# Base Noun Phrase Chunking with Support Vector Machines

Alex Cheng
CS674: Natural Language Processing – Final Project Report
Cornell University, Ithaca, NY
ac327@cornell.edu

**Abstract**
We apply Support Vector Machines (SVMs) to identify base noun phrases in sentences. SVMs are known to achieve high generalization performance even in high dimensional feature space. We explore two different chunk representations (IOB and open/close brackets) and use a two-layer system approach for the classification task. Experiments show that despite using a dynamic programming approach to find the most probable pairing in the open/close brackets representation, the IOB representation performs significantly better. In addition, using higher degree polynomial kernel functions lead to slightly better results. We conclude that SVMs are extremely powerful machine learning approach for many natural language processing tasks and outperforms other learning systems because of SVMs' ability to generalize in high dimension.

## 1. Introduction

Base noun phrase (baseNP) chunking involves dividing sentences into non-overlapping segments of noun phrases. Calculating these noun phrase chunks are usually relatively computationally inexpensive and are often used as a precursor full parsing and further semantic analysis such as markings in noun-phrase co-reference. In addition, noun phrase chunks are used as multi-word indexing terms and are important for information retrieval and information extraction task.

Support Vector Machine (SVM) is a relatively new statistical machine learning approach for solving binary classification problem. Essentially, SVMs maximize the margin between critical training examples by solving a dual optimization problem. Kernel functions allow SVMs to combine the input features at relatively low computational cost and transform SVMs to non-linear classifiers.

Kudo and Matsumoto (2001) apply SVMs to NP chunking and achieve higher precision and recall in the standard data set than previously reported. They use a boosting method of weighted voting between SVMs trained with different chunk representations to identify noun phrase chunks.

In this paper, we apply SVMs to the NP chunking task, by using different system architectures from Kudo and Matsumoto. We introduce a two-layer system of SVMs that iterate in its classification stage until convergence. We examine the effect of higher degree polynomial kernel function on the chunk task. Moreover, we apply SVMs to a open/close brackets chunk representation. We analyze the effect of different parameters used for combining the brackets to find the most probably noun phrase chunks..

[ Pierre Vinken ], [ 61 years old], will join [ the board ] as [ a nonexecutive director ] [ Nov. 29 ].

Figure 1: Base NP Example (sentence extracted from WSJ)

## 1.1 Description Task

The definition of noun phrases can be ambiguous. In this work, we follow the definition proposed by Ramshaw and Marcus (1995) who develop a method for deriving and extracting NP chunks from the Penn Treebank. The goal of NP chunking is to identify the initial portions of non-recursive, non-overlapping noun phrases, including determiners but excluding postmodifying prepositional phrases or clauses. The bracketed portions of Figure 1, for example, show the base NPs.

## 2. Support Vector Machines

SVM is a relatively new machine learning approach based on statistical learning theory. SVMs are well-known for their good generalization performance and have been applied to many recognition problems. Recently, SVMs have been applied to natural language processing tasks such as chunking (Kudo & Matsumoto, 2001) and text classification (Joachims, 2002). In particular, these two specific NLP systems are reported to have achieved higher accuracy than most state of the art systems (both learning and knowledge-based approaches). There are theoretical and empirical results that indicate the good performance of SVMs' ability to generalize in a high dimensional feature space without over-fitting the training data (Joachims, 2002).

### 2.1 Optimal Hyperplane

Essentially, SVM is a linear classifier for solving binary classification problem. We will first introduce the linear Hard Margin SVM which operates under the assumption that the training examples are linearly separable. We then generalize it to a Soft Margin SVM which we use for our work in NP chunking. We can define the problem as follows:

We are given a training set *S* containing examples $x_i$, a feature vector of dimension *d*, each $x_i$ belong either a positive negative class:

$$(x_1, y_1),\ldots,(x_n, y_n) \qquad x_1 \in \Re^d \qquad y_1 \in \{+1,-1\}$$

A Hard Margin SVM seeks to (1) find an optimal separating hyperplane *h* that classify all training examples correctly where $h = w \bullet x + b$ and *w* is the orthogonal vector and *b* is the offset from the origin. We can measure the optimality of the hyperplane by the margin, the minimum distance from *h* to the training example of both classes. Intuitively, the a hyperplane *h* with a larger margin has the highest generalization power. It is not hard to show that the minimum distance between h and the training example is $1 / ||w||$[1].

[1] $(w \bullet x_1) + b = 1$
$(w \bullet x_2) + b = -1$

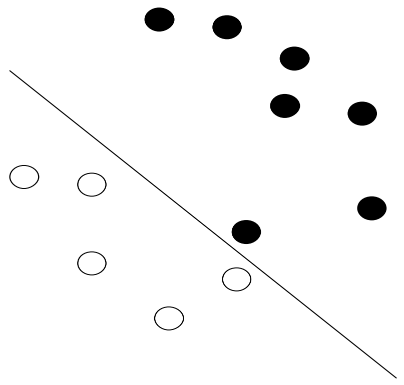


Figure 2: A Hard Margin SVM will not perform well

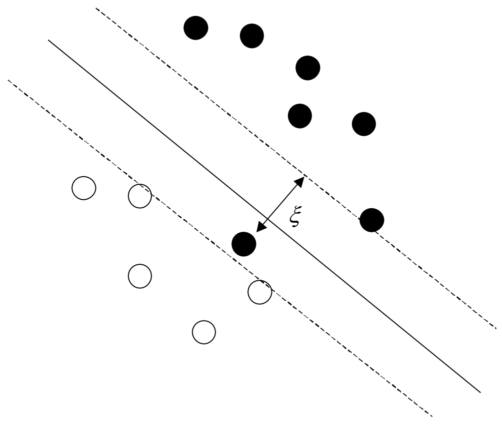


Figure 3: The training error for the example on the wrong side of the hyperplane.

