Journal of Machine Learning Research 2 (2002) 615-637 Submitted 9/01; Published 3/02

Text Chunking based on a Generalization of Winnow

Tong Zhang TZHANG@WATSON.IBM.COM
Fred Damerau DAMERAU@WATSON.IBM.COM
David Johnson DEJOHNS@US.IBM.COM
T.J. Watson Research Center
Route 134, Yorktown Heights, NY 10598, U.S.A.

Editors: James Hammerton, Miles Osborne, Susan Armstrong and Walter Daelemans

Abstract

This paper describes a text chunking system based on a generalization of the Winnow algorithm. We propose a general statistical model for text chunking which we then convert into a classification problem. We argue that the Winnow family of algorithms is particularly suitable for solving classification problems arising from NLP applications, due to their robustness to irrelevant features. However in theory, Winnow may not converge for linearly non-separable data. To remedy this problem, we employ a generalization of the original Winnow method. An additional advantage of the new algorithm is that it provides reliable confidence estimates for its classification predictions. This property is required in our statistical modeling approach. We show that our system achieves state of the art performance in text chunking with less computational cost then previous systems.

1. Introduction

In the field of natural language processing, one important research area is the extraction and formatting of information from unstructured text. One can think of the end goal of information extraction in terms of filling templates codifying the extracted information. The field of text-based information extraction has a long history in the artificial intelligence and computational linguistics areas (Muslea, 1999).

Although the most accurate systems often involve language processing modules that are hand-built, substantial progress has been made in applying supervised machine learning techniques to a number of processes necessary for extracting information from text. This task naturally decomposes into a sequence of processing steps, typically including: tokenization, sentence segmentation, part-of-speech assignment, named-entity identification, phrasal parsing, sentential parsing, semantic interpretation, discourse interpretation, template filling and merging.

Typically, information extraction systems only partially parse the text, i.e., chunk the text into typed phrases only to the extent required to instantiate the attributes of a template. The key idea here is the assumption that for a wide range of constrained applications, the predefined attributes of templates and the semantic types of the attribute-values permit the system to make a reasonably accurate determination of the roles of the various typed chunks identified by the linguistic processors (named-entity identifiers, noun phrase chunkers, verbal group chunkers, etc.) without attempting a complete sentence parse. The systems are typically organized as a cascade of linguistic processors.

The application of machine learning techniques to information extraction is motivated by the fact that hand-built systems are time consuming to develop and require special expertise in linguistics and artificial intelligence/computational linguistics. This, coupled with the fact that domain-specific information is required to build an accurate system, motivates research on trainable information extraction systems.

An important component of information extraction as well as many other natural language processing system is the development of accurate text chunkers, which when cascaded, would carry out the partial parsing required for template filling. In order to facilitate measurable progress in the area of partial parsing, the natural language processing community has defined a set of common text chunking tasks, the CoNLL, described in detail in Section 2.

In recent years, many machine learning techniques have been applied to NLP problems. One method that has been quite successful in such applications is the SNoW architecture (Dagan et al., 1997, Khardon et al., 1999). This architecture is based on the Winnow algorithm (Littlestone, 1988, Grove and Roth, 2001), 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 makes it attractive for NLP tasks.

However, the convergence of the Winnow algorithm is only guaranteed for linearly separable data. In practical NLP applications, data may not always be linearly separable. Consequently, a direct application of Winnow can lead to numerical instability. A second problem for the original Winnow method is that it does not produce a reliable estimate of the confidence of its predictions. Such information can be very useful in statistical modeling. In fact, some kind of confidence estimate is required in our approach to text chunking.

Due to the above mentioned problems, we use a modification of Winnow which can handle the linearly non-separable case, and produce a reliable probability estimate. A version of the new method was originally proposed by Zhang (2001), and was used in an earlier version of this work (Zhang et al., 2001). The basic idea is to modify the original Winnow algorithm so that it solves a regularized optimization problem. The resulting method 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, which may not be linearly separable.

In this paper, we apply generalized Winnow to text chunking tasks (Abney, 1991). In order for us to rigorously compare our system with others, we use the CoNLL-2000 shared task dataset (Sang and Buchholz, 2000), which is publicly available from *http://lcg-www.uia.ac.be/conll2000/chunking*. A large number of state of the art statistical natural language processing methods have already been applied to the data, allowing us to 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 less computation than earlier systems of comparable performance.

