Exact Decoding for Jointly Labeling and Chunking Sequences

Nobuyuki Shimizu
Department of Computer Science
State University of New York at Albany
Albany, NY 12222, USA
nobuyuki@shimizu.name

Andrew Haas
Department of Computer Science
State University of New York at Albany
Albany, NY 12222 USA
haas@cs.albany.edu

Abstract

There are two decoding algorithms essential to the area of natural language processing. One is the Viterbi algorithm for linear-chain models, such as HMMs or CRFs. The other is the CKY algorithm for probabilistic context free grammars. However, tasks such as noun phrase chunking and relation extraction seem to fall between the two, neither of them being the best fit. Ideally we would like to model entities and relations, with two layers of labels. We present a tractable algorithm for exact inference over two layers of labels and chunks with time complexity $O(n^2)$, and provide empirical results comparing our model with linear-chain models.

1 Introduction

The Viterbi algorithm and the CKY algorithms are two decoding algorithms essential to the area of natural language processing. The former models a linear chain of labels such as part of speech tags, and the latter models a parse tree. Both are used to extract the best prediction from the model (Manning and Schutze, 1999).

However, some tasks seem to fall between the two, having more than one layer but flatter than the trees created by parsers. For example, in relation extraction, we have entities in one layer and relations between entities as another layer. Another task is shallow parsing. We may want to model part-of-speech tags and noun/verb chunks at the same time, since performing simultaneous labeling may result in increased joint accuracy by sharing information between the two layers of labels.

To apply the Viterbi decoder to such tasks, we need two models, one for each layer. We must feed the output of one layer to the next layer. In such an approach, errors in earlier processing nearly always accumulate and produce erroneous results at the end. If we use CKY, we usually end up flattening the output tree to obtain the desired output. This seems like a round-about way of modeling two layers.

There are previous attempts at modeling two layer labeling. Dynamic Conditional Random Fields (DCRFs) by (McCallum et al, 2003; Sutton et al, 2004) is one such attempt, however, exact inference is in general intractable for these models and the authors were forced to settle for approximate inference.

Our contribution is a novel model for two layer labeling, for which exact decoding is tractable. Our experiments show that our use of label-chunk structures results in significantly better performance over cascaded CRFs, and that the model is a promising alternative to DCRFs.

The paper is organaized a follows: In Section 2 and 3, we describe the model and present the decoding algorithm. Section 4 describes the learning methods applicable to our model and the baseline models. In Section 5 and 6, we describe the experiments and the results.

Token	POS	NP
U.K.	JADJ	B
base	NOUN	I
rates	NOUN	I
are	VERB	O
at	OTHER	O
their	OTHER	B
highest	JADJ	I
level	NOUN	I
in	OTHER	O
eight	OTHER	B
years	NOUN	I
.	OTHER	O

Table 1: Example with POS and NP tags

2 Model for Joint Labeling and Chunking

Consider the task of finding *noun chunks*. The noun chunk extends from the beginning of a noun phrase to the head noun, excluding postmodifiers (which are difficult to attach correctly). Table 1 shows a sentence labeled with POS tags and segmented into noun chunks. B marks the first word of a noun chunk, I the other words in a noun chunk, and O the words that are not in a noun chunk. Note that we collapsed the 45 different POS labels into 5 labels, following (McCallum et al, 2003). All different types of adjectives are labeled as JADJ.

Each word carries two tags. Given the first layer, our aim is to present a model that can predict the second and third layers of tags at the same time. Assume we have n training samples, $\{(x^i, y^i)\}_{i=1}^n$, where x^i is a sequence of input tokens and y^i is a label-chunk structure for x^i. In this example, the first column contains the tokens x^i and the second and third columns together represent the label-chunk structures y^i. We will present an efficient exact decoding for this structure.