Therefore, by minimizing *w,* we maximize the margin. So Hard-Margin SVM seeks *w* and *b* such that

Minimize: $\frac{1}{2} w \bullet w$

Subject to: $\forall_{i=1}^{n} : y_i[w \bullet x + b] \geq 1$

Hard Margin SVMs find a hyperplane that separate the training examples with perfect accuracy. However, a Hard-Margin SVM does not usually work in practice because of noise in the training data (see Figure 2) or a separating hyperplane may not exist. Therefore, a Soft Margin SVM allows for such training noise by introducing an error parameter *C*. The error for each training example $\xi_i$ is measured by the distance between the example and the margin of the example's class (see Figure 3). So the optimization problem becomes the following:

Minimize: $\frac{1}{2} w \bullet w + C\sum_{i=1}^{n} \xi_i$

Subject to: $\forall_{i=1}^{n} : y_i[w \bullet x + b] \geq 1 - \xi_i \qquad \forall_{i=1}^{n} : \xi_i > 0$

Clearly, as $C \rightarrow \infty$, we have the essentially a Hard-Margin SVM since any error will cost the minimizing term to approach infinity. So the parameter *C* controls how much "slack" we give to the SVM which can be adjusted based on knowledge of the learning task.

For computational reasons, it is easier to solve the Wolfe dual of the optimization problem. This can be derived using Lagrange multipliers[2]. This optimization problem can be solved by quadratic programming.

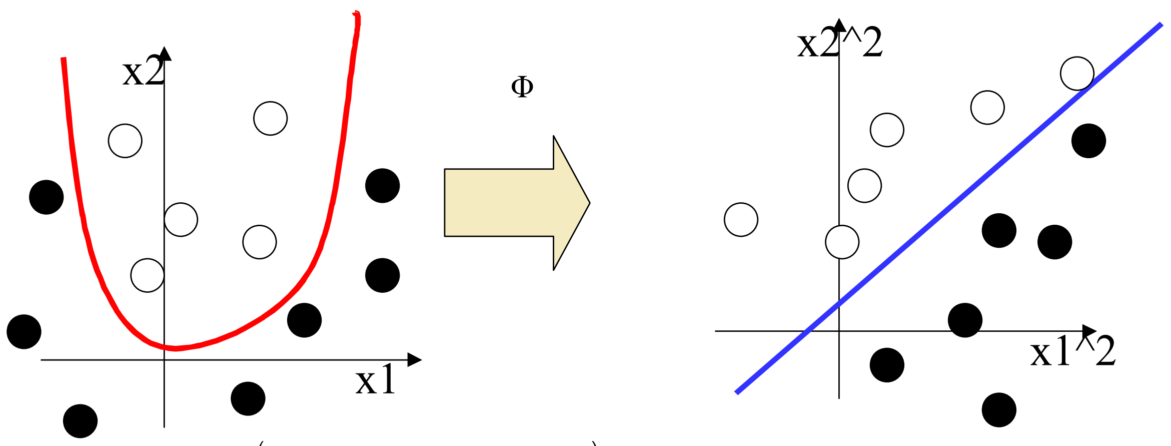
---

⇨ $(w \bullet (x_1 - x_2)) = 2$

⇨ $(w/ ||w|| \bullet (x_1 - x_2)) = 2 / ||w||$ (this is the min distance between a positive and negative examples)

⇨ *Margin* $= 1 / ||w||$

[2] Details can be find in Scholkopf's Tutorial on Statistical Learning and Kernel Methods (2000).



$$\Phi(x) = \Phi(x_1, x_2) = \left(x_1{}^2, x_2{}^2, \sqrt{x_1 x_2}, \sqrt{x_1}, \sqrt{x_2}, 1\right)$$
$$K(x, x') = \Phi(x) \bullet \Phi(x') = (x \bullet x' + 1)^d$$

Figure 4: Kernel function to map from a linear space to a 2nd degree polynomial feature space.

Minimize: $$-\sum_{i=1}^{n} \alpha_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i \bullet x_i)$$

Subject to: $$\sum_{i=1}^{n} y_i \alpha_i = 0 \qquad \forall_{i=1}^{n} : 0 \le \alpha_i \le C$$

All training examples with $\alpha_i > 0$ are called support vectors. These training examples define the position of the optimal hyperplane. We can view the $\alpha_i$ as the relative "forces" or weights acting on the hyperplane.

## 2.2 Kernel Functions

In many classification tasks, the training examples, represented in *d* dimensional feature vectors, are distributed such a way that a linear classifier in the current feature space cannot perform well (see Figure 4). A mapping to a feature space in higher dimension is sometimes needed to linearly separate the training examples. In the case of artificial neural networks, the hidden layers increase the expressive power of the learning system. For SVM, one can introduce kernel functions to map the training examples into a feature space with higher dimension. Nomrally, a transformation from the original feature space of the training examples to a higher dimensional feature space is computational expensive. But notice that in our optimization problem, we are only calculating the dot products between training examples. Suppose we have a polynomial transformation $\Phi : \Re^d \to \Re^{d'} \; d < d'$, then normally we will calculate the dot product $\Phi(x) \bullet \Phi(x')$. Now we can introduce a kernel function $K$ such that $K(x, x') = \Phi(x) \bullet \Phi(x')$. In a polynomial transformation, $K(x, x') = (x \bullet x' + 1)^d$. Kernel function allows this transformation of feature space to a higher dimension with relatively low extra computational cost.

