RC22011 (W0104-005) 4 April 2001 Computer Science

IBM Research Report

Text Chunking Using Regularized Winnow

Tong Zhang
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

Text Chunking using Regularized Winnow

Tong Zhang TZHANG@WATSON.IBM.COM
Fred Damerau DAMERAU@WATSON.IBM.COM
David Johnson DEJOHNS@US.IBM.COM
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 USA

Abstract

Many machine learning methods have recently been applied to natural language processing tasks. Among them, the Winnow algorithm has been argued to be particularly suitable for NLP problems, due to its robustness to irrelevant features. However in theory, Winnow may not converge for non-separable data. To remedy this problem, a modification called regularized Winnow has been proposed. In this paper, we apply this new method to text chunking. We show that this method achieves state of the art performance with significantly less computation than previous approaches.

1. Introduction

Recently there has been considerable interest in applying machine learning techniques to problems in natural language processing. One method that has been quite successful in many applications is the SNoW architecture (Dagan et al., 1997; Khardon et al., 1999). This architecture is based on the Winnow algorithm (Littlestone, 1988), which in theory is suitable for problems with many irrelevant attributes. In natural language processing, one often encounters a very high dimensional feature space, although most of the features are irrelevant. Therefore the robustness of Winnow to high dimensional feature space is considered an important reason why it is suitable for NLP tasks.

However, the convergence of the Winnow algorithm is only guaranteed for linearly separable data. In practical NLP applications, data are often linearly non-separable. Consequently, a direct application of Winnow may lead to numerical instability. A remedy for this, called regularized Winnow, has been recently proposed in (Zhang, 2000). This method modifies the original Winnow algorithm so that it solves a regularized optimization problem. It converges both in the linearly separable case and in the linearly non-separable case. Its numerical stability implies that the new method is more suitable for practical NLP problems that may not be linearly separable.

In this paper, we apply regularized Winnow to text chunking tasks. In order for us to rigorously compare our system with others, we use the CoNLL-2000 shared task dataset (Sang & Buchholz, 2000), which is publicly available from *http://lcg-www.uia.ac.be/conll2000/chunking*. An advantage of using this dataset is that a large number of state of the art statistical natural language processing methods have already been applied to the data. Therefore we can readily compare our results with other reported results.

We show that state of the art performance can be achieved by using the newly proposed regularized Winnow method. Furthermore, we can achieve this result with significantly less computation than earlier systems of comparable performance.

The paper is organized as follows. In Section 2, we describe the Winnow algorithm and the regularized Winnow method. Section 3 describes the CoNLL-2000 shared task. In Section 4, we give a detailed description of our system that employs the regularized Winnow algorithm for text chunking. Section 5 contains experimental results for our system on the CoNLL-2000 shared task. Some final remarks will be given in Section 6.

2. Winnow and Regularized Winnow for Binary Classification

We review the Winnow algorithm and the regularized Winnow method. Consider the binary classification problem: to determine a label $y \in \{-1, 1\}$ associated with an input vector x. A useful method for solving this problem is through linear discriminant functions, which consist of linear combinations of the components of the input variable. Specifically, we seek a weight vector w and a threshold θ such that $w^Tx < \theta$ if its label $y = -1$ and $w^Tx \geq \theta$ if its label $y = 1$.

For simplicity, we shall assume $\theta = 0$ in this paper. The restriction does not cause problems in practice since one can always append a constant feature to the input data x, which offsets the effect of θ.

Given a training set of labeled data $(x^1, y^1), \dots, (x^n, y^n)$, a number of approaches to finding linear discriminant functions have been advanced over the years. We are especially interested in the Winnow multiplicative update algorithm (Littlestone, 1988). This algorithm updates the weight vector w by going through the training data repeatedly. It is mistake driven in the sense that the weight vector is updated only when the algorithm is not able to correctly classify an example.

The Winnow algorithm (with positive weight) employs multiplicative update: if the linear discriminant function misclassifies an input training vector x^i with true label y^i, then we update each component j of the weight vector w as: $w_j \leftarrow w_j \exp(\eta x^i_j y^i)$, where $\eta > 0$ is a parameter called the learning rate. The initial weight vector can be taken as $w_j = \mu_j > 0$, where μ is a prior which is typically chosen to be uniform.

There can be several variants of the Winnow algorithm. One is called balanced Winnow, which is equivalent to an embedding of the input space into a higher dimensional space as: $\tilde{x} = [x, -x]$. This modification allows the positive weight Winnow algorithm for the augmented input $\tilde{x}$ to have the effect of both positive and negative weights for the original input x.

