Bidirectional Inference with the Easiest-First Strategy for Tagging Sequence Data

Abstract

This paper presents a bidirectional inference algorithm for sequence labeling problems such as POS tagging, named entity recognition and shallow parsing (chunking). The algorithm can enumerate all possible decomposition structures and find the highest probability sequence together with the corresponding decomposition structure in polynomial time. We also present an efficient decoding algorithm based on the easiest-first strategy, which gives comparably good performance to full bidirectional inference with significantly lower computational cost. Experimental results of POS tagging and text chunking show that the proposed bidirectional inference methods consistently outperform unidirectional inference methods and bidirectional MEMMs give comparable performance to that achieved by state-of-the-art learning algorithms including kernel support vector machines.

1 Introduction

The task of labeling sequence data such as part-of-speech tagging, shallow parsing (chunking) and named entity recognition is one of the most important tasks in natural language processing.

Conditional random fields (CRFs) (Lafferty et al., 2001) have recently attracted much attention because they are free from so-called label bias problems which reportedly degrade the performance of sequential classification approaches like maximum entropy markov models (MEMMs).

Although sequential classification approaches could suffer from label bias problems, they have several advantages over CRFs. One is the efficiency of training. CRFs need to perform dynamic programming over the whole sentence in order to compute feature expectations in each iteration of numerical optimization. Training, for instance, second-order CRFs using a rich set of features can require prohibitive computational resources. This is presumably the reason why CRFs have not yet succeeded in achieving the best accuracy on English POS tagging.

Another advantage is that one can employ a variety of machine learning algorithms as the local classifier. In the machine learning community, there is huge amount of work about developing classification algorithms that have high generalization performance. Being able to incorporate such state-of-the-art machine learning algorithms is a big advantage. Indeed, sequential classification approaches with kernel support vector machines offer competitive performance in POS tagging and chunking (Gimenez and Marquez, 2003; Kudo and Matsumoto, 2001).

One obvious way to improve the performance of sequential classification approaches is to enrich the information that the local classifiers can have. In standard decomposition techniques, the local classifiers cannot have the information about future tags (e.g. the right-side tags in left-to-right decoding), which would be very helpful information for them to predict the tag of the target word. To make use of the information about future tags, Toutanova pro-

posed a tagging algorithm based on bidirectional dependency networks (Toutanova et al., 2003) and achieved the best accuracy on POS tagging on the Wall Street Journal corpus. As they pointed out in their paper, however, the method potentially suffers from "collusion" effects which make the model lock onto conditionally consistent but jointly unlikely sequences. We will later show that the effects make the method less effective in another sequence labeling task (i.e. chunking).

In this paper we propose an alternative way of making use of future tags. Our inference method considers all possible ways of decomposition and chooses the "best" decomposition, so the information about future tags is used in appropriate situations. We also present a deterministic version of the inference method and show their effectiveness with experiments of English POS tagging and chunking, using standard evaluation sets.

2 Bidirectional Inference

The task of labeling sequence data is to find the sequence of tags $t_1...t_n$ that maximizes the following probability given the observation $o = o_1...o_n$

$$P(t_1...t_n|o). \quad (1)$$

Observations are typically words and their lexical features in the task of POS tagging. Sequential classification approaches decompose the probability as follows,

$$P(t_1...t_n|o) = \prod_{i=1}^{n} p(t_i|t_1...t_{i-1}o). \quad (2)$$

This is the left-to-right decomposition. If we make a first-order markov assumption, the equation becomes

$$P(t_1...t_n|o) = \prod_{i=1}^{n} p(t_i|t_{i-1}o). \quad (3)$$

Then we can employ a probabilistic classifier trained with the preceding tag and observations in order to obtain $p(t_i|t_{i-1}o)$ for local classification. A common choice for the local probabilistic classifier is maximum entropy classifiers (Berger et al., 1996). The best tag sequence can be efficiently computed by using a Viterbi decoding algorithm in polynomial time.