## 2.3 Feature Combination with Kernels

Polynomial kernel functions are especially interesting when used for Natural Language Processing tasks. N-gram can be modeled using kernel functions. Traditionally, if we want to model bi-gram in most machine learning approaches, we have to explicitly add the features $w_i w_j$ to denote the bigram $w_i w_j$. With this increase of feature space, many machine learning approaches will likely to overfit the training data and loses its generalization power. Therefore, these features are usually heuristically chosen to avoid this problem. However, for SVMs, using kernel functions, we can represent n-grams using an n-degree polynomial kernel function. For example, suppose we have the lexical features and Part of Speech (POS) features for the noun phrase "The car" with its corresponding POS "DT" "NN". Using a 2nd degree kernel functions, the new feature space consists of all possible pairs: ("The car", "The DT", "The NN", "car DT", "car NN", "DT NN"). The kernel function allows this additional feature to be added to the learning system with minimal extra computational cost. In addition, the hyperplane computed by SVM in this new feature space will be optimal and maximizes the margin between training examples in this feature space. Therefore, SVM does not lose its generalization power with this transformation.

## 3. Chunk Representation

Ramshaw and Marcus (1995) have introduced a data representation for chunking by converting it to a tagging task. Most of the recent work on NP chunking uses this tagging scheme. Different representations have been proposed, but it has been found that tag representation has only a minor influence on the performance of the system (Sang, 1999). For our work, we decide to experiement with two different tag representations, IOB and open/close brackets which represents the two major classes of chunk representation that have been used in previous experiments.

### 3.1 IOB Tag

The chunk set tag {I, O, B} has been originally proposed by Ramshaw and Marcus (1995). These tags are used to indicate the boundaries for each NP chunk:
**I** – the current word is inside a NP chunk
**O** – the current word is outside a NP chunk
**B** – the current word is the beginning of a chunk which immediately follows another chunk.

(Figure 5) is an example of using IOB tags to represent NP chunks.

### 3.2 Open/Close Bracket

Another chunk representation is the open / close brackets. We can also view this as a tagging task.
**[** – Denotes the beginning of a chunk
**]** – Denotes the end of a chunk
**.** – Denotes everything else that is not a beginning or the end.
Note that a word can be tagged with both **[** and **]** if the chunk contains only one word.

(Figure 6) is an example of using open/close brackets to represent NP chunks.

```
Pierre(I)   Vinken(I)       ,(O)    61(I)    years(I)
  old(O)       ,(O)    will(O)    join(O)    the(I)
    board(I)    as(O)     a(I)    nonexecutive(I)
        director(I)    Nov.(B)     29(I)    .(O)
```

Figure 5: Example of a sentence tagged with IOB notations.

```
Pierre([)   Vinken(])       ,(.)    61([)    years(])
  old(.)       ,(.)    will(.)    join(.)    the([)
   board(])    as(.)       a ([)    nonexecutive(.)
        director(])    Nov.([)     29(])    .(.)
```

Figure 6: Example of a sentence tagged with open/close brackets.

Essentially, both representations are equivalent in terms of expressive power. One can easily convert from one representation to another. Sang and Veenstra (1999) find that different representations has minior influence on the performance. Their experiments consist of using 7 different chunk representation, 4 are variant of the IOB tag, and 3 are variant of the open/close bracket. Therefore, we decide to use these two classes of chunk representation for our work.

## 4. Features

SVMs are binary classifiers; therefore, for multi-class problem such as NP chunking, we have to extend SVM to a multi-class classifier. There are generally two approaches: a *pairwise* method and a *one vs. all other* method. In the *pairwise* method, each class is trained against examples from another class, producing *n(n-1)/2* SVMs for an n-class classification problem. For the IOB tag representations, there will be SVMs trained to classify (tag I vs. tag O), (tag I vs. tag B) and (tag O vs. tag B). In the *one vs. all other* method, exactly *n* SVMs are trained and each SVM is trained to classify examples of one label vs. all examples that are not of that same label (for example, tag I vs. not Tag I {Tag O and Tag B}). As far as we can know, there are currently no theoretical results which show the advantage of one system over the other. In practice, the *pairwise* method is more computational tractable. Despite having polynomially more classifiers to train, the reduction in training size increases the computation speed of the optimiziation task. Kudo and Matsumoto (2000) use the *pairwise* approach for their SVMs for NP chunking. We decide to use the *one vs. all other* approach to compare the results of different approach..

The set for features used for training the SVMs are all the information available in the surrounding context, such as word, part of speech as well as chunk labels (IOB tags or open/close brackets) using a window of five:

| | | | | | |
|---|---|---|---|---|---|
| Word | wi-2 | wi-1 | wi | wi+1 | wi+2 |
| Capitalize | cpi-2 | cpi-1 | cpi | cpi+1 | cpi+2 |
| POS | pi-2 | pi-1 | pi | pi+1 | pi+2 |
| Chunk | ci-2 | ci-1 | ? | ci+1 | ci+2 |