One problem of the Winnow online update algorithm is that it may not converge when the data are not linearly separable. One may partially remedy this problem by decreasing the learning rate parameter η during the updates. However, this is rather ad hoc since it is unclear what is the best way to do so. Therefore in practice, it can be quite difficult to implement this idea properly. A more systematic solution to this problem has been recently proposed in (Zhang, 2000), where the original Winnow algorithm has been converted into a numerical optimization problem that can handle linearly non-separable problem. Specifically, one seeks a linear weight $\hat{w}$ that solves

$$\min_w \Big[\sum_j w_j \ln \frac{w_j}{e\mu_j} + C \sum_{i=1}^n \max(1 - w^T x^i y^i, 0)\Big].$$

Where $C > 0$ is a given parameter called the regularization parameter. The optimal solution $\hat{w}$ of the above optimization problem can be derived from the solution $\hat{\alpha}$ of the following dual optimization problem:

$$\hat{\alpha} = \max_\alpha \sum_i \alpha^i - \sum_j \mu_j \exp(\sum_i \alpha^i x^i_j y^i)$$
$$\text{s.t.} \quad \alpha^i \in [0, C] \quad (i = 1, \dots, n).$$

The j-th component of $\hat{w}$ is given by

$$\hat{w}_j = \mu_j \exp(\sum_{i=1}^n \hat{\alpha}^i x^i_j y^i).$$

A Winnow-like update rule can be derived for the dual regularized Winnow formulation. At each data point (x^i, y^i), we fix all α_k with $k \neq i$, and update α_i to approximately maximize the dual objective functional using gradient ascent:

$$\alpha^i \to \max(\min(C, \alpha^i + \eta(1 - w^T x^i y^i)), 0), \qquad (1)$$

where $w_j = \mu_j \exp(\sum_i \alpha^i x^i_j y^i)$. We update α and w by repeatedly going over the data from $i = 1, \dots, n$.

Learning bounds of regularized Winnow that are similar to the mistake bound of the original Winnow have been given in (Zhang, 2000). These results imply that the new method, while it can properly handle non-separable data, shares similar theoretical advantages of Winnow in that it is also robust to irrelevant features. This theoretical insight implies that the algorithm is suitable for NLP tasks with large feature spaces.

3. CoNLL-2000 Chunking Task

The text chunking task is to divide text into syntactically related non-overlapping groups of words (chunks). It is considered an important problem in natural language processing.

The CoNLL-2000 shared task (Sang & Buchholz, 2000), introduced last year, is an attempt to set up a standard dataset so that researchers can compare different statistical chunking methods. The data are extracted from sections of the Penn Treebank. The training set consists of WSJ sections 15-18 of the Penn Treebank, and the test set consists of WSJ sections 20. Additionally, a part-of-speech (POS) tag was assigned to each token by a standard POS tagger (Brill, 1994) that was trained on the Penn Treebank.

The data contains eleven different chunk types. However, except for the most frequent three types: NP (noun phrase), VP (verb phrase), and PP (prepositional phrase), each of the remaining chunks has less than 5% occurrences. The chunks are represented by the following three types of tags:

B-X first word of a chunk of type X

I-X non-initial word in an X chunk

O word outside of any chunk

A standard software program has been provided (which is available from *http://lcg-www.uia.ac.be/conll2000/chunking*) to compute the performance of each algorithm. For each chunk, three figures of merit are computed: precision (the percentage of detected phrases that are correct), recall (the percentage of phrases in the data that are found), and the $F_{\beta=1}$ metric which is the harmonic mean of the precision and the recall. The overall precision, recall and $F_{\beta=1}$ metric on all chunks are also computed. The overall $F_{\beta=1}$ metric gives a single number that can be used to compare different algorithms.

4. System Description

4.1 Encoding of Basic Features

An advantage of regularized Winnow is its robustness to irrelevant features. We can thus include as many features as possible, and let the algorithm itself find the relevant ones. This strategy ensures that we do not miss any features that are important. However, using more features requires more memory and slows down the algorithm. Therefore in practice it is still necessary to limit the number of features used.