The paper is organized as follows. Section 2 describes the text chunking problem and the CoNLL-2000 shared task. In Section 3, we present a general statistical model for text chunking as a sequential prediction problem. We then convert it into a classification problem. In Section 4, we describe the Winnow algorithm and the regularized Winnow method. Section 5 gives a detailed description of our system that employs the regularized Winnow algorithm for text chunking. Section 6 contains experimental results for our system on the CoNLL-2000 shared task. Some final remarks will be given in Section 7.

2. Text Chunking and the CoNLL-2000 Chunking

The text chunking problem (Abney, 1991) is to divide text into syntactically related non-overlapping groups of words (chunks). As an example, the sentence "*Balcor, which has interests in real estate, said the position is newly created.*" can be divided as follows:

[NP Balcor], [NP which] [VP has] [NP interests] [PP in] [NP real estate], [VP said] [NP the position] [VP is newly created].

In this example, NP denotes non phrase, VP denotes verb phrase, and PP denotes prepositional phrase.

Text chunking has important applications in natural language processing. In addition, many shallow parsing and information extraction tasks can be formulated as grouping unstructured text into syntactically or semantically related chunks. This means a good method for text chunking can be potentially very useful for other similar NLP tasks as well.

In the past decade, there has been significant interest in machine learning approaches to the text chunking problem. To facilitate the comparison among different methods, the CoNLL-2000 shared task was introduced in 2000 (Sang and Buchholz, 2000). It 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 (211727 tokens), and the test set consists of WSJ section 20 (47377 tokens).

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. These POS tags can be used as features in a machine learning based chunking algorithm. See Section 5 for detail. As an example, for the previous example sentence, the associated POS tags, given in the parenthesis following each token, are:

Balcor (NNP) , (,) which (WDT) has (VBZ) interests (NNS) in (IN) real (JJ) estate (NN) , (,) said (VBD) the (DT) position (NN) is (VBZ) newly (RB) created (VBN) . (.)

The CoNLL-2000 data contains eleven different chunk types. However, except for the most frequent three types: NP, VP, and PP, each of the remaining chunks accounts for less than 5% of the occurrences. The eleven chunk types are: ADJP, ADVP, CONJP, INTJ, LST, NP, PP, PRT, SBAR, VP, and UCP.[1] They are represented by the following three types of tags associated with each word or punctuation in a sentence:

B-X: first word of a chunk of type X.

I-X: non-initial word in a chunk of type X.

O: word outside of any chunk.

1. The UCP chunk-type does not appear in the test set. It will not be counted by the evaluation program if the chunker to be evaluated does not produce a UCP chunk type either.

A standard software program, available from *http://lcg-www.uia.ac.be/conll2000/chunking*, is provided to compute the performance of an 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.

3. Text Chunking as a Sequential Prediction Problem

One can view text chunking as a sequential prediction problem, where we predict the chunk-tag associated with every token in a sequence of tokens. In our case, each token is either a word or a punctuation in the text.

We denote by $\{w_i\}$ $(i = 0, 1, \ldots, m)$ the sequence of tokenized text. The goal is to find the chunk-tag t_i associated with each w_i. Since we assume that w_i is known, in a probability model, we are interested in the conditional probability of the sequence of chunk-tag $\{t_i\}$ given $\{w_i\}$. To simplify the problem, we consider the following decomposition of the conditional probability:

$$P(\{t_i\}|\{w_i\}) = \prod_{\ell=0}^{m} P(t_\ell|\{t_j\}_{j<\ell}, \{w_i\}) = \prod_{\ell=0}^{m} P(t_\ell|x_\ell(\{t_j\}_{j<\ell}, \{w_i\})), \tag{1}$$

where each x_ℓ is called the "feature vector" at ℓ which gives a sufficient statistics of the conditional probability of t_ℓ given $\{t_j\}_{j<\ell}$ and $\{w_i\}$. At this stage, the detailed representation of the feature vector x_ℓ is not important. In practice, one may consider a Markov model so that each x_ℓ is determined by text in a neighborhood of w_ℓ (it may also contain additional information such as the part-of-speech). Our choices of feature representation for text chunking are given in Section 5.