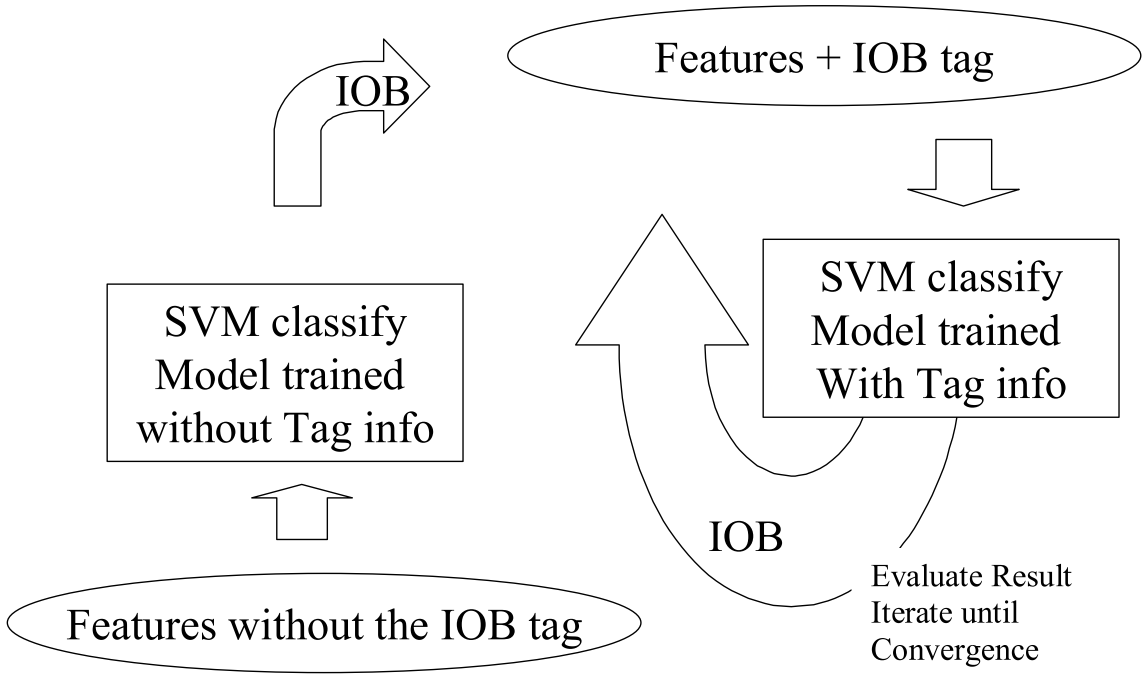

Features + IOB tag
IOB
SVM classify Model trained With Tag info
SVM classify Model trained without Tag info
IOB
Evaluate Result Iterate until Convergence
Features without the IOB tag

Figure 7: Architecture for IOB Predictor

More specifically, for the words, we stem each word using porter stemmer and change everything to lower case. We added an additional feature for each word to signify if it is capitalized or not. For the part of speech tags (POS), we use the POS provided in the Treebank for both the testing and training. The chunk labels of the surround words are provided during the training phrase. However, during the testing phrase, these tags labels are not known. Therefore we train two sets of classifiers, one with the chunk label features and one without and connect the output of one as an input to the other.

## 5. System Architecture

We develop two systems, one for the IOB tags representation and one for the Open/Close brackets representation. The main difference between the approaches is that for the open/close predictor, we have to use dynamic programming to combines the open and close bracket to form the most probable bracket pairs.

### 5.1 IOB Tag Predictor

Two predictors are learned, which differ in their input features. The first predictor takes in the word itself and its POS as well as these information for its surrounding words. The first predictor outputs the tag prediction for each word. The second predictor takes in the word and POS information as well as the out of the first predictor as its input feature vector. Using this second model, the system outputs the predicted tag labels. These tag labels will then act as input features to the second predictor. This process iterates until the accuracy of the system converges (see Figure 7).

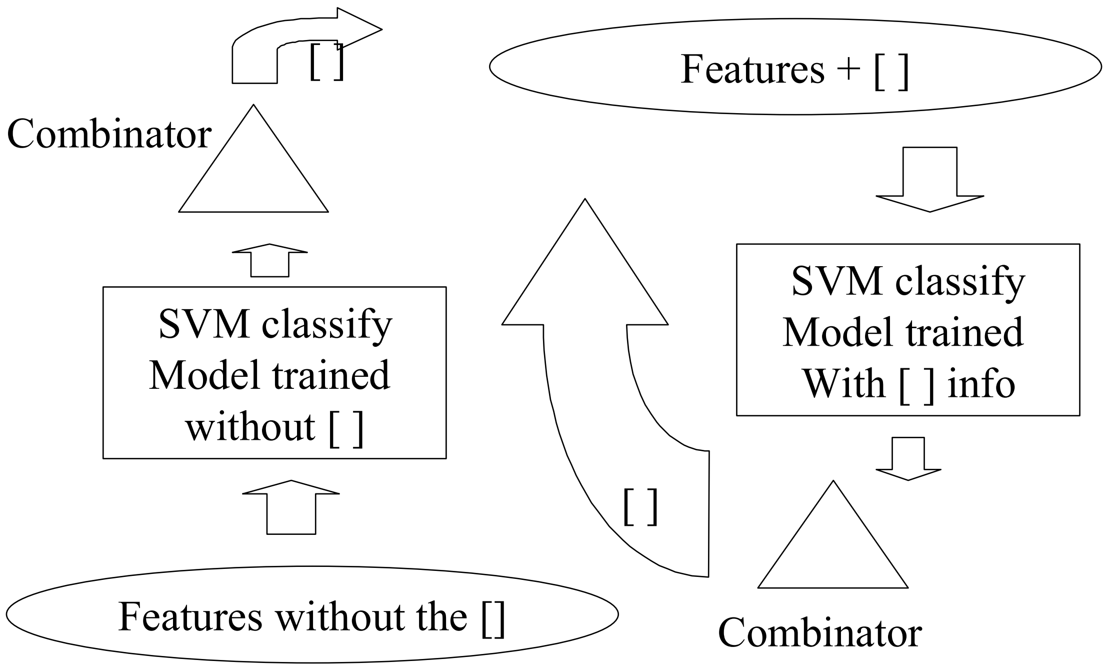

[ ]
Features + [ ]
Combinator
SVM classify Model trained With [ ] info
SVM classify Model trained without [ ]
[ ]
Features without the []
Combinator

Figure 8: Architecture for Open/Close Predictor

**5.2 Open/Close Brackets Predictor**
The system architecture of the open/close brackets predictor is essentially the same as the IOB Tag predictor except for the combinator. The combinator is necessary because the two SVMs at each stage classify opposing brackets – one seeks to classify a word as [ another SVM seeks to classify a word as ]. These pair of brackets need to consistent and that number of [ must equal the number of ]. For each prediction, SVM outputs certainty score[3] and normally, we classify the example to be in class +1 if the certainty score is greater than 0 and -1 if it is less than 0. If we simply use this approach, many of the bracket pairs will be overlapping, which does not form NP chunks and that the number of brackets will certainly not be the same for open and close brackets. Therefore we need to use the combinator to find the most probable bracket pair. (see Figure 8)

**5.2.1 Combinator**
The combinator proposed here is similar to the one used in SNoW, (Spare Network of linear separator) which is a learning approach for shallow parsing (Munoz et al. 1999). SVM outputs a certainty value for the classification of each example. We use this certainty value to maximize the most probable bracket pairs.