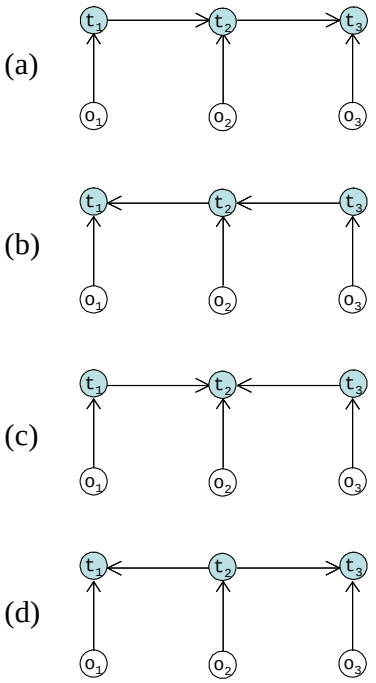

Figure 1: Different structures for decomposition

The right-to-left decomposition is

$$P(t_1...t_n|o) = \prod_{i=1}^{n} p(t_i|t_{i+1}o). \quad (4)$$

These two ways of decomposition are widely used in various tagging problems in natural language processing. The issue with such decompositions is that you have only the information about the preceding (or following) tags when performing local classification.

From the viewpoint of local classification, we want to give the classifier as much information as possible because the information about neighboring tags is useful in general.

As an example, consider the situation where we are going to annotate a three-word sentence with part-of-speech tags. Figure 1 shows four possible ways of decomposition. They correspond to the following equations:

$$(a)\ P(t_1...t_3|o) = P(t_1|o)P(t_2|t_1o)P(t_3|t_2o) \quad (5)$$

$$(b)\ P(t_1...t_3|o) = P(t_3|o)P(t_2|t_3o)P(t_1|t_2o) \quad (6)$$

$$(c)\ P(t_1...t_3|o) = P(t_1|o)P(t_3|o)P(t_2|t_3t_1o) \quad (7)$$

$$(d)\ P(t_1...t_3|o) = P(t_2|o)P(t_1|t_2o)P(t_3|t_2o) \quad (8)$$

(a) is a standard left-to-right decomposition, and (b) is a right-to-left decomposition. Notice that in decomposition (c) the local classifier can use the information about the tags on both sides when deciding t_2. If, for example, the central word is difficult to tag (e.g. an unknown word), we might as well take the decomposition structure (c) because the local classifier can have rich information when deciding the tag of the most difficult word. In general if we have an n-word sentence and adopt a first-order markov assumption, we have 2^{n-1} possible ways of decomposition because each of the $n-1$ edges in the corresponding graph has two directions (left-to-right or right-to-left).

Our bidirectional inference method is to consider all possible decomposition structures and choose the "best" structure and tag sequence. We will show in the next section that this is actually possible in polynomial time by dynamic programming.

As for training, let us look at the equations of four different decompositions above. You can notice that there are only four types of local conditional probabilities: $P(t_i|t_{i-1}o)$, $P(t_i|t_{i+1}o)$, $P(t_i|t_{i-1}t_{i+1}o)$, and $P(t_i|o)$.

This means that if we have these four types of local classifiers, we can consider any decomposition structures in the decoding stage. These local classifiers can be obtained by standard training with corresponding neighboring tag information. Training the first two types of classifiers is exactly the same as the training of popular left-to-right and right-to-left sequential classification models respectively.

If we take a second-order markov assumption, we need to train 16 types of local classifiers because each of the four neighboring tags of a classification target has two possibilities of availability. In general, if we take a k-th order markov assumption, we need to train 2^{2k} types of local classifies.