The label-chunk structure, shown in Table 2, is a representation of the two layers of tags. The tuples in Table 2 are called *parts*. If the token at index r carries a POS tag P and a chunk tag C, the first layer includes part $\langle C, P, r\rangle$. This part is called a *node*. If the tokens at index $r-1$ and r are in the same chunk, and C is the label of that chunk, the first layer also includes part $\langle C, P0, P, r-1, r\rangle$ (where $P0$ and P are the POS tags of the tokens at $r-1$ and r

Token	First Layer (POS)	Second Layer (NP)
U.K.	$\langle I, \mathit{JADJ}, 0\rangle$	
	$\langle I, \mathit{JADJ}, \mathit{NOUN}, 0, 1\rangle$	
base	$\langle I, \mathit{NOUN}, 1\rangle$	
	$\langle I, \mathit{NOUN}, \mathit{NOUN}, 1, 2\rangle$	
rates	$\langle I, \mathit{NOUN}, 2\rangle$	$\langle I, 0, 2\rangle$
		$\langle I, O, 2, 3\rangle$
are	$\langle O, \mathit{VERB}, 3\rangle$	
	$\langle O, \mathit{VERB}, \mathit{OTHER}, 3, 4\rangle$	
at	$\langle O, \mathit{OTHER}, 4\rangle$	$\langle O, 3, 4\rangle$
		$\langle O, I, 4, 5\rangle$
their	$\langle I, \mathit{OTHER}, 5\rangle$	
	$\langle I, \mathit{OTHER}, \mathit{JADJ}, 5, 6\rangle$	
highest	$\langle I, \mathit{JADJ}, 6\rangle$	
	$\langle I, \mathit{JADJ}, \mathit{NOUN}, 6, 7\rangle$	
level	$\langle I, \mathit{NOUN}, 7\rangle$	$\langle I, 5, 7\rangle$
		$\langle I, O, 7, 8\rangle$
in	$\langle O, \mathit{OTHER}, 8\rangle$	$\langle O, 8, 8\rangle$
		$\langle O, I, 8, 9\rangle$
eight	$\langle I, \mathit{OTHER}, 9\rangle$	
	$\langle I, \mathit{OTHER}, \mathit{NOUN}, 9, 10\rangle$	
years	$\langle I, \mathit{NOUN}, 10\rangle$	$\langle I, 9, 10\rangle$
		$\langle I, O, 10, 11\rangle$
.	$\langle O, \mathit{OTHER}, 11\rangle$	$\langle O, 11, 11\rangle$

Table 2: Example Parts

respectively). This part is called a *transition*. If a chunk tagged C extends from the token at q to the token at r inclusive, the second layer includes part $\langle C, q, r\rangle$. This part is a *chunk node*. And if the token at $q-1$ is the last token in a chunk tagged $C0$, while the token at q is the first token of a chunk tagged C, the second layer includes part $\langle C0, C, q-1, q\rangle$. This part is a *chunk transition*.

In this paper we use the common method of factoring the score of the label-chunk structure as the sum of the scores of all the parts. Each part in a label-chunk structure can be lexicalized, and gives rise to several features. For each feature, we have a corresponding weight. If we sum up the weights for these features, we have the score for the part, and if we sum up the scores of the parts, we have the score for the label-chunk structure.

Suppose we would like to score a pair (x^i, y^i) in the training set, and it happens to be the one shown in Table 2. To begin, let's say we would like to find the features for the part $\langle I, \mathit{NOUN}, 7\rangle$ of POS node type (1st Layer). This is the NOUN tag on the seventh token "level" in Table 2. By default, the POS node type generates the following binary feature.

- Is there a token labeled with "NOUN" in a chunk labeled with "I"?

Now, to have more features, we can lexicalize POS node type. Suppose we use x_r to lexicalize POS node $\langle C, P, r\rangle$, then we have the following binary feature, as it is $\langle I, NOUN, 7\rangle$ and x^i_7 = "level".

- Is there a token "level" labeled with "NOUN" in a chunk labeled with "I"?

We can also use x_{r-1} and x_r to lexicalize the parts of POS node type.

- Is there a token "level" labeled with "NOUN" in a chunk labeled with "I" that's preceded by "highest"?