For each example classified by the SVM let $v_o(i)$ be denote the certainty score output by the open bracket classifier for example $i$ and $v_c(i)$ be the certainity score by the close bracket classifier We call $p = (i,j)$ a pair where i is the position of the open bracket and j is the position of the close bracket and that $i <= j$. Note that $i = j$ if the chunk consist of only one word. Then we define $v(p) = v_o(i)*v_c(j)$. Two pairs $(i,j)$ and $(i',j')$ are

[3] Certainty score = w•x + b for a linear SVM

| Data | Sentences | Words | NP Patterns |
|---|---|---|---|
| Training (WSJ 15-18) | 8936 | 211727 | 54758 |
| Testing (WSJ 20) | 2012 | 47377 | 12335 |

Table 1: Sizes of the training and test data sets (Munoz et al. 1999)

*compatible* if $j > i'$. Then for each sentence $s$, we want to maximize the set of *compatible* pairs $P$ such that the sum of all the values is the maximum. Since the number of possible pairs in a sentence of size $m = |s|$ is $m!$, we introduce a threshold $t$ to reduce the number of candidate pair consider in the problem. More precisely, we set $v_k(i)' = v_k(i) + t$ *and* we only consider brackets in which $v_k(i)' > 0\ k \in \{o,c\}$. Then the $v(p)' = v_o(i)' * v_c(j) > 0$ for all possible pair. In addition, we introduce a distance parameter $d$. We find empirically that some of chunks $p = (i,j)$ consist of too many words because
$v(p) > \sum v(p'),$ for some $p'=(i',j')$ $i < i'$ and $j > j'$. Thus, this parameter $d$ adds value to $v(p)$ depending on the distance between the open bracket and the close bracket. More precisely, $v(p = (i,j)) = v_o(i) * v_c(j) + d / (j-i+1)$.
Given this formulation, we can extract a list of possible candidates with their value. We can then use a standard dynamic programming approach[4] to find the optimal set of pairings for each sentence $s$.

## 6 Experiment and Results

For our experiments, we use the software package SVM-light (Joachims 2000). We vary the kernel functions (2nd degree polynomial vs. 3rd degree polynomial) and keep the cost parameter C constant (1). We use the standard dataset proposed originally by Ramshaw and Marcus (1995). The training data consists of 4 sections (15-18) of the WSJ part of the Penn Tree bank. The test data consists of one section (20). The size of the training and test data are summarized in Table 1.

### 6.1 Experiment Setting

*Experiment 1:* For the baseline system, we use the part of speech information only. For each word, we assign the chunk tag that is most frequently associated with that part of speech tag in training.

*Experiment 2:* This is the first predictor of the IOB system using 2nd degree polynomial with no tag info as features.

*Experiment 3:* This is the first predictor of the IOB system using 3rd degree polynomial with no tag info as features.

*Experiment 4:* This is the second predictor of the IOB system using 2nd degree polynomial with surrounding tag info as features. The tag info is the result from the first predictor (*Experiment 2*).

[4] This problem is equivalent to the Weighted Interval Scheduling Problem see (www.cs.cornell.edu/courses/cs482/), which can be solved in O(n) time where n is the number of candidate pairs.

*Experiment 5:* This is the second predictor of the IOB system using 3rd degree polynomial with surrounding tag info as features. The tag info is the result from the first predictor (*Experiment 3*).

*Experiment 6:* This is the first predictor for the open/close bracket system using 2nd degree polynomial with no tag info as features. We experiment with different parameters of threshold *t* and distance value d to find the best result for the combinator.

*Experiment 7:* This is the second predictor for the open/close bracket system using 2nd degree polynomial with surrounding tag info as features. The tag info is the result of best (highest $F_{\beta = 1}$) bracket pairs found by the first predictor going through the combinator and best parameters selection (*Experiment 6*).

## 6.2 Evaluation

An evaluation script[5] has been provided by Erik Sang for CoNNL-2000. This script uses the IOB tag representation and measures precision (percentage of chunks found that are correct), recall (percentage of correct chunks found) and F-measure [$\beta = 1$] (2*precision*recall / (precision + recall). Both ends of a chunk have to match exactly for it to be counted. For the open/close bracket system, we convert the brackets into IOB tags, and feed this converted representation into the evaluation script. Since the two representations are equivalent in expressive power, performance is not affected by the conversion. We have achieved an F-measure of 94.14% which is slightly worse than the state of the art system (94.22% Kudo and Matsumoto, 2001)

Results are summarized in Table 2.

## 6.3 Effect of Chunk Representation

We find that the IOB representation performs significantly better than the open/close bracket representation. We speculate that with the IOB representation, we use three SVMs to classify each tag, whereas for the open/close bracket representation, we have two SVMs to classify the open and close bracket. It seems that the increase number of SVMs simplify the learning problem. Each additional class reduces the number of positive examples for each SVM, thereby making the data easier to separate. Thus, the number of support vectors decreases as the number of class increases, reducing the complexity of the hypothesis space.

One of the major problems with the Open/Close system is that for the second predictor, we use the best overall output from the first predictor. However, the best set of pairings result from the first predictor using some paramenters t,d might not be the best input for the second predictor. Therefore, the performance results from the Open/Close system is not as good as the IOB system.