2.1 Polynomial Time Inference

This section describes an algorithm to find the decomposition structure and tag sequence that give the highest probability. The algorithm for the first-order case is an adaptation of the algorithm for decoding the best sequence on a bidirectional dependency network introduced by (Toutanova et al., 2003), which originates from the Viterbi decoding algorithm for second-order markov models.

```
function bestScore()
{
  return bestScoreSub(n+2, ⟨end, end, end⟩, ⟨L, L⟩);
}

function bestScoreSub(i+1, ⟨t_{i-1}, t_i, t_{i+1}⟩, ⟨d_{i-1}, d_i⟩)
{
  // memorization
  if (cached(i+1, ⟨t_{i-1}, t_i, t_{i+1}⟩, ⟨d_{i-1}, d_i⟩))
    return cache(i+1, ⟨t_{i-1}, t_i, t_{i+1}⟩, ⟨d_{i-1}, d_i⟩);
  // left boundary case
  if (i = -1)
    if (⟨t_{i-1}, t_i, t_{i+1}⟩ = ⟨start, start, start⟩)   return 1;
    else                                                   return 0;
  // recursive case
  P = localClassification(i, ⟨t_{i-1}, t_i, t_{i+1}⟩, ⟨d_{i-1}, d_i⟩);
  return max_{d_{i-2}} max_{t_{i-2}} P× bestScoreSub(i,
⟨t_{i-2}, t_{i-1}, t_i⟩, ⟨d_{i-2}, d_{i-1}⟩);
}

function localClassification(i, ⟨t_{i-1}, t_i, t_{i+1}⟩, ⟨d_{i-1}, d_i⟩)
{
  if (d_{i-1} = L & d_i = L)   return P(t_i|t_{i+1}, o);
  if (d_{i-1} = L & d_i = R)   return P(t_i|o);
  if (d_{i-1} = R & d_i = L)   return P(t_i|t_{i-1}t_{i+1}, o);
  if (d_{i-1} = R & d_i = R)   return P(t_i|t_{i-1}, o);
}
```

Figure 2: Pseudo-code for bidirectional inference for the first-order conditional markov models. d_i is the direction of the edge between t_i and t_{i+1}.

Figure 2 shows a polynomial time decoding algorithm for our bidirectional inference. It enumerates all possible decomposition structures and tag sequences, and finds the highest probability sequence. Polynomial time is achieved by caching. Note that for each local classification, the algorithm needs to choose the appropriate local classifier by taking into account the directions of the adjacent edges of the classification target.

The second-order case is similar but slightly more complex. Figure 3 shows the algorithm. The recursive function needs to consider the directions of the four adjacent edges of the classification target, and maintain the directions of the two neighboring edges to enumerate all possible edge directions. In addition, the algorithm must rule out cycles in the structure.

2.2 Decoding with the Easiest-First Strategy

We presented a polynomial time decoding algorithm in the previous section. However, polynomial time is not low enough in practice. Indeed, even the Viterbi decoding of second-order markov models for POS

```
function bestScore()
{
  return bestScoreSub(n+3, ⟨end, end, end, end, end⟩, ⟨L, L, L, L⟩, ⟨L, L⟩);
}

function bestScoreSub(i+2, ⟨t_{i-2}, t_{i-1}, t_i, t_{i+1}t_{i+2}⟩, ⟨d'_{i-1}, d_{i-1}, d_i, d'_{i+1}⟩, ⟨d_{i-2}, d'_i⟩)
{
  // to avoid cycles
  if (d_{i-1} = d_i & d_i != d'_i)   return 0;
  // memorization
  if (cached(i+2, ⟨t_{i-2}, t_{i-1}, t_i, t_{i+1}t_{i+2}⟩, ⟨d'_{i-1}, d_{i-1}, d_i, d'_{i+1}⟩, ⟨d_{i-2}, d'_i⟩)
    return cache(i+1, ⟨t_{i-2}, t_{i-1}, t_i, t_{i+1}t_{i+2}⟩, ⟨d'_{i-1}, d_{i-1}, d_i, d'_{i+1}⟩, ⟨d_{i-2}, d'_i⟩));
  // left boundary case
  if (i = -2)
    if (⟨t_{i-2}, t_{i-1}, t_i, t_{i+1}, t_{i+2}⟩ = ⟨start, start, start, start, start⟩)   return 1;
    else                                                                                   return 0;
  // recursive case
  P = localClassification(i, ⟨t_{i-2}, t_{i-1}, t_i, t_{i+1}, t_{i+2}⟩, ⟨d'_{i-1}, d_{i-1}, d_i, d'_{i+1}⟩);
  return max_{d'_{i-2}} max_{d_{i-3}} max_{t_{i-3}} P ×
      bestScoreSub(i+1, ⟨t_{i-3}, t_{i-2}, t_{i-1}, t_i t_{i+1}⟩, ⟨d'_{i-2}, d_{i-2}, d_{i-1}, d'_i⟩, ⟨d_{i-3}, d'_{i-1}⟩);
}
```

