa(x_i))$$

---

![Diagram](image)

(a) (b) (c)
function bestScore()
    return bestScoreSub(n + 2, "end", "end", "end");

function bestScoreSub(i + 1, \(t_{i-1}, t_i, t_{i+1}\))
% memoization
if (cached(i + 1, \(t_{i-1}, t_i, t_{i+1}\)))
    return cache(i + 1, \(t_{i-1}, t_i, t_{i+1}\));
% left boundary case
if (i == -1) return 0;
else
    return max\(t_{i+2}\) bestScoreSub(i, \(t_{i-2}, t_{i-1}, t_i\))
        * \(P(t_i|t_{i-1}, t_{i+1}, w_i)\);

Figure 3: Pseudocode for polynomial inference in the first-order bidirectional CMM (memoized).

where \(P(a|x_i)\) are the nodes with arcs to the node \(x_i\).

In the case of an acyclic model, this score will be the joint probability of the event \(x, P(x)\). In the general case, it will not be. However, we can still ask for the event, in this case the tag sequence, with the highest score. For dependency networks like those in figure 1, an adaptation of the Viterbi algorithm can be used to find the maximizing sequence in polynomial time. Figure 3 gives pseudocode for the concrete case of the network in figure 1(d); the general case is similar, and is in fact just a max-plus version of standard inference algorithms for Bayes’ nets (cite). In essence, there is no difference between inference on that network and a second-order left-to-right CMM or HMM. The only difference is that, when the Markov window is at a position \(i\), rather than receiving the score for \(P(t_i|t_{i-1}, t_{i-2}, w_i)\), one receives the score for \(P(t_{i-1}|t_i, t_{i-2}, w_{i-1})\).

There are some foundational issues worth mentioning. As discussed previously, the maximum scoring sequence need not be the sequence with maximum likelihood according to the model. There is therefore a worry with these models about a kind of “collusion” where the model locks onto conditionally consistent but jointly unlikely sequences. Consider the two-node network in figure 2(c). If we have the following distribution of observations (in the form \(ab\)) \(\{11, 11, 11, 12, 21, 33\}\), then clearly the most likely state of the network is 11. However, the score of 11 is \(P(a = 1|b = 1)P(a = 1|b = 1) = 3/4 \times 3/4 = 9/16\), while the score of 33 is 1. An additional related problem is that the per-state training loss does not bound the per-state training set error. Consider the following training set, for the same network, with each entire data point considered as a label: \(\{11, 22\}\). The relative-frequency model assigns score 1 to both training examples, but cannot do better than 50% accuracy in regenerating the training data labels. Foundational issues are further discussed in Heckerman et al. (2000); in practice, these issues do not prevent the system from performing well.

It is useful to contrast this framework with the conditional random fields of Lafferty et al. (2001). The CRF approach uses similar local features, but rather than chaining together local models, they construct a single, globally normalized model. The principle advantage of the dependency network approach is that advantageous bidirectional effects can be obtained without the extremely expensive global training required for CRFs.

To summarize, we draw a dependency network in which each node has as neighbors all the other nodes that we would like to have influence it directly. Each node’s neighborhood is then considered in isolation and a local model is trained to maximize the conditional likelihood over the training data of that node given its parents. At test time, the sequence with the highest product of local conditional scores is calculated and returned. We can always find the exact maximizing sequence, but only in the case of an acyclic net will it be guaranteed to also be the maximum likelihood sequence.

3 Experiments

The part of speech tagged data used in our experiments is the Wall Street Journal data from Penn Treebank III (Marcus et al., 1994). We extracted tagged sentences from the parse trees.\(^5\) We split the data into training, development, and test sets as in (Collins, 2002). Table 1 lists characteristics of the three splits.\(^6\) Except where indicated for the model

\(^5\)Note that these tags (and sentences) are not identical to those obtained from the tagged/pos directories of the same disk: hundreds of tags in the RB/RP/IN set were changed to be more consistent in the parsed/mrg version. Maybe we were the last to discover this, but we’ve never seen it in print.