This way, we have a complete specification of the feature set given the part type, lexicalization for each part type and the training set. Let us define $\mathbf{f}$ a boolean feature vector function such that each dimension of $\mathbf{f}(x^i, y^i)$ contains 1 if the pair (x^i, y^i) has the feature, 0 otherwise. Now define a real-valued weight vector $\mathbf{w}$ with the same dimension as $\mathbf{f}$. To represent the score of the pair (x^i, y^i), we write $s(x^i, y^i) = \mathbf{w}^\top\mathbf{f}(x^i, y^i)$ We could also have $\mathbf{w}^\top\mathbf{f}(x^i, \{p\})$ where p just a single part, in which case we just write $s(p)$.

Assuming an appropriate feature representation as well as a weight vector $\mathbf{w}$, we would like to find the highest scoring label-chunk structure $y = argmax_{y'}(\mathbf{w}^\top\mathbf{f}(x, y'))$ given an input sentence x.

In the upcoming section, we present a decoding algorithm for the label-chunk structures, and later we give a method for learning the weight vector used in the decoding.

3 Decoding

The decoding algorithm is shown in Figure 1. The idea is to use two tables for dynamic programming: label_table and chunk_table.

Suppose we are examining the current position r, and would like to consider extending the chunk $[q, r-1]$ to $[q, r]$. We need to know the chunk tag C for $[q, r-1]$ and the last POS tag $P0$ at index $r-1$. The array entry label_table$[q][r-1]$ keeps track of this information.

Then we examine how the current chunk is connected with the previous chunk. The array entry chunk_table$[q][C0]$ keeps track of the score of the best label-chunk structure from 0 up to the index q that has the ending chunk tag $C0$. Now checking the chunk transition from $C0$ to C at the index q is simple, and we can record the score of this chunk to chunk_table$[r][C]$, so that the next chunk starting at r can use this information.

In short, we are executing two Viterbi algorithms on the first and second layer at the same time. One extends $[q, r-1]$ to $[q, r]$, considering the node indexed by r (first layer). The other extends $[0, q]$ to $[0, r]$, considering the node indexed by $[q, r]$ (second layer). The dynamic programming table for the first layer is kept in the label_table ($r-1$ and $P0$ are used in the Viterbi algorithm for this layer) and that for the second layer in the chunk_table (q and $C0$ used). The algorithm returns the best score of the label-chunk structure.

To recover the structure, we simply need to maintain back pointers to the items that gave rise to the each item in the dynamic programming table. This is just like maintaining back pointers in the Viterbi algorithm for sequences, or the CKY algorithm for parsing.

The pseudo-code shows that the run-time complexity of the decoding algorithm is O(n^2) unlike that of CFG parsing, O(n^3). Thus the algorithm performs better on long sentences. On the other hand, the constant is c^2p^2 where c is the number of chunk tags and p is the number of POS tags.

4 Learning

4.1 Voted Perceptron

In the CKY and Viterbi decoders, we use the forward-backward or inside-outside algorithm to find the marginal probabilities. Since we don't yet have the inference algorithm to find the marginal probabilities of the parts of a label-chunk structure, we use an online learning algorithm to train the model. Despite this restriction, the voted perceptron is known for its performance (Sha and Pereira, 2003).

The voted perceptron we use is the adaptation of (Freund and Schapire, 1999) to the structured setting. Algorithm 4.1 shows the pseudo code for the training, and the function $update(\mathbf{w}_k, x^i, y^i, y')$ returns $\mathbf{w}_k - \mathbf{f}(x^i, y') + \mathbf{f}(x^i, y^i)$.

Given a training set $\{(x^i y^i)\}_{i=1}^n$ and the epoch number T, Algorithm 4.1 will return a list of