Figure 3: Pseudo-code for bidirectional inference for the second-order conditional markov models. d_i is the direction of the edge between t_i and t_{i+1}. d'_i is the direction of the edge between t_{i-1} and t_{i+1}. We omit the localClassification function because it is the obvious extension of that for the first-order case.

tagging is not practical unless some pruning method is involved. The computational cost of the bidirectional decoding algorithm presented in the previous section is, of course, larger than that because it enumerates all possible directions of the edges on top of the enumeration of possible tag sequences.

In this section we present a greedy version of the decoding method for bidirectional inference, which is extremely simple and significantly more efficient than full bidirectional decoding.

Instead of enumerating all possible decomposition structures, the algorithm determines the structure by adopting the easiest-first strategy. The whole decoding algorithm is given below.

1. Find the "easiest" word to tag.
2. Tag the word.
3. Go back to 1. until all the words are tagged.

The "easiest" word to tag is the word for which the classifier outputs the highest probability. In finding the easiest word, we use the appropriate local classifier according to the availability of the neighboring tags. Therefore, in the first iteration, we always use the local classifiers trained with no contextual tag information (i.e. $(P(t_i|o))$. Then, for example, if t_3 has been tagged in the first iteration in a three-word sentence, we use $P(t_2|t_3 o)$ to compute the probability for tagging t_2 in the second iteration (as in Figure 1 (b)).

A naive implementation of this algorithm requires $O(n^2)$ invocations of local classifiers, where n is the number of the words in the sentence, because we need to update the probabilities over the words at each iteration. However, a k-th order Markov assumption obviously allows us to skip most of the probability updates, resulting in $O(kn)$ invocations of local classifiers. This enables us to build a very efficient tagger.

3 Maximum Entropy Classifier

For local classifiers, we used a maximum entropy model which is a common choice for incorporating various types of features for classification problems in natural language processing (Berger et al., 1996).

Regularization is important in maximum entropy modeling to avoid overfitting to the training data. For this purpose, we use the maximum entropy modeling with inequality constraints (Kazama and Tsujii, 2003). The model gives equally good performance as the maximum entropy modeling with Gaussian priors (Chen and Rosenfeld, 1999), and

the size of the resulting model is much smaller than that of Gaussian priors because most of the parameters become zero. This characteristic enables us to easily handle the model data and carry out quick decoding, which is convenient when we repetitively perform experiments. This modeling has one parameter to tune as in Gaussian prior modeling. The parameter is called *width factor*. We tuned this parameter using development data in each type of experiments.

4 Experiments

To evaluate the bidirectional inference methods presented in the previous sections, we ran experiments on POS tagging and text chunking with standard English data sets.

Although achieving the best accuracy is not the primary purpose of this paper, we explored useful feature sets and parameter setting by using development data in order to make the experiments realistic.

4.1 Part-of-speech tagging experiments

We split the Penn Treebank corpus (Marcus et al., 1994) into training, development and test sets as in (Collins, 2002). Sections 0-18 are used as the training set. Sections 19-21 are the development set, and sections 22-24 are used as the test set. All the experiments were carried out on the development set, except for the final accuracy report using the best setting.

For features, we basically adopted the feature set provided by (Toutanova et al., 2003) except for complex features such as crude company-name detection features because they are specific to the Penn Treebank and we could not find the exact implementation details. Table 2 lists the feature templates used in our experiments.

We tested the proposed bidirectional methods, conventional unidirectional methods and the bidirectional dependency network proposed by Toutanova (Toutanova et al., 2003) for comparison. [1]. All the models are second-order. Table 2 shows the accuracy and tagging speed on the development data [2]. Bidirectional inference methods clearly outperformed unidirectional methods. Note that the easiest-first decoding method achieves equally good performance with full bidirectional inference. Table 2 also shows that the easiest-last strategy, where we select and tag the most difficult word at each iteration, is clearly a bad strategy.