Given a probability model of the form (1), the goal is to estimate the chunk-tag sequence $\{\hat{t}_i\}$ to minimize the error rate. One method that is often used in practical applications is to find the sequence $\{\hat{t}_i\}$ with the highest probability. However, this corresponds to the maximization of log probability $\sum_\ell \ln P(t_\ell|x_\ell)$ rather than minimizing error. In principle, the expected number of correct predictions in a sequence $\{\hat{t}_i\}$ of chunk estimates is given by

$$\sum_i \sum_{\{t_j\}_{j\neq i}} \prod_{\ell=0}^{m} P(t_\ell|x_\ell(\{t_j\}_{j<\ell}, \{w_i\})),$$

where $\sum_{\{t_j\}_{j\neq i}}$ denotes the sum over all sequences $\{t_j\}$ with the i-th token t_i left out. In the above, we fix $t_i = \hat{t}_i$. From this expression, the estimator $\hat{t}_i$ that minimizes the expected error rate is given by:

$$\hat{t}_i = \arg\max_{t_i} \sum_{\{t_j\}_{j\neq i}} \prod_{\ell=0}^{m} P(t_\ell|x_\ell(\{t_j\}_{j<\ell}, \{w_i\})). \tag{2}$$

The right hand side of the above expression is the marginal conditional density: $P(t_i|\{w_\ell\})$. Although minimizing word level error rate does not correspond exactly to minimizing chunk level error rate, the latter is significantly more difficult to model. Also it is reasonable to believe that a small word level error rate implies a small chunk level error rate. If the dependency of t_ℓ on $\{t_j\}_{j<\ell}$ is Markov, then in principle the right hand side of (2) can be computed using dynamic programming. However, the corresponding computational cost is still in the order of the number of all possible dependencies of t_ℓ on $\{t_j\}_{j<\ell}$, summed over ℓ. For example, if each t_ℓ only depends on $t_{\ell-1}, \cdots, t_{\ell-c}$, and assume that there are T possible chunk type values, then the number of evaluations of $P(t_\ell | x_\ell(\{t_j\}_{j<\ell}, \{w_i\}))$ is in the order of $T^{c+1}m$, which can be a fairly large number. From the computational point of view, we would like to reduce this number to Tm. It is thus necessary to use an approximation. In this paper, we employ an idea that is related to the mean-field approximation method widely used in statistical physics and graphical model inference: we replace $x_\ell(\{t_j\}_{j<\ell}, \{w_i\})$ by $x_\ell(\{\hat{t}_j\}_{j<\ell}, \{w_i\})$. Now the right hand side of (2) becomes $P(t_i | x_\ell(\{\hat{t}_j\}_{j<i}, \{w_i\}))$. Summing over i, we obtain the following approximate solution of (2):

$$\{\hat{t}_i\} = \arg\max_{\{t_i\}} \sum_{i=0}^{m} P(t_i | x_i), \tag{3}$$

where x_i is the short-hand notation of $x_i(\{t_j\}_{j<i}, \{w_\ell\})$. The right hand side is an approximation of the expected number of correct predictions. Further details concerning how we apply the approximation (3) to text chunking is given in Section 5. It can be observed now that the remaining task is to estimate the conditional probability $P(t_i | x_i)$ for each possible chunk-tag t_i. This can be regarded as a binary classification problem (for each possible chunk-tag): given a feature vector x, what is the probability that the associated chunk-tag is t_i.

As we will see in Section 5, we use a large number of features for the text chunking problem. Consequently, we need a classification algorithm that is robust to large feature size. In this regard, the Winnow family algorithm is very suitable. Furthermore we require our algorithm to provide probability information and to be robust to noise in the data. It is thus necessary for us to modify the Winnow algorithm, which we describe next.

4. Winnow and Generalized Winnow for Binary Classification

We review the Winnow algorithm and derive a modification that can handle linearly non separable data and provide confidence information.

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 components of the input vector. Specifically, we seek a weight vector w and a threshold θ such that $w^T x < \theta$ if its label $y = -1$ and $w^T x \geq \theta$ 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 set of labeled data $(x^1, y^1), \ldots, (x^n, y^n)$, 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:

$$w_j \leftarrow w_j \exp(\eta x^i_j y^i), \tag{4}$$

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.