Algorithm 3.1: DECODE(the scoring function $s(p)$)

```
$score := 0$;
for $q := index\_start$ to $index\_end$
    for $length := 1$ to $index\_end - q$
        $r := q + length$;
        for each Chunk Tag $C$
            for each Chunk Tag $C0$
                for each POS Tag $P$
                    for each POS Tag $P0$
                        $score := 0$;
                        if $(length > 1)$
                            #Add the score of the transition from r-2 to r-1. (1st Layer, POS)
                            $score := score + s(\langle C, P0, P, r-2, r-1 \rangle) + label\_table[q][r-1][C][P0]$;
                        #Add the score of the node at r-1. (1st Layer, POS)
                        $score := score + s(\langle C, P, r-1 \rangle)$;
                        if $(score >= label\_table[q][r][C][P])$
                            $label\_table[q][r][C][P] := score$;
                        #Add the score of the chunk node at [q,r-1]. (2nd Layer, NP)
                        $score := score + s(\langle C, q, r-1 \rangle)$;
                        if $(index\_start < q)$
                            #Add the score of the chunk transition from q-1 to q. (2nd Layer, NP)
                            $score := score + s(\langle C0, C, q-1, q \rangle) + chunk\_table[q][C0]$;
                        if $(score >= chunk\_table[r][C])$
                            $chunk\_table[r][C] := score$;
                    end for
                end for
            end for
        end for
    end for
end for
$score := 0$;
for each $C$ in $chunk\_tags$
    if $(chunk\_table[index\_end][C] >= score)$
        $score := chunk\_table[index\_end][C]$;
        $last\_symbol := C$;
end for
return $(score)$
```

Note: Since the scoring function $s(p)$ is defined as $\mathbf{w}^\top \mathbf{f}(\mathbf{x}^i, \{p\})$, the input sequence $\mathbf{x}^i$ and the weight vector $\mathbf{w}$ are also the inputs to the algorithm.

Figure 1: Decoding Algorithm

weighted perceptrons $\{(\mathbf{w}_1, c_1), ..(\mathbf{w}_k, c_k)\}$. The final model V uses the weight vector

$$\mathbf{w} = \frac{\sum_{j=1}^{k}(c_j \mathbf{w}_j)}{Tn}$$

(Collins, 2002).

```
Algorithm 4.1: TRAIN($T, \{(x^i, y^i)\}_{i=1}^n$)

 $k := 0$;
 $\mathbf{w}_1 := \mathbf{0}$;
 $c_1 := 0$;
 for $t := 1$ to $T$
   for $i := 1$ to $n$
     $y' := argmax_y(\mathbf{w}_k^\top \mathbf{f}(y, x^i))$
     if $(y' = y^i)$
       $c_k := c_k + 1$;
     else
       $\mathbf{w}_{k+1} := update(\mathbf{w}_k, x^i, y^i, y')$;
       $c_{k+1} := 1$;
       $k := k + 1$;
       $c_k := c_k + 1$;
   end for
 end for
 return $(\{(\mathbf{w}_1, c_1), ..(\mathbf{w}_k, c_k)\})$
```

```
Algorithm 4.2: UPDATE1($\mathbf{w}_k, x^i, y^i, y'$)

 return $(\mathbf{w}_k - \mathbf{f}(x^i, y') + \mathbf{f}(x^i, y^i))$
```

```
Algorithm 4.3: UPDATE2($\mathbf{w}_k, x^i, y^i, y'$)

 $\delta = max(0, min(\frac{l_i(y') - s(x^i, y^i) + s(x^i, y')}{\|\mathbf{f}_i(y^i) - \mathbf{f}_i(y')\|^2}, 1))$;
 return $(\mathbf{w}_k - \delta\mathbf{f}(x^i, y') + \delta\mathbf{f}(x^i, y^i))$
```

4.2 Max Margin

4.2.1 Sequential Minimum Optimization

A max margin method minimizes the regularized empirical risk function with the hard (penalized) margin

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i (s(x^i, y^i) - \max_y (s(x^i, y) - l_i(y)))$$

l_i finds the loss for y with respect to y^i, and it is assumed that the function is decomposable just as y is decomposable to the parts. This equation is equivalent to

$$\min_{\mathbf{w}} \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$
$$\forall i, y,\ s(x^i, y^i) + \xi_i \geq s(x^i, y) - l_i(y)$$

After taking the Lagrange dual formation, we have

$$\max_{\alpha \geq 0} -\frac{1}{2}\|\sum_{i,y} \alpha_i(y)(\mathbf{f}(x^i, y^i) - \mathbf{f}(x^i, y))\|^2 + \sum_{i,y} \alpha_i(y) l_i(y)$$

$$\text{such that } \sum_y \alpha_i(y) = C$$

and

$$\mathbf{w} = \sum_{i,y} \alpha_i(y)(\mathbf{f}(x^i, y^i) - \mathbf{f}(x^i, y)) \qquad (1)$$