Current word	w_i	& t_i
Previous word	w_{i-1}	& t_i
Next word	w_{i+1}	& t_i
Bigram features	w_{i-1}, w_i	& t_i
	w_i, w_{i+1}	& t_i
Previous tag	t_{i-1}	& t_i
Tag two back	t_{i-2}	& t_i
Next tag	t_{i+1}	& t_i
Tag two ahead	t_{i+2}	& t_i
Tag Bigrams	t_{i-2}, t_{i-1}	& t_i
	t_{i-1}, t_{i+1}	& t_i
	t_{i+1}, t_{i+2}	& t_i
Tag Trigrams	$t_{i-2}, t_{i-1}, t_{i+1}$	& t_i
	$t_{i-1}, t_{i+1}, t_{i+2}$	& t_i
Tag 4-grams	$t_{i-2}, t_{i-1}, t_{i+1}, t_{i+2}$	& t_i
Tag/Word combination	t_{i-1}, w_i	& t_i
	t_{i+1}, w_i	& t_i
	t_{i-1}, t_{i+1}, w_i	& t_i
Prefix features	prefixes of w_i (up to length 10)	& t_i
Suffix features	suffixes of w_i (up to length 10)	& t_i
Lexical features	whether w_i has a hyphen	& t_i
	whether w_i has a number	& t_i
	whether w_i has a capital letter	& t_i
	whether w_i is all capital	& t_i

Table 1: Feature templates used in POS tagging experiments. Tags are parts-of-speech. Tag features are not necessarily used in all the models. For example, "next tag" features cannot be used in left-to-right models.

An example of easiest-first decoding is given below:

> The/DT/4 company/NN/7 had/VBD/11 sought/VBN/14 increases/NNS/13 totaling/VBG/12 $/$/2 80.3/CD/5 million/CD/8 ,/,/1 or/CC/6 22/CD/9 %/NN/10 ././3

Each token represents Word/PoS/DecodingOrder. Typically, punctuations and articles are tagged first. Verbs are usually tagged in later stages because their tags are likely to be ambiguous.

[1] For dependency network and full bidirectional decoding, we conducted pruning because the computational cost was too large to perform exhaustive search. We pruned a tag candidate if the zero-th order probability of the candidate $P(t_i|o)$ was lower than one hundredth of the zero-th order probability of the most likely tag at the token.

[2] Tagging speed was measured on a server with an AMD Opteron 2.4GHz CPU.

Method	Accuracy (%)	Speed (tokens/sec)
Left-to-right (Viterbi)	96.92	844
Right-to-left (Viterbi)	96.89	902
Dependency Networks	97.06	1,446
Easiest-last	96.58	2,360
Easiest-first	**97.13**	2,461
Full bidirectional	97.12	34

Table 2: POS tagging accuracy and speed on the development set.

Method	Accuracy (%)
Dependency Networks (2003)	97.24
Perceptron (2002)	97.11
SVM (2003)	97.05
Hidden Markov Models (2000)	96.48
Easiest-first	97.10
Full Bidirectional	97.15

Table 3: POS tagging accuracy on the test set.

We applied our bidirectional inference methods to the test data. The results are shown in Table 3. The table also summarizes the accuracies achieved by several other research efforts. The best accuracy is 97.24% achieved by bidirectional dependency networks (Toutanova et al., 2003) with a richer set of features that are carefully designed for the corpus. A perceptron algorithm gives 97.11% (Collins, 2002). Gimenez and Marquez achieves 97.05% with support vector machines (SVMs). This result indicates that bidirectional inference with maximum entropy modeling can achieve comparable performance to other state-of-the-art POS tagging methods.

4.2 Chunking Experiments

The task of chunking is to find all types of non-recursive phrases in a sentence. For example, a text chunker segments the sentence "He reckons the current account deficit will narrow to only 1.8 billion in September" into the following,

[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only 1.8 billion] [PP in] [NP September] .

We can regard chunking as a tagging task by converting chunks into tags on tokens. There are several ways of representing text chunks (Sang and Veenstra, 1999). We tested the Start/End representation in addition to the popular IOB2 representation because local classifiers can have fine-grained information about the neighboring tags in the Start/End representation.