[5] http://cnts.uia.ac.be/conll2000/chunking/conlleval.txt

| Ex. | System | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|---|
| 1 | Baseline | 78.2% | 81.9% | 80.01% |
| 2 | IOB (no tag) degree-2 | 93.50% | 93.91% | 93.70% |
| 3 | IOB (no tag) degree-3 | 93.43% | 93.74% | 93.58% |
| 4a | IOB (tag) degree-2 (first iteration) | 93.94% | 94.03% | 93.99% |
| 4b | IOB (tag) degree-2 (converged) | 94.06% | 94.07% | 94.07% |
| 5a | IOB (tag) degree-3 (first iteration) | 94.05% | 94.06% | 94.06% |
| 5b | IOB (tag) degree-3 (converged) | 94.14% | **94.13%** | **94.14%** |
| 6a | Open/Close (no tag) d-2 (best precision)<br>Parameter $t = 0$, $d = 4.8$ | 94.90% | 90.97% | 92.89% |
| 6b | Open/Close (no tag) d-2 (best recall)<br>Parameter $t = 1.0$, $d = 0.8$ | 90.82% | 93.16% | 91.97% |
| 6c | Open/Close (no tag) d-2 (best F-measure)<br>Parameter $t = 0.25$, $d = 4.8$ | 94.07% | 92.03% | 93.04% |
| 7a | Open/Close (no tag) d-2 (best precision)<br>Parameter $t = 0$, $d = 4.8$ | **94.65%** | 91.14% | 92.86% |
| 7b | Open/Close (no tag) d-2 (best recall)<br>Parameter $t = 1.0$, $d = 1.2$ | 92.92% | 93.70% | 93.31% |
| 7c | Open/Close (no tag) d-2 (best F-measure)<br>Parameter $t = 0.75$, $d = 2.0$ | 93.49% | 93.17% | 93.33% |

Table 2: Result of the seven experiments (the best result in each category is in bold)

### 6.3.1 Parameters for the Combinator

The threshold parameters *t* and distance parameters *d* directly affects the precision and recall of the system (see Appendix A). In particular, decreasing the parameter *t* improves the precision of the system, but the performance for recall drops. This is intuitive because the decrease of threshold *t* means that fewer candidates are considered and that the system is more certain of the correctness of the classification. However, many possible candidates are missed as a result and therefore recall rate drops. The precision and recall of the system seems to be directly correlated with the value of the threshold *t*.

The distance parameter *d* varies the bias for shorter chunks (chunks containing fewer words). The larger the parameter *d*, the more the system is biased towards shorter chunks. This is interesting because as we considered more candidates (threshold *t* is small), the precision of the system increases as *d* increases. This probably allows the system to eliminate many possible long chunks that could lead to poor performance. However, as the threshold *t* increase, the increase of *d* leads to poor performance. The distance parameter does not seem to have any significant effect on the recall rate.

## 6.4 Effect of different Kernel Functions

For our experiments, due to time constraints, we only train our system with a 2nd and 3rd degree polynomial kernel function. We find that the 3rd degree polynomial kernel function perform slightly better. One of the reasons is that by using the 3rd degree polynomial kernel, we can model tri-grams as our features, which increases the power of the learning system.

We find that in our iterative approach, the system converges relatively quickly. After an average of four passes, there are no longer changes in precision or recall of the system. This iterative approach performs better than the any of the system trained for using a single chunk representation. The best result thus far is the SVM approach by Kudo and Tatsumoto (2001), and the best F-measure for the a single chunk presentation is 94.11% and our system achieve a slightly better result of 94.14%.

## 7. Related Work

The approach to view chunking as a tagging problem by encoding the chunk structure in tags attached to each word is first proposed by Ramshaw and Marcus (1995). They put forward a standard dataset for this chunking task and derive a simple algorithm for extracting base noun phrase from the Penn Tree bank. They apply a transformation based learning to chunking by tagging the text, using similar techniques from Brill's Part of Speech tagger and hows that baseNP recognition (F(beta = 1) = 92.0) is easier than finindg both NP and VP chunks and that increasing the size of the training data increases the performance on the test set.

Other memory based sequence learning approach has been proposed. In one particular system, Cardie and Pierce (1998) use a method that uses only POS tag sequence to form complete baseNPs achieve a surprisingly good performance (F(beta = 1) = 90.9) without any lexical information.

Munoz et al (1999) develop a learning system (SNoW) which uses a sparse network of linear functions over a predefined or incrementally learned feature space. The two goals of their research are to examine the contribution of chaining – outputs from one of the predictors are used as input features of another predictor, and to compare between the Inside/Outside (IOB) and the Open/Close approach. They show that the Open/Close approach performs better than the Inside/Outside one. Their systems achieve an F-measure of 92.8%. In particular, for the Open/Close approach, the lexical information along with the use of the output feature of the first predictor has significant effect on the performance of the system.

Kudo and Matsumoto (2001) apply SVM to NP chunking. They use 4 different chunk representations and used a voting scheme to classify the chunk. Instead of using a two-layer system, they incorporate a parsing method for their SVM classification. Specifically, in a forward parsing direction, the chunk label of the previous two words are used as input features for the classification of the current tag. Thus they have 8 systems in total – 4 different chunk representations and 2 parsing direction. They then compared different voting methods: uniform, Cross validation,VC Bound and leave one out error. They find that the VC Bound is the best predictor for the performance of the system. Their system achieve the best result thus far, a F-measure of 94.22%.

## 8. Future Work and Concluding Remarks

We have trained two different learning systems for the NP chunking task using two different representations and found that the IOB representation performs significantly better than the open/close brackets representation. However, the differences in the kernel functions do not perform significantly better. The parameters for the combinator can adjust the precision and recall power of the system.