This quadratic program can be optimized by bi-coordinate descent, known as Sequential Minimum Optimization. Given an example i and two label-chunk structures y' and y'',

$$d = \frac{l_i(y') - l_i(y'') - (s(x^i, y'') - s(x^i, y'))}{\|\mathbf{f}_i(y'') - \mathbf{f}_i(y')\|^2} \qquad (2)$$

$$\delta = max(-\alpha_i(y'), min(d, \alpha_i(y'')))$$

The updated values are : $\alpha_i(y') := \alpha_i(y') + \delta$ and $\alpha_i(y'') := \alpha_i(y'') - \delta$.

Using the equation (1), any increase in α can be translated to $\mathbf{w}$. For a naive SMO, this update is executed for each training sample i, for all pairs of possible parses y' and y'' for x^i. See (Taskar and Klein, 2005; Zhang, 2001; Jaakkola et al, 2000).

Here is where we differ from (Taskar et al, 2004). We choose y'' to be the correct parse y^i, and y' to be the best runner-up. After setting the initial weights using y^i, we also set $\alpha_i(y^i) = 1$ and $\alpha_i(y') = 0$. Although these alphas are not correct, as optimization nears the end, the margin is wider; $\alpha_i(y^i)$ and $\alpha_i(y')$ gets closer to 1 and 0 respectively. Given this approximation, we can compute δ. Then, the function $update(\mathbf{w}_k, x^i, y^i, y')$ will return $\mathbf{w}_k - \delta\mathbf{f}(x^i, y') + \delta\mathbf{f}(x^i, y^i)$ and we have reduced the SMO to the perceptron weight update.

4.2.2 Margin Infused Relaxed Algorithm

We can think of maximizing the margin in terms of extending the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al, 2003) to learning with structured outputs. (McDonald et al, 2005) presents this approach for dependency parsing.

In particuler, Single-best MIRA (McDonald et al, 2005) uses only the single margin constraint for the runner up y' with the highest score. The resulting online update would be $\mathbf{w}_{k+1}$ with the following condition: $min\|\mathbf{w}_{k+1} - \mathbf{w}_k\|$ such that $s(x^i, y^i) - s(x^i, y') \geq l_i(y')$ where $y' = argmax_y s(x^i, y)$.

Incidentally, the equation (2) for d above when $\alpha_i(y^i) = 1$ and $\alpha_i(y') = 0$ solves this minimization problem as well, and the weight update is the same as the SMO case.

4.2.3 Conditional Random Fields

Instead of minimizing the regularized empirical risk function with the hard (penalized) margin, conditional random fields try to minimize the same with the negative log loss:

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i (s(x^i, y^i) - log(\sum_y s(x^i, y)))$$

Usually, CRFs use marginal probabilities of parts to do the optimization. Since we have not yet come up with the algorithm to compute marginals for a label-chunk structure, the training methods for CRFs is not applicable to our purpose. However, on sequence labeling tasks CRFs have shown very good performance (Lafferty et al, 2001; Sha and Pereira, 2003), and we will use them for the baseline comparison.

5 Experiments

5.1 Task: Base Noun Phrase Chunking

The data for the training and evaluation comes from the CoNLL 2000 shared task (Tjong Kim Sang and Buchholz, 2000), which is a portion of the Wall Street Journal.

We consider each sentence to be a training instance x^i, with single words as tokens.

The shared task data have a standard training set of 8936 sentences and a test set of 2012 sentences. For the training, we used the first 447 sentences from the standard training set, and our evaluation was done on the standard test set of the 2012 sentences. Let us define the set D to be the first 447 samples from the standard training set .

There are 45 different POS labels, and the three NP labels: begin-phrase, inside-phrase, and other. (Ramshaw and Marcus, 1995) To reduce the inference time, following (McCallum et al, 2003), we collapsed the 45 different POS labels contained in the original data. The rules for collapsing the POS labels are listed in the Table 3.