Current word	w_i	& t_i
Previous word	w_{i-1}	& t_i
Word two back	w_{i-2}	& t_i
Next word	w_{i+1}	& t_i
Word two ahead	w_{i+2}	& t_i
Bigram features	w_{i-2}, w_{i-1}	& t_i
	w_{i-1}, w_i	& t_i
	w_i, w_{i+1}	& t_i
	w_{i+1}, w_{i+2}	& t_i
Current POS	p_i	& t_i
Previous POS	p_{i-1}	& t_i
POS two back	p_{i-2}	& t_i
Next POS	p_{i+1}	& t_i
POS two ahead	p_{i+2}	& t_i
Bigram POS features	p_{i-2}, p_{i-1}	& t_i
	p_{i-1}, p_i	& t_i
	p_i, p_{i+1}	& t_i
	p_{i+1}, p_{i+2}	& t_i
Trigram POS features	p_{i-2}, p_{i-1}, p_i	& t_i
	p_{i-1}, p_i, p_{i+1}	& t_i
	p_i, p_{i+1}, p_{i+2}	& t_i
Previous tag	t_{i-1}	& t_i
Tag two back	t_{i-2}	& t_i
Next tag	t_{i+1}	& t_i
Tag two ahead	t_{i+2}	& t_i
Bigram tag features	t_{i-2}, t_{i-1}	& t_i
	t_{i-1}, t_{i+1}	& t_i
	t_{i+1}, t_{i+2}	& t_i

Table 4: Feature templates used in chunking experiments.

For training and testing, we used the data set provided for the CoNLL-2000 shared task. The training set consists of section 15-18 of the WSJ corpus, and the test set is section 20. In addition, we made the development set from section 21 of the corpus [3].

We basically adopted the feature set provided in (Collins, 2002) and used POS-trigrams in addition. Table 4 lists the features used in chunking experiments.

Table 5 shows the results on the development set. Again, bidirectional methods exhibit better performance than unidirectional methods. The difference is bigger with the Start/End representation. Dependency networks did not work well for this chunking task, especially with the Start/End representation.

We applied the best model on the development set in each chunk representation type to the test data. Table 6 summarizes the performance on the

[3] We used a Perl script provided in in http://ilk.kub.nl/ sabine/chunklink/

Representation	Method	Order	Recall	Precision	F-score	Speed (tokens/sec)
IOB2	Left-to-right	1	93.17	93.05	93.11	1,775
		2	93.13	92.90	93.01	989
	Right-to-left	1	92.92	92.82	92.87	1,635
		2	92.92	92.74	92.87	927
	Dependency Networks	1	92.71	92.91	92.81	2,534
		2	92.61	92.95	92.78	1,893
	Easiest-first	1	93.17	93.04	93.11	2,441
		2	93.35	93.32	**93.33**	1,248
	Full Bidirectional	1	93.29	93.14	93.21	712
		2	93.26	93.12	93.19	48
Start/End	Left-to-right	1	92.98	92.69	92.83	861
		2	92.96	92.67	92.81	439
	Right-to-left	1	92.92	92.83	92.87	887
		2	92.89	92.74	92.82	451
	Dependency Networks	1	87.10	89.56	88.32	1,894
		2	87.16	89.44	88.28	331
	Easiest-first	1	93.33	92.95	93.14	1,950
		2	93.31	92.95	93.13	1,016
	Full Bidirectional	1	93.52	93.26	**93.39**	392
		2	93.44	93.20	93.32	4

Table 5: Chunking F-scores on the development set.

test set. Our bidirectional methods achieved F-scores of 93.63 and 93.70, which are better than the best F-score (93.48) of the CoNLL-2000 shared task (Sang and Buchholz, 2000) and comparable to those achieved by other state-of-the-art methods.

5 Discussion

There are some reports that one can improve the performance of unidirectional models by combining the outputs of taggers with different decoding directions with some kind of voting. Shen reported an 0.39% accuracy improvement of supertagging with pairwise voting (Shen and Joshi, 2003). The biggest difference between our approach and such voting methods is that the local classifier in our bidirectional inference methods can have rich information for decision. Also, voting methods generally need many tagging processes to be run on a sentence, which makes it difficult to build a fast tagger.