In order to obtain a systematic solution to this problem, we shall first examine a derivation of the Winnow algorithm by Gentile and Warmuth (1998), which motivates a more general solution to be presented later.

Following Gentile and Warmuth (1998), we consider a loss function $\max(-w^T x^i y^i, 0)$, often called "hinge loss". Using the general on-line learning framework of Kivinen and Warmuth (1997), for each data point (x^i, y^i), we consider an online update rule such that the weight w^{i+1} after seeing the i-th example is given by the solution to

$$\min_{w^{i+1}} \left[\sum_j w^{i+1}_j \ln \frac{w^{i+1}_j}{e w^i_j} + \eta \max\left(-w^{(i+1)T} x^i y^i, 0\right) \right]. \tag{5}$$

Setting the gradient of the above formula to zero, we obtain

$$\ln \frac{w^{i+1}}{w^i} + \eta \nabla_{w^{i+1}} = 0. \tag{6}$$

In the above equation, $\nabla_{w^{i+1}}$ denotes the gradient (or more rigorously, a subgradient) of $\max(-w^{(i+1)T} x^i y^i, 0)$, which takes the value 0 if $w^{(i+1)T} x^i y^i > 0$, the value $-x^i y^i$ if $w^{(i+1)T} x^i y^i < 0$, and a value in between if $w^{(i+1)T} x^i y^i = 0$.

We can approximately solve (6) with $\nabla_{w^{i+1}}$ replaced by ∇_{w^i}. This leads to

$$w^{i+1} = \begin{cases} w^i & w^{iT} x^i y^i > 0, \\ w^i \exp(s\eta x^i y^i) & w^{iT} x^i y^i \le 0, \end{cases}$$

where $s = 1$ if $w^{iT} x^i y^i < 0$ and $s \in [0, 1]$ if $w^{iT} x^i y^i = 0$. In practice, we can simply set $s = 1$, which leads to the winnow update formula (4).

Although the above derivation does not solve the non-convergence problem of the original Winnow method when the data are not linearly separable, it does provide valuable insights which can lead to a more systematic solution of the problem. The basic idea was given by Zhang (2001, 2002), where the original Winnow algorithm was converted into a numerical optimization problem that can handle linearly non-separable data.

The resulting formulation is closely related to (5). However, instead of looking at one example at a time as in an online formulation, we incorporate all examples at the same time. In this paper, we also use the following formulation that replaced the "hinge loss" in (5) by a function $f(v)$ given below (the reason for choosing this loss function will become clear later). Specifically, we seek a linear weight $\hat{w}$ that solves

$$\hat{w} = \arg\min_{w} \left[\sum_j w_j \ln \frac{w_j}{e\mu_j} + c \sum_{i=1}^{n} f(w^T x^i y^i) \right], \tag{7}$$

where

$$f(v) = \begin{cases} -2v & v < -1 \\ \frac{1}{2}(v-1)^2 & v \in [-1, 1] \\ 0 & v > 1. \end{cases}$$

and $c > 0$ 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 (see Appendix A for derivation):

$$\begin{aligned} \hat{\alpha} = \max_{\alpha} & \sum_i \left[\alpha^i - \frac{1}{2c}(\alpha^i)^2 \right] - \sum_j \mu_j \exp\left(\sum_i \alpha^i x^i_j y^i \right) \\ \text{s.t.} \quad & \alpha^i \in [0, 2c] \quad (i = 1, \ldots, n). \end{aligned} \tag{8}$$

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

$$\hat{w}_j = \mu_j \exp\left(\sum_{i=1}^{n} \hat{\alpha}^i x^i_j y^i \right). \tag{9}$$

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:

$$\alpha^i \to \max\left(\min\left(2c, \alpha^i + \eta \left(1 - \frac{\alpha^i}{c} - w^T x^i y^i \right) \right), 0 \right), \tag{10}$$

where $w_j = \mu_j \exp\left(\sum_i \alpha^i x^i_j y^i \right)$.

