unpublished manuscript. Please do not distribute

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

Abstract

This paper presents a bidirectional inference algorithm for sequence labeling problems such as POS tagging 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 also present an efficient decoding algorithm based on the easiest-first heuristics, which gives comparably good performance to full bidirectional inference with extremely less computational cost. Experimental results of POS tagging, base NP chunking and text chunking show that the proposed bidirectional inference methods consistently outperform unidirectional inference methods and our bidirectional MEMMs give comparative performance 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, phrase chunking and named entity tagging 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 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. In general training higher-order CRFs requires huge computational resource.

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

One obvious way to improve the performance of a sequential classification method is to enrich the information that the local classifiers can use. In particular, the information about neighboring tags are helpful in most cases. However in general the local classifier cannot have the information about future tags (e.g. the right-side tags in left-to-right decoding). To make use of the information about future tags, Toutanova proposed a decoding 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,

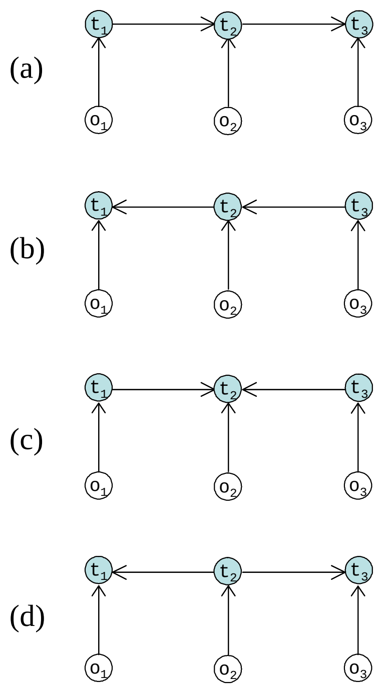

Figure 1: Different structures for decomposition

the method potentially suffers from "collusion" effects which make the model lock onto conditionally consistent but jointly unlikely sequences.

In this paper we present a natural way for making use of future tags. Our inference method considers all possible ways of decomposition and choose the "best" decomposition structure, so the information about future tags is used in appropriate situations.

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 $o = o_1...o_n$

$$P(t_1...t_n|o). \tag{1}$$

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

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

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

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

Then we can use 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.

A right-to-left decomposition is

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

These two ways of decomposition is widely used in various tagging problems in natural language processing. The point is that you have only the information about the preceding (or following) tags when performing local classification ~~whichever way of decomposition you take~~.

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

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:

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

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

$$(c)\ P(t_1...t_3|o) = P(t_1|o)P(t_3|o)P(t_2|t_3t_1o) \tag{7}$$

$$(d)\ P(t_1...t_3|o) = P(t_2|o)P(t_1|t_2o)P(t_3|t_2o) \tag{8}$$

(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 in 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 possible ways of decomposition because each of the n 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 learning, 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 structure in the decoding time. 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 left-to-right and right-to-left classifiers 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 first-order cases is an adaptation of the algorithm for decoding the best sequence on a bidirectional dependency network introduced by (Toutanova et al., 2003), which is originated from the Viterbi decoding algorithm for second-order markov models.

Figure 2 shows a polynomial time decoding algorithm for our bidirectional inference. It enumerates all possible decomposition structures and tag sequences, and find 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.

```
function bestScore()
{
  return bestScoreSub(n+2, $\langle end, end, end \rangle$, $\langle L, L \rangle$);
}

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

function localClassification(i, $\langle t_{i-1}, t_i, t_{i+1} \rangle$, $\langle d_{i-1}, d_i \rangle$)
{
  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: Pseudocode 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}.

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 small enough in practice. Indeed, even the Viterbi decoding of second-order markov models for POS tagging is not practical unless some pruning method is involved. The complexity 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 an alternative decod-

```
**function** bestScore()
{
   return bestScoreSub(n+3, $\langle end, end, end, end, end\rangle$, $\langle L, L, L, L\rangle$, $\langle L, L\rangle$);
}

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

Figure 3: Pseudocode 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.

ing method for bidirectional inference, which is extremely simple and efficient than full bidirectional decoding.

The deterministic version of 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" means the word with 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. For example, if t_3 was decided in the first iteration, we use $P(t_2|t_3 o)$ to compute the probability for tagging t_2 in the next iteration (as in Figure 1 (b)).

In experiments we extended this algorithm to keep multiple hypotheses, which is essentially beam search. We should mention that this easiest-first strategy goes well with beam search because few alternative hypotheses are generated in earlier stages of decoding.

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 that purpose, we used 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.

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.

4 Experiments

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

Although achieving the best accuracy is not the objective of this paper, we explored useful feature sets and parameter setting 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 like crude company name detection features because they are specific to the Penn Treebank. Table 2 lists the feature templates used in our experiments.

Method	Accuracy (%)
Left-to-right	96.92
Right-to-left	96.88
Dependency Networks	97.06
Easiest-first	**97.14**
Easiest-first (deterministic)	97.13
Full bidirectional	97.12

Table 2: POS tagging accuracy on the development set.

Method	Accuracy (%)
Dependency Net (2003)	97.24
Perceptron (2002)	97.11
SVM (2003)	97.05
TnT (2000)	96.48
Easiest-first	97.10

Table 3: POS tagging accuracy on the test set.