Let $tok_{-c}, tok_{-c+1}, \ldots, tok_0, \ldots, tok_{c-1}, tok_c$ be a string of tokenized text (each token is a word or punctuation). We want to predict the chunk type of the current token tok_0. For each word tok_i, we let pos_i denote the associated POS tag, which is assumed to be given in the CoNLL-2000 shared task. The following is a list of the features we use as input to the regularized Winnow (where we choose $c = 2$):

- first order features: tok_i and pos_i $(i = -c, \ldots, c)$
- second order features: $pos_i \times pos_j$ $(i, j = -c, \ldots, c,\ i < j)$, and $pos_i \times tok_j$ $(i = -c, \ldots, c;\ j = -1, 0, 1)$

In addition, since in a sequential process, the predicted chunk tags t_i for tok_i are available for $i < 0$, we include the following extra chunk type features:

- first order chunk-type features: t_i $(i = -c, \ldots, -1)$
- second order chunk-type features: $t_i \times t_j$ $(i, j = -c, \ldots, -1,\ i < j)$, and POS-chunk interactions $t_i \times pos_j$ $(i = -c, \ldots, -1; j = -c, \ldots, c)$.

The features are encoded as a binary vector x, which is the input to the Winnow. Each component of x corresponds to a possible feature value. For example, each of the possible POS value of pos_0 corresponds to a component of x: the component has value one if pos_0 is the value represented by the component, and value zero otherwise. The same encoding is applied to all other first order and second order features.

Note that the above representation leads to a sparse, but very large dimensional vector. This explains why we do not include all possible second order features since this will quickly consume more memory than we can handle.

Also the above list of features are not necessarily the best available. We only included the most straightforward features and pair-wise feature interactions. One might try even higher order features to obtain better results.

Since Winnow is robust to irrelevant features, it is usually helpful to provide the algorithm with as many features as possible, and let the algorithm pick up relevant ones. The main problem that prohibits us from using more features in the Winnow algorithm is memory consumption (mainly in training). The time complexity of the Winnow algorithm does not dependent on the number of features, but rather on the average number of non-zero features per data, which is usually quite small.

Due to the memory problem, in our implementation we have to limit the number of token features (words or punctuation) to 5000: we sort the tokens by their frequencies in the training set from high frequency to low frequency; we then treat tokens of rank 5000 or higher as the same token. Since the number 5000 is still reasonably large, this restriction is relatively minor.

There are possible remedies to the memory consumption problem, although we have not implement them in our current system. One solution comes from noticing that although the feature vector is of very high dimension, most dimensions are empty. Therefore one may create a hash table for the features, which can significantly reduce the memory consumption.

4.2 Using Enhanced Linguistic Features

We were interested in determining if additional features with more linguistic content would lead to even better performance. The ESG (English Slot Grammar) system in (McCord, 1989) is not directly comparable to the phrase structure grammar implicit in the WSJ treebank. ESG is a dependency grammar in which each phrase has a head and dependent elements, each marked with a syntactic role. ESG normally produces multiple parses for a sentence, but has the capability, which we used, to output only the highest ranked parse, where rank is determined by a system-defined measure. There are a number of incompatibilities between the treebank and ESG in tokenization, which had to be compensated for in order to transfer the syntactic role features to the tokens in the standard training and test sets. (ESG, for example, discards some punctuation, separates the negative contraction "n't" from the preceding verbal stem, and other things.) We also transferred the ESG part-of-speech codes (different from those in the WSJ corpus) and made an attempt to attach B-PP, B-NP and I-NP tags as inferred from the ESG dependency structure. In the end, the latter two tags did not prove useful. It might seem odd to use a parser output as input to a machine learning system to find syntactic chunks. As noted above, ESG or any other parser normally produces many analyses, whereas in the kind of applications for which chunking is used, e.g., information extraction, only one solution is normally desired. Secondly, ESG is very fast, parsing several thousand sentences on an IBM RS/6000 in a few minutes of clock time.

We denote by f_i the syntactic role tag associated with token tok_i. Each tag takes one of 138 possible values. The following features are added to our system.

- first order features: f_i $(i = -c, \dots, c)$
- second order features: self interactions $f_i \times f_j$ $(i, j = -c, \dots, c, i < j)$, and iterations with POS-tags $f_i \times pos_j$ $(i, j = -c, \dots, c)$.

4.3 Dynamic Programming

Given input vectors x consisting of features constructed as above, we apply the regularized Winnow algorithm to train linear weight vectors. Since the Winnow algorithm only produces positive weights, we employ the balanced version of Winnow with x being transformed into $\tilde{x} = [x, 1, -x, -1]$. As explained earlier, the constant term is used to offset the effect of threshold θ. Once a weight vector $\tilde{w} = [w_+, \theta_+, w_-, \theta_-]$ is obtained, we let $w = w_+ - w_-$ and $\theta = \theta_+ - \theta_-$. The prediction with an incoming feature vector x is then $L(w, x) = w^T x - \theta$.