The dual variable α is initialized to be zero, which corresponds to $w = \mu$. We then update α and w by repeatedly going over the data sequentially from $i = 1, \ldots, n$. This is very similar to an online algorithm such as the original Winnow method. However, we memorize each α^i and when we revisit the same point in later iterations, the memorized α^i plays a role in the update formula.

The specific choice of loss function $f(v)$ is analyzed in Appendix B, where we show that if we approximately minimize the expected loss, then we obtain a weight such that $(T(\hat{w}^T x) + 1)/2$ is close to the conditional probability $P(y = 1|x)$. We use $T(p)$ to denote the truncation of p into $[-1, 1]$. Note that the general formulation outlined in Section 3 requires the classifier to provide a conditional probability estimate. Therefore results in Appendix B imply that regularized Winnow is suitable for our statistical approach to the text chunking problem.

Moreover regularized Winnow has attractive generalization properties. It was shown by Littlestone (1988) that the original Winnow method was robust to irrelevant features in that the number of mistakes it makes to obtain a classifier (in the separable case) depends only logarithmically on the dimensionality of the feature space. Generalization bounds of regularized Winnow that are similar to the mistake bound of the original Winnow (in the sense of logarithmic dependent on the dimensionality) have been given by Zhang (2001). 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.

5. System Description

This section describes our text chunking system using the generalized Winnow algorithm and the statistical modeling approach in Section 3. We focus on feature representation and some computational issues.

5.1 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.

As an example, consider the following sentence used in Section 2:

Balcor (NNP) , (,) which (WDT) has (VBZ) interests (NNS) in (IN) real (JJ) estate (NN) , (,) said (VBD) the (DT) position (NN) is (VBZ) newly (RB) created (VBN) . (.)

Assume that the current token we are looking at is tok_0 = "has". Then tok_{-1} = "which", pos_0 = "VBZ", pos_{-1} = "WDT", and so on.

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

For each data point (corresponding to the current token tok_0), the associated features are encoded as a binary vector x, which is the input to Winnow. Each component of x corresponds to a possible feature value v of a feature f in one of the above feature lists. The value of the component corresponds to a test which has value one if the corresponding feature f achieves value v, or value zero if the corresponding feature f achieves another feature value.

For example, since pos_0 is in our feature list, each of the possible POS values v of pos_0 corresponds to a component of x: the component has value one if $pos_0 = v$ (the feature value represented by the component is active), and value zero otherwise. Similarly for a second order feature in our feature list such as $pos_0 \times pos_1$, each possible value $v_0 \times v_1$ in the set $\{pos_0 \times pos_1\}$ is represented by a component of x: the component has value one if $pos_0 = v_0$ and $pos_1 = v_1$ (the feature value represented by the component is active), and value zero otherwise. The same encoding is applied to all other first order and second order features, with each possible test of "feature = feature value" corresponds to a unique component in x.

Clearly, in this representation, the high order features are conjunction features that become active when all of their components are active. In principle, one might also consider disjunction features that become active when some of their components are active. However, such features are not considered in this work.

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 consumes too much memory.

Also the above list of features are not necessarily the best available. We only included the most straight-forward 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. The main drawback of using more features is memory consumption (which mainly affects training). The time complexity of the Winnow algorithm does not depend on the number of features, but rather on the average number of non-zero features per data point, which is usually quite small.

One way to alleviate the memory problem is to use a hash table. This approach works because despite the high dimensionality of the feature space, most of these dimensions are empty. In an earlier version of the current work (Zhang et al., 2001), in order to save memory without using a hash table, we limited the number of token feature values (words or punctuation) to 5000 by removing less frequent tokens. However, since then we have implemented a hash table. We now use all token features for results reported in this paper. This not only slightly improves the accuracy, but also accelerates the computation due to better memory locality. Consequently, timing reported in this paper is better than that of

Zhang et al. (2001). For consistency, we use the same set of features as those used by Zhang et al. (2001). The slight improvement we obtained in this paper is due to the fact that we do not limit the number of tokens to 5000.

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

We use the same encoding scheme as the that of the basic features described earlier. Each possible value of the additional features is represented by a component in the data vector x which is the input to Winnow. The component has value one if the feature value represented by the component is active itself, and value zero otherwise. It is clear that this particular encoding scheme is quite versatile. We can include additional features and feature interactions without paying attention to the semantics and compatibility of different features.