\(^6\)Tagger results are only comparable when tested not only on the same data and tagset, but with the same amount of training data. Brants (2000) illustrates very clearly how tagging performance increases as training set size grows, largely because the
<table>
<thead>
<tr>
<th>Data Set</th>
<th>Sect’s</th>
<th>Sent.</th>
<th>Tokens</th>
<th>Unkn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>0–18</td>
<td>38,219</td>
<td>912,344</td>
<td>0</td>
</tr>
<tr>
<td>Develop</td>
<td>19–21</td>
<td>5,527</td>
<td>131,768</td>
<td>4,467</td>
</tr>
<tr>
<td>Test</td>
<td>22–24</td>
<td>5,462</td>
<td>129,654</td>
<td>3,649</td>
</tr>
</tbody>
</table>

Table 1: Data set splits used.

BEST, all results are on the development set.

One innovation in our reporting of results is that we present whole-sentence accuracy numbers as well as the traditional per-tag accuracy measure (over all tokens, even unambiguous ones). As taggers get better, we would argue that presenting whole-sentence accuracies makes a lot of sense. This is the quantity that most sequence models attempt to maximize (and has been motivated over doing per-state optimization as being more useful for subsequent linguistic processing: one wants to find a coherent sentence interpretation). Further, while some tag errors matter much more than others (and a more nuanced approach could define an appropriate loss function), to a first cut getting a single tag wrong in many of the more common ways (e.g., proper noun vs. common noun; noun vs. verb) would lead to errors in a subsequent processor such as an information extraction system or a parser that would greatly degrade results for the entire sentence. Finally, the fact that the measure has much more dynamic range has some appeal when reporting tagging results.

The per-state models in this paper are log-linear models, building upon the models in (Ratnaparkhi, 1996) and (Toutanova and Manning, 2000), though some models are in fact strictly simpler. The features in the models are defined using templates; there are different templates for rare words aimed at learning the correct tags for unknown words.\(^7\) We present the results of four classes of experiments: experiments with directionality, experiments with lexicalization, experiments with smoothing, and experimentation with more sophisticated linguistic features.

percentage of unknown words decreases while system performance on them increases (they become increasingly restricted as to wordclass).

\(^7\)Except where otherwise stated, a count cutoff of 2 was used for common word features and 35 for rare word features (templates need a support set strictly greater in size than the cutoff before they are included in the model).

3.1 Experiments with Directionality

It is natural to expect that the part of speech tag of the current word can be better predicted by simultaneously considering the tags of both preceding and following words. We discuss experiments using log-linear CMMs to populate nets with various structures, exploring the relative value of neighboring words’ tags. Table 2 lists the networks discussed in this section. All networks have the same vertical feature templates: \(\langle t_0, w_0 \rangle\) features for known words and various \(\langle t_0, \sigma(w_{1:n}) \rangle\) word signature features for all words, known or not, including spelling and capitalization features (see section 5).

Just this vertical conditioning (baseline) gives an accuracy of 93.69% as a baseline.\(^8\) Conditioning on the previous tag as well (model L, \(\langle t_0, t_{-1} \rangle\) features) gives 95.79%. The reverse, model R, using the next tag instead, is slightly inferior at 95.14%. Model L+R, using both tags simultaneously (but with only the individual-direction features) gives a much better accuracy of 96.57%. Since this model has roughly twice as many tag-tag features, the fact that it outperforms the unidirectional models is not by itself compelling evidence for using bidirectional networks. However, it also outperforms model L+L\(_2\) which adds the \(\langle t_0, t_{-2} \rangle\) second-previous word features instead of next word features, which gives only 96.05% (and R+R\(_2\) gives 95.25%). Therefore, if one wishes to condition on two neighboring nodes (using two sets of 2-tag features), the symmetric bidirectional model is superior.