One disadvantage of a two-pass system is that the computational cost for training and classification is high, especially with SVM in a high dimensional feature space. This prohibits the use of SVM from many of the real-time application such as information retrieval.

## 8.1 Future Work

- Adding a bag of words approach to as a feature to the SVM. We can then use the context of the whole sentence, with an increase in windows size independent of the ordering.
- For the open/close bracketing approach, one can find a more robust way of selecting the parameters. In particular, tuning the parameter can lead to better performance in either precision or recall. We can use the result from these different predictors as input features to the second predictor. We believe this approach will allow the system to perform as well, if not better than the IOB representation.
- We can incorporate a boosting method by using a weighted voting between the IOB system and the open/close system.
- One reason why IOB representations perform better is that three classifiers are trained to determine the tag as opposed to two in the open/close bracket representation. We can try using a tag representation with more than three different tags and see if SVM performs better.

## Reference

Claire Cardie and David Pierce. 1998. Error-Driven Pruning of Treebank Grammars for Base Noun Phrase Identification. In *Proceedings of COLING-ACL'98*, pages 218-224

Thorsten Jaochims. 2000 SVM-light version 5.0 http://svmlight.joachims.org/

Thorsten Joachims. 2002. *Learning to classify text using support vector machines – methods, theory, and algorithms*. Kluwer.

Taku Kudo and Yuji Matsumoto. 2000. Use of Support Vector Learning for Chunk Identification. In *Proceedings of the 4th Conference on CoNLL-2000 and LLL-2000*, pages 142-144.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with Support Vector Machines. In *Proceedings of the NAACL 2001*, pages 192-199.

Marcia Muñoz, Vasin Punyakanok, Dan Roth and Dav Zimak. 1999 A Learning Approach to Shallow Parsing, In *Proceedings of EMNLP/WVLC-99,* pages 67-73

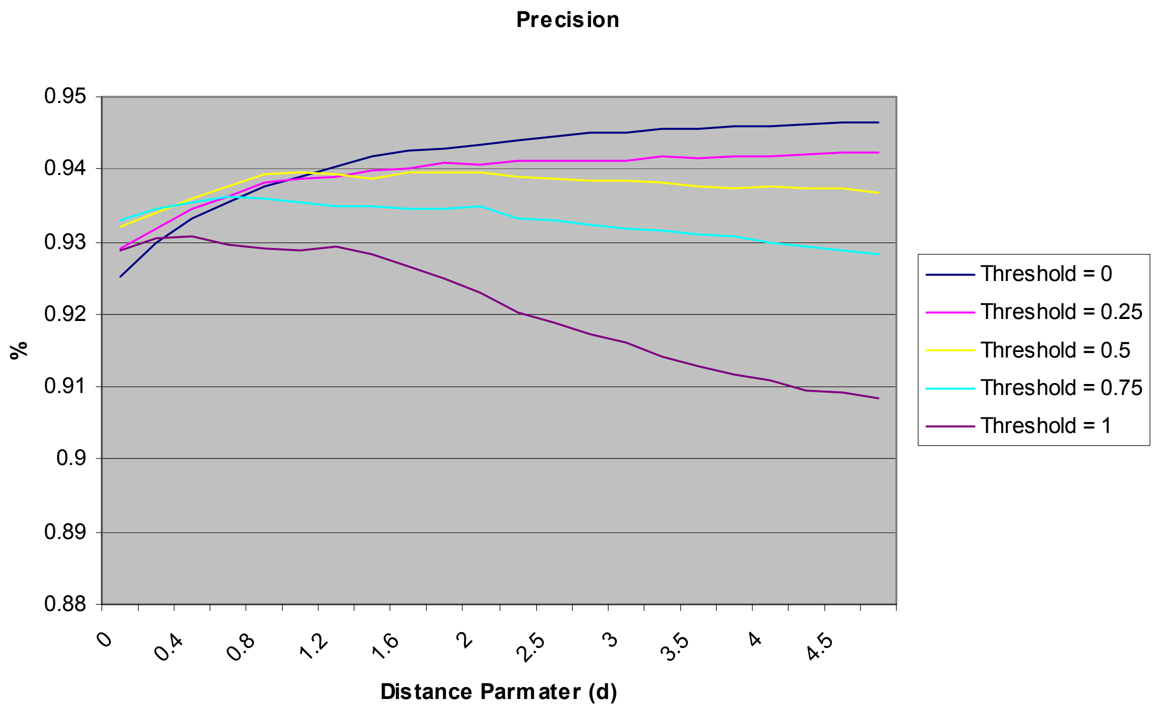
Lance A. Ramshaw and Mitchell P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the 3rd Workshop on Very Large Corpus*, pages 88-94.