As for the computational cost for training, our methods require us to train 2^{2n} types of classifiers when we adopt an nth order markov assumption. In many cases a second-order model is sufficient because further increase of n has little impact on performance. Thus the training typically takes four or 16 times as much time as it would take for training a single classifier, which looks somewhat expensive. However, because each type of classifier can be trained independently, the training can be performed completely in parallel and run with the same amount of memory as that for training a single classifier. This advantage contrasts to the case for CRFs which requires substantial amount of memory and computational cost if one tries to incorporate higher-order features about tag sequences.

Tagging speed is another important factor in building practical taggers for large-scale information extraction from a huge amount of text such as WWW documents. Our inference algorithm with the easiest-first strategy needs no Viterbi decoding unlike MEMMs and CRFs, and makes it possible to perform very fast tagging with high precision.

6 Conclusion

We have presented a bidirectional inference algorithm for sequence labeling problems such as POS tagging, named entity recognition and text chunking. The algorithm can enumerate all possible decomposition structures and find the highest probability sequence together with the corresponding decomposition structure in polynomial time. We have also presented an efficient bidirectional inference algorithm based on the easiest-first strategy, which gives comparable performance to full bidirectional inference with significantly lower computational cost.

Method	Recall	Precision	F-score
SVM (Kudoh and Matsumoto, 2000)	93.51	93.45	93.48
SVM voting (Kudo and Matsumoto, 2001)	93.92	93.89	93.91
Regularized Winnow (with basic features) (Zhang et al., 2002)	93.60	93.54	93.57
Perceptron (Carreras and Marquez, 2003)	93.29	94.19	93.74
Easiest-first (IOB2, second-order)	93.59	93.68	93.63
Full Bidirectional (Start/End, first-order)	93.70	93.65	93.70

Table 6: Chunking F-scores on the test set.

Experimental results of POS tagging and text chunking show that the proposed bidirectional inference methods consistently outperform unidirectional inference methods and our bidirectional MEMMs give comparable performance to that achieved by state-of-the-art learning algorithms including kernel support vector machines.

A natural extension of this work is to replace the maximum entropy modeling, which was used as the local classifiers, with other machine learning algorithms. Support vector machines with appropriate kernels is a good candidate because they have good generalization performance as a single classifier. Although SVMs do not output probabilities, the easiest-first method would be easily applicable by considering the margins output by SVMs as the confidence of local classification.

References

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Thorsten Brants. 2000. TnT – a statistical part-of-speech tagger. In *Proceedings of the 6th Applied NLP Conference (ANLP)*.

Xavier Carreras and Lluis Marquez. 2003. Phrase recognition by filtering and ranking with perceptrons. In *Proceedings of RANLP-2003*.

Stanley F. Chen and Ronald Rosenfeld. 1999. A gaussian prior for smoothing maximum entropy models. *Technical Report CMUCS -99-108, Carnegie Mellon University*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP 2002*, pages 1–8.

Jesus Gimenez and Lluis Marquez. 2003. Fast and accurate part-of-speech tagging: The SVM approach revisited. In *Proceedings of RANLP 2003*, pages 158–165.

Jun'ichi Kazama and Jun'ichi Tsujii. 2003. Evaluation and extension of maximum entropy models with inequality constraints. In *Proceedings of EMNLP 2003*.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL 2001*.

Taku Kudoh and Yuji Matsumoto. 2000. Use of support vector learning for chunk identification. In *Proceedings of CoNLL-2000*, pages 142–144.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML 2001*, pages 282–289.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132.

Erik F. Tjong Kim Sang and Jorn Veenstra. 1999. Representing text chunks. In *Proceedings of EACL 1999*, pages 173–179.

Libin Shen and Aravind K. Joshi. 2003. A SNoW based Supertagger with Application to NP Chunking. In *Proceedings of ACL 2003*, pages 505–512.

Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*, pages 252–259.

Tong Zhang, Fred Damereau, and David Johnson. 2002. Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, 2:615–638.