High-performance taggers typically also include joint three-tag counts in some way, either as tag trigrams (Brants, 2000) or tag-triple features (Ratnaparkhi, 1996; Toutanova and Manning, 2000). Models LL, RR, and CR use only the vertical features and a single set of tag-triple features: the left tags \(t_{-2}, t_{-1}\) and \(t_0\), right tags \(t_{0}, t_{+1}, t_{+2}\), or centered tags \(t_{-1}, t_{0}, t_{+1}\) respectively. Again, with roughly equivalent feature sets, the left context is better than the right, and the centered context is better than the left but not much.

\(^8\)Charniak et al. (1993) noted that such a simple model got 90.25%, but this was with no unknown word model beyond a prior distribution over tags. Abney et al. (1999) raise this baseline to 92.34%, and with our sophisticated unknown word model, it gets even higher. The large number of unambiguous tokens make the baseline for this task high, while substantial annotator noise creates an unknown upper bound on the task.
Table 2: Tagging accuracy with different sequence feature templates. †All models include the same vertical word-tag features, though the baseline uses a lower cutoff for these features.

<table>
<thead>
<tr>
<th>Model</th>
<th>Feature Templates</th>
<th>Features</th>
<th>Sentence Accuracy</th>
<th>Token Accuracy</th>
<th>Unknown Word Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>⌀</td>
<td></td>
<td>56.805</td>
<td>26.74%</td>
<td>93.69%</td>
</tr>
<tr>
<td>L</td>
<td>(\langle t_0, t_{-1} \rangle)</td>
<td></td>
<td>27.474</td>
<td>41.89%</td>
<td>95.79%</td>
</tr>
<tr>
<td>R</td>
<td>(\langle t_0, t_{+1} \rangle)</td>
<td></td>
<td>27.648</td>
<td>36.31%</td>
<td>95.14%</td>
</tr>
<tr>
<td>L+L</td>
<td>(\langle t_0, t_{-1} \rangle, \langle t_0, t_{-2} \rangle)</td>
<td></td>
<td>32.935</td>
<td>44.04%</td>
<td>96.05%</td>
</tr>
<tr>
<td>R+R</td>
<td>(\langle t_0, t_{+1} \rangle, \langle t_0, t_{+2} \rangle)</td>
<td></td>
<td>33.423</td>
<td>37.20%</td>
<td>95.25%</td>
</tr>
<tr>
<td>L+R</td>
<td>(\langle t_0, t_{-1} \rangle, \langle t_0, t_{+1} \rangle)</td>
<td></td>
<td>32.610</td>
<td>49.50%</td>
<td>96.57%</td>
</tr>
<tr>
<td>LL</td>
<td>(\langle t_0, t_{-1} \rangle, \langle t_0, t_{-2} \rangle)</td>
<td></td>
<td>45.332</td>
<td>44.60%</td>
<td>96.10%</td>
</tr>
<tr>
<td>RR</td>
<td>(\langle t_0, t_{+1} \rangle, \langle t_0, t_{+2} \rangle)</td>
<td></td>
<td>45.446</td>
<td>38.41%</td>
<td>95.40%</td>
</tr>
<tr>
<td>LR</td>
<td>(\langle t_0, t_{-1} \rangle, \langle t_0, t_{+1} \rangle)</td>
<td></td>
<td>45.478</td>
<td>49.30%</td>
<td>96.55%</td>
</tr>
<tr>
<td>L+LL+LLL</td>
<td>(\langle t_0, t_{-1} \rangle, \langle t_0, t_{-2} \rangle)</td>
<td>(\langle t_0, t_{-1} \rangle, \langle t_0, t_{-2} \rangle)</td>
<td>118.752</td>
<td>45.14%</td>
<td>96.20%</td>
</tr>
<tr>
<td>R+LR+LLR</td>
<td>(\langle t_0, t_{+1} \rangle, \langle t_0, t_{-1} \rangle, \langle t_0, t_{+1} \rangle)</td>
<td>(\langle t_0, t_{+1} \rangle, \langle t_0, t_{-1} \rangle, \langle t_0, t_{+1} \rangle)</td>
<td>115.790</td>
<td>51.69%</td>
<td>96.77%</td>
</tr>
<tr>
<td>L+LL+LR+RR+R</td>
<td>(\langle t_0, t_{-1} \rangle, \langle t_0, t_{-2} \rangle)</td>
<td>(\langle t_0, t_{-1} \rangle, \langle t_0, t_{+1} \rangle, \langle t_0, t_{-1} \rangle, \langle t_0, t_{+1} \rangle, \langle t_0, t_{+2} \rangle)</td>
<td>81.049</td>
<td>53.23%</td>
<td>96.92%</td>
</tr>
</tbody>
</table>