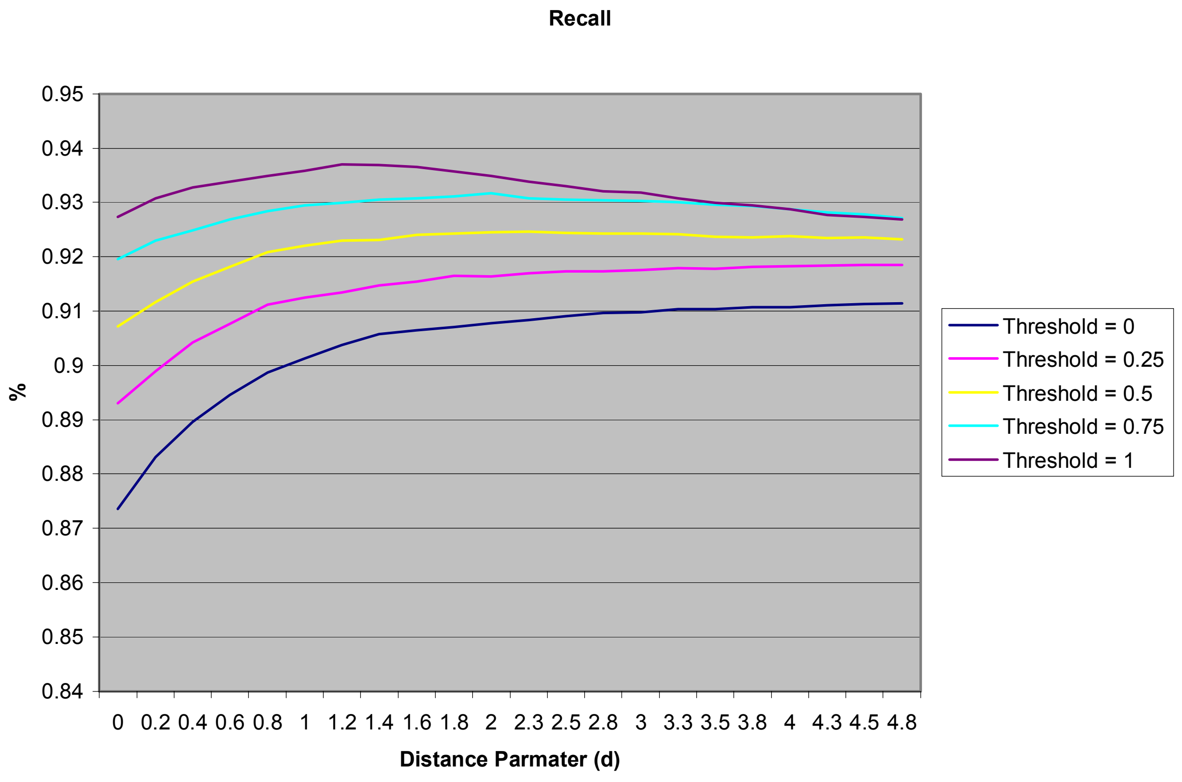
Erik F. Tjong Kim Sang and John Veenstra. 1999. Representing text chunks. In *Proceedings of EACL 1999*, pages 173-179.

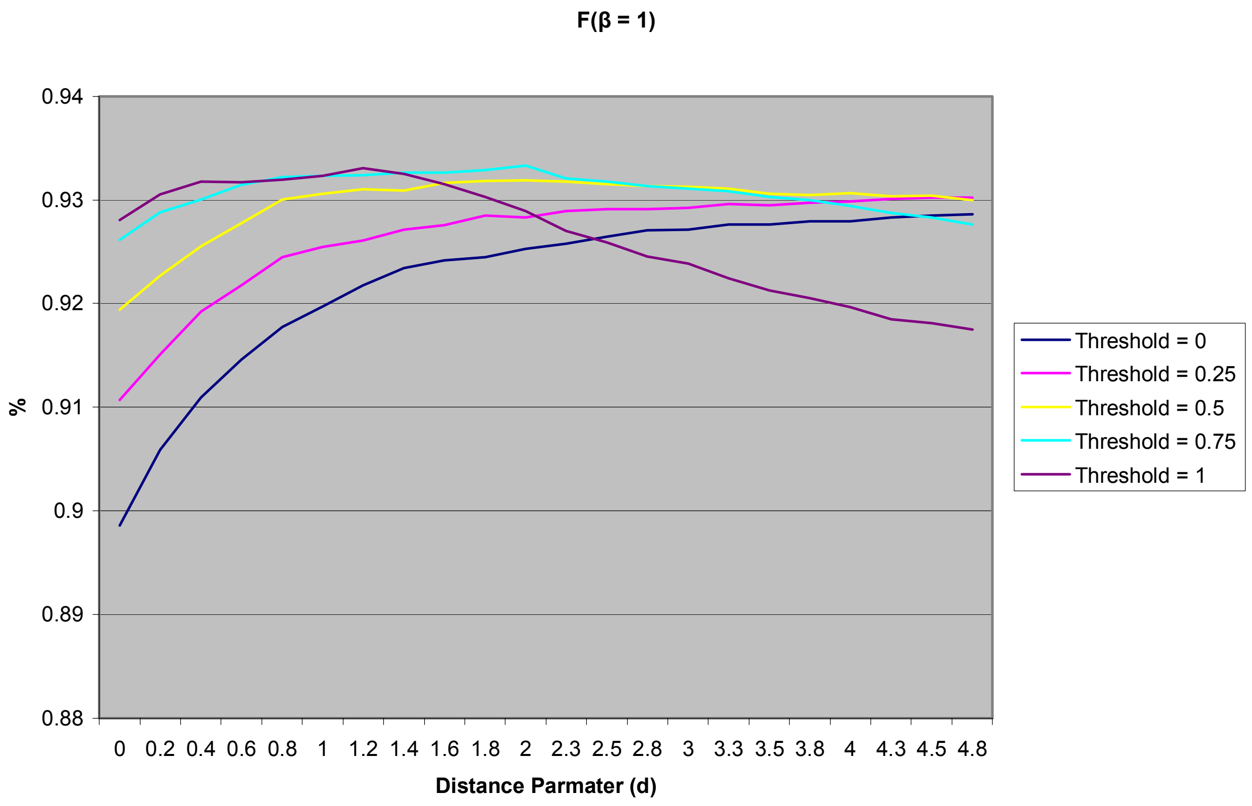
Bernhard Scholkopf. 2000. Statistical Learning and Kernel Methods. Technical Report MSR-TR-2000-23

## Appendix A

Open/Close Bracket Predictors: results using different parameters t, d.






F(β = 1)
0.94
0.93
0.92
0.91
0.9
0.89
0.88
%
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 2.3 2.5 2.8 3 3.3 3.5 3.8 4 4.3 4.5 4.8
Distance Parmater (d)
Threshold = 0
Threshold = 0.25
Threshold = 0.5
Threshold = 0.75
Threshold = 1