Original	Collapsed
all different types of nouns	NOUN
all different types of verbs	VERB
all different types of adjectives	JADJ
all different types of adverbs	RBP
the remaining POS labels	OTHER

Table 3: Rules for collapsing POS tags

Token	POS	Collapsed	Chunk	NP
U.K.	JJ	JADJ	B-NP	B
base	NN	NOUN	I-NP	I
rates	NNS	NOUN	I-NP	I
are	VBP	VERB	B-VP	O
at	IN	OTHER	B-PP	O
their	PRP$	OTHER	B-NP	B
highest	JJS	JADJ	I-NP	I
level	NN	NOUN	I-NP	I
in	IN	OTHER	B-PP	O
eight	CD	OTHER	B-NP	B
years	NNS	NOUN	I-NP	I
.	.	OTHER	O	O

Table 4: Example with POS and NP labels, before and after collapsing the labels.

We present two experiments: one comparing our label-chunk model with a cascaded linear-chain model and a simple linear-chain model, and one comparing different learning algorithms.

The cascaded linear-chain model uses one linear-chain model to predict POS tags, and another linear-chain model to predict NP labels, using the POS tags predicted by the first model as a feature.

More specifically, we trained a POS-tagger using the training set D. We then used the learned model and replaced the POS labels of the test set with the labels predicted by the learned model. The linear-chain NP chunker was again trained on D and evaluated on this new test set with POS supplied by the earlier processing. Note that the new test set has exactly the same word tokens and noun chunks as the original test set.

5.2 Systems

5.2.1 POS Tagger and NP Chunker

There are three versions of POS taggers and NP chunkers: CRF, VP, MMVP. For CRF, L-BFGS, a quasi-Newton optimization method was used for the training, and the implementation we used is CRF++ (Kudo, 2005). VP uses voted perceptron, and MMVP uses max margin update for the voted perceptron. For the voted perceptron, we used aver-

if x_q matches	then t_q is
[A-Z][a-z]+	CAPITAL
[A-Z]	CAP_ONE
[A-Z]+	CAP_ALL
[A-Z]+[a-z]+[A-Z]+[a-z]	CAP_MIX
.*[0-9].*	NUMBER

Table 5: Rules to create t_q for each token x_q

First Layer (POS)

Node $\langle C, P, r\rangle$	Trans. $\langle C, P0, P, r-1, r\rangle$
x_{r-1}	x_{r-1}
x_r	x_r
x_{r+1}	
t_r	

Second Layer (NP)

Node $\langle C, q, r\rangle$	Trans. $\langle C0, C, q-1, q\rangle$
x_q	x_{q-1}
x_{q-1}	x_q
x_r	
x_{r+1}	

Table 6: Lexicalized Features for Joint Models

aging of the weights suggested by (Collins, 2002). The features are exactly the same for all three systems.

5.2.2 Cascaded Models

For each CRF, VP, MMVP, the output of a POS tagger was used as a feature for the NP chunker. The feeds always consist of a POS tagger and NP chunker of the same kind, thus we have CRF+CRF, VP+VP, and MMVP+MMVP.

5.2.3 Joint Models

Since CRF requires the computation of marginals for each part, we were not able to use the learning method. VP and MMVP were used to train the label-chunk structures with the features explained in the following section.

5.3 Features

First, as a preprocessing step, for each word token x_q, feature t_q was created with the rule in Table 5, and included in the input files. This feature is included in x along with the word tokens. The feature tells us whether the token is capitalized, and whether digits occur in the token. No outside resources such as a list of names or a gazetteer were used.

Table 6 shows the lexicalized features for the joint labeling and chunking. For the first iteration of training, the weights for the lexicalized features were not updated. The intention is to have more weights on the unlexicalized features, so that when lexical feature is not found, unlexicalized features could provide useful information and avoid overfitting, much as back-off probabilities do.