4 Lexicalization

Lexicalization has been the key factor in the advance of statistical parsing models. The only lexicalization consistently included in tagging models is the dependence of the part of speech tag of a word on the word itself. Words surrounding the current have been used in taggers, such as (Ratnaparkhi, 1996), Brill’s transformation based tagger (Brill, 1994) and the HMM model of Lee et al. (2000). Lee et al. (2000) find that lexicalization of word generation probabilities and tag transition probabilities leads to a significant improvement of an HMM-based model.

In maximum entropy models, joint features which look at surrounding words and their tags, as well as joint features of the current word and surrounding words are in principle straightforward additions, but have not been incorporated into previous models. We have found these features to be very useful. We explore here lexicalization both alone and in combination with preceding and following tag histories.

Table 3 shows the development set accuracy of several models with various lexical features. All models use the same rare word features as the models in Table 2.

The first row shows a baseline model using the current word only. The count cutoff for this feature was 0 in this model and the model in the next row.

As there are no tag sequence features in these models, the accuracy drops significantly if a higher cutoff is used (the token accuracy is only \(\approx 60\%\) if a cutoff of 2 is used).

The second row shows a model where a tag is decided solely by the three words centered at the tag position (3W). As far as we are aware, models of this sort have not been explored previously, but its accuracy is surprisingly high: despite having no sequence model at all, it is more accurate than a model which uses standard tag fourgram HMM features shown in Table 2, row 9.

The third and fourth rows show models with bi-directional tagging features. The third model (3W+TAGS) uses the same tag sequence features as the last model in Table 2 \(\langle t_0, t_{-1} \rangle, \langle t_{-1}, t_{-2} \rangle, \langle t_0, t_{-1}, t_{+1} \rangle, \langle t_0, t_{+1}, t_{+2} \rangle\) and current, previous, and next word. The last model has in addition the multi-lexical feature templates \(\langle t_0, w_{0}, t_{-1} \rangle, \langle t_0, w_{0}, t_{-2} \rangle, \langle t_0, w_{0}, t_{-1} \rangle, \langle t_0, w_{-1}, w_{0} \rangle, \) and \(\langle t_0, w_{0}, w_{+1} \rangle\). We call this model BEST. BEST has a token accuracy on the final test set of 97.21% and a sentence accuracy of 55.75%.

\(\text{Thede and Harper (1999) use } \langle t_{-1}, t_{0}, w_{0} \rangle \text{ templates in their “full-second order” HMM model, achieving an accuracy of 96.86%. Here we can add the opposite tiling and other additional features.}\)
5 Other Features

Also important to final tagger performance are details of smoothing, accurate modeling of unknown words, and so on.

5.1 Smoothing

With so many features in the model, overtraining is a distinct possibility when using pure maximum likelihood estimation. We avoid this by using a Gaussian prior (aka quadratic regularization or quadratic penalization) which resists high feature weights unless they produce great score gain. The regularized objective $F$ is:

$$ F(\lambda) = \prod_i \log(P_\lambda(t_i | w, t)) + \sum_{j=1}^n \frac{\lambda_j^2}{2\sigma^2} $$

Since we use a conjugate-gradient procedure to maximize the data likelihood, the addition of a penalty term is easily incorporated. Both the total size of the penalty and the partial derivatives with respect to each $\lambda_j$ are trivial to compute; these are added to the log-likelihood and log-likelihood derivatives, and the penalized optimization proceeds without further modification.