5.3 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,[2] we let $w = (w_+ - w_-, \theta_+ - \theta_-)$. The prediction with an incoming feature vector x is then $L(w, x) = (w_+ - w_-)^T x - (\theta_+ - \theta_+)$.

For each chunk type t, we use the regularized Winnow to compute a weight w^t. From Appendix B, we know that $L(w^t, x)$ can be regarded as an estimate of $P(t_i|x_i)$ in (3): $P(t_i|x_i) \approx (1+T(L(w^T, x_i)))/2$. Equation (3) can be solved by using dynamic programming.

The use of dynamic programming for sequential prediction problems is not new. A specific advantage of the dynamic programming approach is that constraints required in a valid prediction sequence can be handled in a principled way (for example, see Punyakanok and Roth, 2001). In the case of text chunking, we would like to produce a sequence of chunk tags consistent with respect to the B-I-O encoding scheme (otherwise, some heuristic post-processing has to be done). That is, any valid sequence of chunk tags has to satisfy the following constraint: if the current chunk tag is I-X, then the previous chunk tag can only be either B-X or I-X.

We denote by V the set of all valid chunk sequences (that is, the sequence satisfies the above chunk type constraint). Let $tok_1, \ldots, tok_m$ be the sequence of tokenized text for which we would like to find the associated chunk types. Let $x_1, \ldots, x_m$ be the associated feature vectors for this text sequence. Let $t_1, \ldots, t_m$ be a sequence of potential chunk types that is valid: $\{t_1, \ldots, t_m\} \in V$. In our system, we find a valid sequence of chunk types that maximizes (3) as:

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

where

$$L'(w^{t_i}, x_i) = T(L(w^{t_i}, x_i)) = \min(1, \max(-1, L(w^{t_i}, x_i))).$$

Note that for generalized Winnow, $L'(w^{t_i}, x_i)$ corresponds to an estimate of the conditional probability $P(t_i|x_i)$. The statistical modeling approach described in Section 3 can be directly applied. For the original Winnow, $L'(w^{t_i}, x_i)$ itself is not a conditional probability estimate. Therefore this approach can only be justified heuristically. In this case $L'(w^{t_i}, x_i)$ can be regarded as a confidence score of chunk type t_i given x_i, which is correlated to $P(t_i|x_i)$. Our formulation can then be interpreted as to find the sequence of chunk types that has the highest value of overall score.

The optimization problem

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

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

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

2. In our implementation, we explicitly keep both w_+ and w_-. However if memory consumption is a concern, it is also possible to store w_+ only since each component of w_- is inversely proportional to the corresponding component in w_+.

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

$$S(t_{k+1}) = L'(w^{t_{k+1}}, x_{k+1}) + \max_{\{t_k, t_{k+1}\} \in V} S(t_k). \tag{11}$$

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

Observe that in (11), x_{k+1} depends on the previous chunk-types $\hat{t}_k, \ldots, \hat{t}_{k+1-c}$ (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

$$\hat{t}_{k-i} = \arg \max_{t_{k-i}:\{t_{k-i}, \hat{t}_{k-i+1}\} \in V} S(t_{k-i}) \qquad (i = 1, \ldots, c).$$

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 (11) as $\hat{t}_k = \arg\max_{t_k:\{t_k, \hat{t}_{k+1}\} \in V} S(t_k)$.

6. Experimental Results

In this section, we study the following issues empirically: the influence of parameters in the regularized Winnow algorithm using five-fold cross-validation within the training set, the testset results with default parameters and comparison with other methods, and the effect of different feature space sizes.

We should mention that since the UCP chunk type does not appear in the standard test set, some other systems do not learn this chunk. Although such an approach might not be truly regarded as cheating, one may consider going even further by discarding a few other chunks of type I-X which do not appear in the test set either. To avoid this problem, in our opinion, it is best to learn and test with all chunk types without using the knowledge that UCP chunk does not appear in the test set. Therefore results reported in this paper were obtained by learning all eleven chunk types leading to a twenty-three class classification problem. Our timing will be about 10% better if we do not learn the UCP chunk type.