Since Winnow only solves binary classification problems, we train one linear classifier for each chunk type. In this way, we obtain twenty-three linear classifiers, one for each chunk type t. Denote by w^t the weight associated with type t, then a straight-forward method to classify an incoming datum is to assign the chunk tag as the one with the highest score $L(w^t, x)$.

However, there are constraints in any valid sequence of chunk types: if the current chunk is of type I-X, then the previous chunk type can only be either B-X or I-X. This constraint can be explored to improve chunking performance. We denote by V the set of all valid chunk sequences (that is, the sequence satisfies the above chunk type constraint).

Let $tok_1, \dots, tok_m$ be the sequence of tokenized text for which we would like to find the associated chunk types. Let $x_1, \dots, x_m$ be the associated feature vectors for this text sequence. Let $t_1, \dots, t_m$ be a sequence of potential chunk types that is valid: $\{t_1, \dots, t_m\} \in V$. In our system, we find the sequence of chunk types that has the highest value of overall truncated score as:

$$\{\hat{t}_1, \dots, \hat{t}_m\} = \arg \max_{\{t_1, \dots t_m\} \in V} \sum_{i=1}^{m} L'(w^{t_i}, x_i),$$

where

$$L'(w^{t_i}, x_i) = \min(1, \max(-1, L(w^{t_i}, x_i))).$$

The truncation onto the interval $[-1, 1]$ is to make sure that no single point contributes too much in the summation.

The optimization problem

$$\max_{\{t_1, \dots t_m\} \in V} \sum_{i=1}^{m} L'(w^{t_i}, x_i)$$

can be solved by using dynamic programming. We build a table of all chunk types for every token tok_i. For each fixed chunk type t_{k+1}, we define a value

$$S(t_{k+1}) = \max_{\{t_1, \dots t_k, t_{k+1}\} \in V} \sum_{i=1}^{k+1} L'(w^{t_i}, x_i).$$

It is easy to verify that we have the following recursion:

$$S(t_{k+1}) = L'(w^{t_{k+1}}, x_{k+1}) + \max_{\{t_k, t_{k+1}\} \in V} S(t_k). \quad (2)$$

We also assume the initial condition $S(t_0) = 0$ for all t_0. Using this recursion, we can iterate over $k = 0, 1, \dots, m$, and compute $S(t_{k+1})$ for each potential chunk type t_{k+1}.

Observe that in (2), x_{k+1} depends on the previous chunk-types $\hat{t}_k, \dots, \hat{t}_{k+1-c}$ (where $c = 2$). In our implementation, these chunk-types used to create the current feature vector x_{k+1} are determined as follows. We let $\hat{t}_k = \arg\max_{t_k} S(t_k)$, and let $\hat{t}_{k-i} = \arg\max_{t_{k-i}:\{t_{k-i}, \hat{t}_{k-i+1}\} \in V} S(t_{k-i})$ for $i = 1, \dots c$.

After the computation of all $S(t_k)$ for $k = 0, 1, \ldots, m$, we determine the best sequence $\{\hat{t}_1, \ldots, \hat{t}_m\}$ as follows. We assign $\hat{t}_m$ to the chunk type with the largest value of $S(t_m)$. Each chunk type $\hat{t}_{m-1}, \ldots, \hat{t}_1$ is then determined from the recursion (2) as $\hat{t}_k = \arg\max_{t_k:\{t_k,\hat{t}_{k+1}\}\in V} S(t_k)$.

5. Experimental Results

Experimental results reported in this section were obtained by using $C = 1$, and a uniform prior of $\mu_i = 0.1$. We let the learning rate $\eta = 0.01$, and ran the regularized Winnow update formula (1) repeatedly thirty times over the training data.

Table 1 gives results obtained with the basic features. This representation gives a total number of 3.8×10^6 binary features. However, the number of non-zero features per datum is 48, which determines the time complexity of our system. The training time on a 400Mhz Pentium machine running Linux is about sixteen minutes, which corresponds to less than one minute per category. The time using the dynamic programming to produce chunk predictions, excluding tokenization, is less than ten seconds. There are about 7×10^4 non-zero linear weight components per chunk-type, which corresponds to a sparsity of more than 98%. Most features are thus irrelevant.