POS tagging	POS	NP	F1
CRF	91.56%	N/A	N/A
VP	90.55%	N/A	N/A
MMVP	90.02%	N/A	N/A
NP chunking	**POS**	**NP**	**F1**
CRF	given	94.44%	87.52%
VP	given	94.28%	86.96%
MMVP	given	94.17%	86.79%
Both POS & NP	**POS**	**NP**	**F1**
CRF + CRF	above	90.16%	79.08%
VP + VP	above	89.21%	76.26%
MMVP + MMVP	above	88.95%	75.28%
VP Joint	88.42%	90.60%	79.69%
MMVP Joint	88.69%	90.84%	80.34%

Table 7: Performance

6 Result

We evaluated the performance of the systems using three measures: POS accuracy, NP accuracy, and F1 measure on NP. These figures show how errors accumulate as the systems are chained together. For the statistical significance testing, we have used pair-samples t test, and for the joint labeling and chunking task, everything was found to be statistically significant except for CRF + CRF vs VP Joint.

One can see that the systems with joint labeling and chunking models perform much better than the cascaded models. Surprisingly, the perceptron update motivated by the max margin principle performed significantly worse than the simple perceptron update for linear-chain models but performed better on joint labeling and chunking.

Although joint labeling and chunking model takes longer time per sample because of the time complexity of decoding, the number of iteration needed to achieve the best result is very low compared to other systems. The CPU time required to run 10 iterations of MMVP is 112 minutes.

7 Conclusion

We have presented the decoding algorithm for label-chunk structure and showed its effectiveness in finding two layers of information, POS tags and NP chunks. This algorithm has a place between the

POS tagging	Iterations
VP	30
MMVP	40
CRF	126
NP chunking	**Iterations**
VP	70
MMVP	50
CRF	101
Both POS & NP	**Iterations**
VP	10
MMVP	10

Table 8: Iterations needed for the result

Viterbi algorithm for linear-chain models and the CKY algorithm for parsing, and the time complexity is O(n^2). The use of our label-chunk structure significantly boosted the performance over cascaded CRFs despite the online learning algorithms used to train the system, and shows itself as a promising alternative to cascaded models, and possibly dynamic conditional random fields for modeling two layers of tags. Further work includes applying the algorithm to relation extraction, and devising an effective algorithm to find the marginal probabilities of parts.

References

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*

K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. 2003. Online passive aggressive algorithms. In *Advances in Neural Information Processing Systems 15*

K. Crammer, R. McDonald, and F. Pereira. 2004. New large margin algorithms for structured prediction. In *Learning with Structured Outputs Workshop (NIPS)*

Y. Freund and R. Schapire 1999. Large Margin Classification using the Perceptron Algorithm. In *Machine Learning*, 37(3):277-296.

T.S. Jaakkola, M. Diekhans, and D. Haussler. 2000. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*

T. Kudo 2005. CRF++: Yet Another CRF toolkit. Available at *http://chasen.org/˜taku/software/CRF++/*

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proc. of the 18th International Conference on Machine Learning (ICML)*

F. Peng and A. McCallum. 2004. Accurate Information Extraction from Research Papers using Conditional Random Fields. In *Proc. of the Human Language Technology Conf. (HLT)*

F. Sha and F. Pereira. 2003. Shallow parsing with conditional random fields. In *Proc. of the Human Language Technology Conf. (HLT)*

C. Manning and H. Schutze. 1999. *Foundations of Statistical Natural Language Processing* MIT Press.

A. McCallum, K. Rohanimanesh and C. Sutton. 2003. Dynamic Conditional Random Fields for Jointly Labeling Multiple Sequences. In *Proc. of Workshop on Syntax, Semantics, Statistics. (NIPS)*

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proc. of the 43rd Annual Meeting of the ACL*

L. Ramshaw and M. Marcus. 1995. Text chunking using transformation-based learning. In *Proc. of Third Workshop on Very Large Corpora. ACL*

C. Sutton, K. Rohanimanesh and A. McCallum. 2004. Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data. In *Proc. of the 21st International Conference on Machine Learning (ICML)*

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning 2004. Max Margin Parsing. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*

B. Taskar and D. Klein. 2005. Max-Margin Methods for NLP: Estimation, Structure, and Applications Available at *http://www.cs.berkeley.edu/˜taskar/pubs/max-margin-acl05-tutorial.pdf*

E. F. Tjong Kim Sang and S. Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proc. of the 4th Conf. on Computational Natural Language Learning (CoNLL)*

T. Zhang. 2001. Regularized winnow methods. In *Advances in Neural Information Processing Systems 13*