6.1 The Influence of Parameters in Generalized Winnow

We choose default parameters as $\lambda = 0.1$, and a uniform prior of $\mu_j = 0.1$. We let the learning rate $\eta = 0.01$, and ran the regularized Winnow update formula (10) repeatedly $K = 30$-times over the training data. In our experience, these default parameters are suitable for general problems including some other natural language processing problems for which we have applied the regularized Winnow algorithm. Experiments in this section are performed with the basic features.

The goal of this section is to study the effect of varying any one of these parameters. The experiments are done using a five-fold cross-validation within the standard CoNLL-2000 training set — each time we use a 1/5 continuous block of the training set as the validation data, and the rest of the training set to estimate a chunker.

We first consider varying the regularization parameter λ from 10^{-3} to 10. Table 1 contains the overall precision, recall and $F_{\beta=1}$ measures for these different λ values over the five fold cross-validation. It can be seen that for text chunking, we obtain good results as long as we make λ relative small. This is closely related to the fact that each binary classification problem is nearly separable.[3] The regularization parameter λ plays the role of balancing the bias-variance trade-off typically observed in a statistical estimation problem. If λ is small, the bias is small since results in Appendix B imply that we directly estimate the conditional in-class probability function. However, if λ is small, then the variance is relatively large because the algorithm itself is less stable. It is then natural to choose a regularization parameter that gives a small bias and with good stability properties. For the text chunking problem, from Table 1, it is reasonable to pick the default value of λ as 0.1.

λ	10^{-3}	10^{-2}	10^{-1}	1	10
precision	93.57	93.57	93.56	93.49	92.68
recall	93.49	93.49	93.48	93.43	92.84
$F_{\beta=1}$	93.53	93.53	93.52	93.46	92.76

Table 1: The effect of regularization parameter λ

Table 2 investigates the effect of different values of the prior μ_j. Although the table seems to imply that the algorithm is more sensitive to the choice of prior μ than to the choice of λ, an important reason that contributes to this phenomenon is that we fixed the number of iterations. From the numerical point of view, if we fix the number of iterations, a small μ_j has an effect similar to a small learning rate η. As we explain next, a small η can lead to slower convergence and a large η can lead to instability. Undesirable results corresponding to small priors in Table 2 can be partially explained by using the fact that the algorithm has not yet achieved convergence after 30 iterations. From these results, we can also see that the default value of $\mu_j = 0.1$ shows a good balance of quick convergence and stability.

μ_j	10^{-3}	10^{-2}	10^{-1}	1
precision	91.57	92.63	93.56	93.25
recall	91.99	92.77	93.48	93.16
$F_{\beta=1}$	91.78	92.70	93.52	93.20

Table 2: The effect of prior value μ_j

Table 3 shows the effect of using different values of learning rate η. In general, it is safer to use a smaller learning rate although the corresponding convergence rate is slower. Practically it is desirable to use a learning rate that is not too small which still leads a stable algorithm. The bad performance at $\eta = 0.1$ shows that the algorithm already becomes numerically unstable. The performance comparison between learning rates of $\eta = 10^{-2}$ and 10^{-3} after the default 30 iterations indicates that the algorithm does not converge very fast at $\eta = 10^{-3}$ (although with sufficient number of iterations, it converges to the correct

3. For more difficult problems that are not nearly separable, we usually observe the following phenomenon: as λ increases, the classification accuracy first increases to an optimal point, and then decreases.

solution). The learning rate issue has been more thoroughly investigated by Zhang (2002). In fact, results there suggest that the default choice of $\eta = 0.01$ is a value that can be good for general problems.

η	10^{-3}	10^{-2}	10^{-1}
precision	91.90	93.56	13.88
recall	92.19	93.48	2.74
$F_{\beta=1}$	92.04	93.52	4.58

Table 3: The effect of learning rate η

In Table 4, we study the performance of regularized Winnow using various numbers of iterations. The results indicate that the algorithm has approximately achieved convergence after thirty iterations since the performance has stabilized.

K	5	10	20	30	40	50
precision	92.49	93.05	93.42	93.56	93.58	93.60
recall	92.62	93.05	93.35	93.48	93.5	93.52
$F_{\beta=1}$	92.56	93.05	93.39	93.52	93.54	93.56

Table 4: The effect of iteration number K in training

6.2 Test Set Results and Comparison with Other Methods

Experimental results reported in this section were obtained by using the default values of $\lambda = 0.1$, $\mu_j = 0.1$, and $\eta = 0.01$, and $K = 30$. Although using more iterations may improve results a little, as we can see from Table 4, the performance has more or less stabilized after thirty iterations. Results reported in this section are obtained by training on the standard CoNLL-2000 training set and testing on the standard test set.