We have not extensively experimented with the value of $\sigma$ – which can even be set differently for different parameters or parameter classes. All the results in this paper use a constant $\sigma = 0.5$, so that the denominator disappears in the above expression. Experiments on a simple model with $\sigma$ made an order of magnitude higher or lower both resulted in worse performance than with $\sigma = 0.5$.

Our experiments show that quadratic regularization is very effective in improving the generalization performance of tagging models, mostly by increasing the number of features which could be usefully incorporated. The number of features used in our complex models – in the several hundreds of thousands, is extremely high in comparison with the data set size and the number of features used in other machine learning domains. We describe two sets of experiments aimed at comparing models with and without regularization. One is for a simple model with a relatively small number of features, and the other is for a model with a large number of features.

The usefulness of priors in maximum entropy models is not new to this work: Gaussian prior smoothing is advocated in Chen and Rosenfeld (2000), and used in all the stochastic LFG work (Johnson et al., 1999). However, until recently, its role and importance have not been widely understood. For example, Zhang and Oles (2001) attribute the perceived limited success of logistic regression for text categorization to a lack of use of regularization, and show that a properly regularized model slightly outperforms an SVM, while Komarek and Moore (2003) are still advocating the benefits of early stopping in the fitting process because of their use of unregularized models. At any rate, regularized conditional loglinear models have not previously been applied to the problem of producing a high quality part-of-speech tagger: Ratnaparkhi (1996), Toutanova and Manning (2000), and Collins (2002) all present unregularized models. Indeed, the result of Collins (2002) that including low support features helps a voted perceptron model but harms a maximum entropy model is undone once the weights of the maximum entropy model are regularized.

Table 3 shows results on the test set from two pairs of experiments. The first pair of models use common word templates $\langle t_0, w_0 \rangle$, $\langle t_0, t_{-1}, t_{-2} \rangle$ and the same rare word templates as used in the models in Table 2. The second pair of models use the same features as model BEST with a higher frequency cutoff of 5 for common word features.

For the first pair of models, the error reduction from smoothing is 4.4% overall and 13.2% on unknown words. For the second pair of models, the

<table>
<thead>
<tr>
<th>Model</th>
<th>Feature Templates</th>
<th>Features</th>
<th>Sentence Accuracy</th>
<th>Token Accuracy</th>
<th>Unknown Word Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASELINE</td>
<td>$\langle t_0, w_0 \rangle$</td>
<td>56,805</td>
<td>26.74%</td>
<td>93.69%</td>
<td>82.61%</td>
</tr>
<tr>
<td>3W</td>
<td>$\langle t_0, w_0 \rangle, \langle t_0, t_{-1} \rangle, \langle t_0, t_{-2} \rangle$</td>
<td>239,767</td>
<td>48.27%</td>
<td>96.57%</td>
<td>86.78%</td>
</tr>
<tr>
<td>3W+TAGS</td>
<td>tag sequences, $\langle t_0, w_0 \rangle, \langle t_0, t_{-1} \rangle, \langle t_0, t_{-2} \rangle$</td>
<td>263,160</td>
<td>53.83%</td>
<td>97.02%</td>
<td>88.05%</td>
</tr>
<tr>
<td>BEST</td>
<td>see text</td>
<td>443,131</td>
<td>54.97%</td>
<td>97.12%</td>
<td>88.02%</td>
</tr>
</tbody>
</table>

Table 3: Tagging accuracy with different lexical feature templates.
error reduction is even bigger: 8.7% overall. The especially large reduction in unknown word error reflects that, because penalties are effectively stronger for rare features than frequent ones, the presence of penalties increases the degree to which more general cross-word signature features (which apply to unknown words) are used, relative to word-specific sparse features (which do not apply to unknown words).