We tested unidirectional methods, which are identical to the popular left-to-right and right-to-left MEMMs, and the bidirectional dependency network (Toutanova et al., 2003) for comparison. Table 2 shows the accuracies of various decoding methods on development data. All the models are second-order. Bidirectional inference methods all outperformed unidirectional methods. Note that easiest-first decoding methods achieve equally good performance with full bidirectional inference. 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 are tagged first, and articles tagged next. Proper nouns are usually tagged in later stages because they are likely to be unknown words.

Since the easiest-first decoding method performed best in the development set, we applied it to the test data. The result is 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 rich features. A perceptron algorithm gives 97.11% (Collins, 2002). Gimenez and Marquez achieves 97.05% with support vector machines. 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 Base NP Chunking Experiments

The task of NP chunking is to find non-overlapping, non-recursive noun phrases in a sentence. 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.

For training and testing, we used the data provided by (Ramshaw and Marcus, 1995). The data consists of sections 15-18 of the Wall Street Journal corpus as training material and section 20 of the corpus as test material. In addition, we constructed a development set consisting of section 21 of the corpus as done in (Collins, 2002).

To explore useful features we began with the feature set provided in (Collins, 2002), then found POS-trigrams useful in testing taggers on the development set. Table 4 lists the features used in chunking experiments.

Table 5 shows the F-scores of various decoding methods on development data. As in POS tagging experiments, bidirectional methods consistently outperform unidirectional methods. Bidirectional dependency networks did not work well for this task.

Finally, we applied the second-order full bidirectional decoding method to the test data and obtained an F-score of 93.90. For base NP chunking, the best f-score achieved by a single classifier is 94.38 (Sha and Pereira, 2003). They used a second-order conditional random field and carried out rigorous parameter turning about Gaussian priors and the number of iterations for numerical optimization. Kudo (2001) achieved an f-score of 94.22 by combining many kernel support vector machines.

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.

Representation	Method	Order	F-score
IOB2	Left-to-right	1	93.31
		2	93.26
	Right-to-left	1	93.29
		2	93.29
	Dep. Net	1	90.14
		2	90.32
	Easiest-first	1	93.42
		2	93.52
	Easiest-first (deterministic)	1	93.43
		2	93.54
	Full Bidir.	1	93.48
		2	**93.60**
Start/End	Left-to-right	1	93.43
		2	93.49
	Right-to-left	1	93.29
		2	93.24
	Dep. Net	1	91.28
		2	91.08
	Easiest-first	1	93.89
		2	93.77
	Easiest-first (deterministic)	1	93.90
		2	93.77
	Full Bidir.	1	93.97
		2	**94.04**

Table 5: base NP chunking F-scores on the development set.

Representation	Method	Order	F-score
IOB2	Left-to-right	1	93.11
		2	93.01
	Right-to-left	1	92.87
		2	92.87
	Dep. Net	1	92.81
		2	92.78
	Easiest-first	1	93.11
		2	93.32
	Easiest-first	1	93.11
	(deterministic)	2	**93.33**
	Full Bidir.	1	93.21
		2	93.18
Start/End	Left-to-right	1	92.84
		2	92.81
	Right-to-left	1	92.87
		2	92.82
	Dep. Net	1	88.32
		2	88.28
	Easiest-first	1	93.15
		2	93.13
	Easiest-first	1	93.15
	(deterministic)	2	93.14
	Full Bidir.	1	**93.39**
		2	93.30

Table 6: chunking F-scores on the development set.

4.3 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] .

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 of the corpus. In addition, we made the development set from section 21 using a Perl script [1].

Table 6 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.

Because the first-oder full bidirectional inference method with start/end representation performed best, we applied it to the test data. We obtained an F-score of 93.70, which is better than the best F-score (93.48) of the CoNLL-2000 shared task (Sang and Buchholz, 2000).

[1]provided in http://ilk.kub.nl/ sabine/chunklink/

It is worth mentioning that the F-score for NP recognition was 94.21, which was higher than that achieved by conducting only NP chunking. This suggests that providing rich information about neighboring tags has a positive effect for achieving good performance even if the task becomes more complex.

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 (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.

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 ~~a~~ 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.

As for the tagging speed, our deterministic easiest-first strategy can perform extremely fast decoding with almost no loss of accuracy. This is very important for building practical taggers.

6 Conclusion

We have presented a bidirectional inference algorithm for sequence labeling problems such as POS tagging 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 heuristics, which gives comparably good performance to full bidirectional inference with extremely less computational cost.

Experimental results of POS tagging, base NP chunking and text chunking show that the proposed bidirectional inference methods consistently outperform unidirectional inference methods and our bidirectional MEMMs give comparative performance 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 classifier, which was used as local classifiers, with other machine learning algorithms. Support vector machines with appropriate kernels is a good candidate because they have surprisingly high generalization capacity as a single classifier. In our modeling, however, the local classifiers should output probabilities in order to perform full bidirectional inference. Since SVMs do not provide probabilities, we might need some device to convert their outputs to probabilities as done in (Zhou, 2004).

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)*.

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*.

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.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In David Yarovsky and Kenneth Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94, Somerset, New Jersey. Association for Computational Linguistics.

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.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL 2003*.

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.

Zhou. 2004. Recognizing names in biomedical texts using hidden markov and svm plus sigmoid. In *Proceedings of the COLING 2004 Workshop on Natural Language Processing in Biomedicine and its Applications*.