All previous systems achieving a similar performance are significantly more complex. For example, the previous best result in the literature was achieved by a combination of 231 kernel support vector machines (Kudoh & Matsumoto, 2000) with an overall $F_{\beta=1}$ value of 93.48. Each kernel support vector machine is computationally significantly more expensive than a corresponding Winnow classifier, and they use an order of magnitude more classifiers. This implies that their system should be orders of magnitudes more expensive than ours. This point can be verified from their training time of about one day on a 500Mhz Linux machine. The previously second best system was a combination of five different WPDV models, with an overall $F_{\beta=1}$ value of 93.32 (van Halteren, 2000). This system is again much more complex than the regularized Winnow approach we propose (their best single classifier performance is $F_{\beta=1} = 92.47$). The rest of the eleven reported systems achieved performance of no better than $F_{\beta=1} = 92.50$. Still some of the systems are combinations of memory-based models that are computationally much more expensive than our approach.

The above comparison implies that the regularized Winnow approach achieves state of the art performance with significant less computation than previous systems. The success of this method relies on regularized Winnow's ability to tolerate irrelevant features. This allows us to use a very large feature space and let the algorithm to pick the relevant ones. In addition, the algorithm presented in this paper is simple. Unlike some other approaches, there is nearly no ad hoc engineering tuning involved in our system. This simplicity allows other researchers to reproduce our results easily.

In Table 2, we report the results of our system with the basic features enhanced by using ESG syntactic roles, showing that using more linguistic features can enhance the performance of the system. In addition, since regularized Winnow is able to pick up relevant features automatically, we can easily integrate different features into our system in a systematic way without concerning ourselves with the semantics of the features. The resulting overall $F_{\beta=1}$ value of 94.13 is significantly better than any previous system. The overall complexity of the system is still quite reasonable. The total number of features is about 4.2×10^6, with 88 nonzero features for each data point. The training time is about thirty minutes, and the number of non-zero weight components per chunk-type is about 8×10^4.

Table 1. Our chunk prediction results: with basic features

testdata	precision	recall	$F_{\beta=1}$
ADJP	79.45	72.37	75.75
ADVP	81.46	80.14	80.79
CONJP	45.45	55.56	50.00
INTJ	100.00	50.00	66.67
LST	0.00	0.00	0.00
NP	93.86	93.95	93.90
PP	96.87	97.76	97.31
PRT	80.85	71.70	76.00
SBAR	87.10	87.10	87.10
VP	93.69	93.75	93.72
all	93.53	93.49	93.51

6. Conclusion

In this paper, we described a text chunking system using regularized Winnow. Since regularized Winnow is robust to irrelevant features, we can construct a very high dimensional feature space and let the algorithm pick up the important ones. We have shown that state of the art performance can be achieved by using this approach. Furthermore, the method we propose is computationally significantly more efficient than all other systems reported in the literature that achieved

Table 2. Our chunk prediction results: with enhanced features

testdata	precision	recall	$F_{\beta=1}$
ADJP	82.22	72.83	77.24
ADVP	81.06	81.06	81.06
CONJP	50.00	44.44	47.06
INTJ	100.00	50.00	66.67
LST	0.00	0.00	0.00
NP	94.45	94.36	94.40
PP	97.64	98.07	97.85
PRT	80.41	73.58	76.85
SBAR	91.17	88.79	89.96
VP	94.31	94.59	94.45
all	94.24	94.01	94.13

performance close to ours. Our system is also very simple and involves little engineering tuning. This means that it is relatively easy for other researchers to implement and reproduce our results. Furthermore, the success of regularized Winnow in text chunking suggests that the method might be applicable to other NLP problems where it is necessary to use large feature spaces to achieve good performance.

References

Brill, E. (1994). Some advances in rule-based part of speech tagging. *Proc. AAAI 94* (pp. 722–727).

Dagan, I., Karov, Y., & Roth, D. (1997). Mistake-driven learning in text categorization. *Proceedings of the Second Conference on Empirical Methods in NLP.*

Khardon, R., Roth, D., & Valiant, L. (1999). Relational learning for NLP using linear threshold elements. *Proceedings IJCAI-99.*

Kudoh, T., & Matsumoto, Y. (2000). Use of support vector learning for chunk identification. *Proc. CoNLL-2000 and LLL-2000* (pp. 142–144).

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning, 2,* 285–318.

McCord, M. (1989). Slot grammar: a system for simple construction of practical natural language grammars. *Natural Language and Logic,* 118–145.

Sang, E. F. T. K., & Buchholz, S. (2000). Introduction to the conll-2000 shared tasks: Chunking. *Proc. CoNLL-2000 and LLL-2000* (pp. 127–132).

van Halteren, H. (2000). Chunking with wpdv models. *Proc. CoNLL-2000 and LLL-2000* (pp. 154–156).

Zhang, T. (2000). Regularized winnow methods. *NIPS'00.*