Secondly, use of regularization allows us to incorporate features with low support into the model while improving performance. Whereas Ratnaparkhi (1996) used feature support cutoffs to stop overfitting of the model, and Collins (2002) contends that including low support features harms a maximum entropy model, our results show that low support features are useful in a regularized maximum entropy model. Table 4 contrasts our results with those from Collins (2002). Since the models are not the same, the exact numbers are incomparable, but the difference in direction is important: in the regularized model, performance improves with the inclusion of low support features. Our results are for our model L (left context tag and lexical features only); this behavior is representative.

Finally, in addition to being significantly more accurate, smoothed models train much faster than unsmoothed ones, and do not benefit from early stopping. For example, the first smoothed model in Table 5 required 80 conjugate gradient iterations to converge (somewhat arbitrarily defined as a maximum difference of $10^{-4}$ in feature weights between iterations), while its corresponding unsmoothed model required 335 iterations, thus training was roughly 4 times slower. The second pair of models required 95 and 370 iterations respectively. As might be expected, unsmoothed models reach their highest generalization capacity long before convergence and accuracy on an unseen test set drops considerably with further iterations. This is not the case for smoothed models, as their test set accuracy increases almost monotonically with training iterations. Figure 4 shows a graph of training iterations versus accuracy for the second pair of models on the development set.

5.2 Unknown word features

Most of the models presented here use a set of unknown word features that we inherited from previous work, which include using character $n$-gram prefixes and suffixes (for $n$ up to 4), and detectors for a few other prominent features of words, such as capitalization, hyphens, and numbers. Doing error analysis on unknown words on a simple tagging model (with $\langle l_0, l_{-1} \rangle$, $\langle l_0, l_{-1}, l_{-2} \rangle$, and $\langle w_0, l_0 \rangle$ features) suggests several additional specialized feature training.

---

Table 4: Effect of changing common word feature support cutoffs (features with support $\leq$ cutoff are excluded from the model).

<table>
<thead>
<tr>
<th>Tagger</th>
<th>Min support</th>
<th>Accuracy</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Collins (2002)</td>
<td>0</td>
<td>96.60%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>96.72%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model L</td>
<td>1</td>
<td>96.97%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>96.93%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Tagging accuracy with and without quadratic regularization.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>45,532</td>
<td>44.62%</td>
<td>96.10%</td>
<td>86.35%</td>
</tr>
<tr>
<td>no</td>
<td>45,532</td>
<td>43.35%</td>
<td>95.92%</td>
<td>84.27%</td>
</tr>
<tr>
<td>yes</td>
<td>292,649</td>
<td>55.42%</td>
<td>97.20%</td>
<td>88.57%</td>
</tr>
<tr>
<td>no</td>
<td>292,649</td>
<td>51.20%</td>
<td>96.78%</td>
<td>87.10%</td>
</tr>
</tbody>
</table>

Figure 4: Accuracy vs. Training Iterations for a Smoothed and Unsmoothed Model.

---

10 With current hardware, this is still an important difference: our largest models require about 25 minutes per iteration during training.

11 In practice one notices a fair bit of wiggling in the curve, but the trend remains upward even beyond our chosen convergence point.
tures that can useful improve performance. By far the most significant was a crude company name detector which marked capitalized words followed within 3 words by a company name suffix like Co. or Inc. This suggests (reversing the most common order of processing in pipelined systems!) that further gains could be made by incorporating a good named entity recognizer as a preprocessor to the tagger, and is a good example of something that can only be done when using a conditional model. Minor gains came from a few additional features: an allcaps feature, and a conjunction feature of words that are capitalized and have a digit and a dash in them (such words are normally common nouns, such as CFC-12 or F/A-18). Together with the larger templates, these features contribute to our unknown word accuracies being higher than for those of previously reported taggers.

6 Conclusion

We have shown how broad feature use, combined with appropriate model regularization produces a superior level of tagger performance. While experience suggests that the final accuracy number presented here could be slightly improved upon by classifier combination, it is worth noting that not only is this tagger better than any previous single tagger, but it also appears to outperform Brill and Wu (1998), the best-known combination tagger (they report an accuracy of 97.16% over the same WSJ data, but using a larger training set, which should favor them).

References