Table 6 gives results obtained with the basic features. This representation gives a total number of 4.6×10^5 non-zero hash table entries (binary features with counts of at least one). 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 twelve minutes, which corresponds to about thirty seconds per category. The time using the dynamic programming to produce chunk predictions, excluding tokenization, is less than ten seconds. There are about 8.6×10^4 non-zero linear weight components per chunk-type, which corresponds to a sparsity of more than 80%. Most features are thus irrelevant.

Some of the best results reported previously are summarized in Table 5. All of these systems are more complex than ours. For example, the best performance in Table 5 was achieved by Kudoh and Matsumoto (2000), using a combination of 231 kernel support vector machines. 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

system	precision	recall	$F_{\beta=1}$
voted SVMs (Kudoh and Matsumoto, 2000)	93.45	93.51	93.48
WPDV models (van Halteren, 2000)	93.13	93.51	93.32
combined memory models (Sang, 2000)	94.04	91.00	92.50

Table 5: Summary of some previous results

than ours. This point can be verified from their training time of about one day on a 500Mhz Linux machine. We should mention that their result has been recently improved to 93.91 (Kudo and Matsumoto, 2001) by voting support vector machine outputs of eight different chunk-tag sequence representations. However this is not a fair comparison to our system since it is reasonable to believe that we can achieve appreciable improvement using a similar voting approach. Excluding results reported in the early version of the current work (Zhang et al., 2001), the previously second best system was a combination of five different WPDV models (van Halteren, 2000), with an overall $F_{\beta=1}$ value of 93.32. This system is again more complex than the regularized Winnow approach we propose (their best single classifier performance is $F_{\beta=1} = 92.47$). The third best performance was achieved by using combinations of memory-based models (Sang, 2000), with an overall $F_{\beta=1}$ value of 92.50. A total of eleven systems were summarized by Sang and Buchholz (2000), including a variety of statistical techniques such as maximum entropy, hidden Markov models, and transformation based rule learners. Interested readers are referred to the summary paper by Sang and Buchholz (2000) for references to all systems being tested.

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 as well as the fast online-style dual update algorithm that solves the associated optimization problem. This allows us to use a very large feature space and let the algorithm pick the relevant ones. In addition, the algorithm presented in this paper is simple. Unlike some other approaches, there is little ad hoc engineering tuning involved in our system. This simplicity allows other researchers to reproduce our results easily.

In Table 7, 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 94.17 is better than any previous system. The overall complexity of the system is still quite reasonable. The total number of features is about 5.2×10^5, with 88 nonzero features for each data point. The training time is twenty-two minutes, and the number of non-zero weight components per chunk-type is about 1.1×10^5.

It is also interesting to compare the regularized Winnow results with those of the original Winnow method. We only report results with the basic linguistic features in Table 8. In this experiment, we use the same setup as in the regularized Winnow approach. We start with a uniform prior of $\mu_j = 0.1$, and let the learning rate be $\eta = 0.01$. The Winnow update

testdata	precision	recall	$F_{\beta=1}$
ADJP	79.41	73.97	76.60
ADVP	81.65	81.18	81.41
CONJP	50.00	55.56	52.63
INTJ	100.00	50.00	66.67
LST	0.00	0.00	0.00
NP	93.80	93.99	93.89
PP	96.99	97.88	97.43
PRT	82.61	71.70	76.77
SBAR	86.99	87.48	87.23
VP	93.73	93.69	93.71
all	93.54	93.60	93.57

Table 6: Our chunk prediction results: with basic features

testdata	precision	recall	$F_{\beta=1}$
ADJP	81.68	73.29	77.26
ADVP	82.63	81.29	81.96
CONJP	55.56	55.56	55.56
INTJ	100.00	50.00	66.67
LST	0.00	0.00	0.00
NP	94.39	94.37	94.38
PP	97.64	98.03	97.83
PRT	81.44	74.53	77.83
SBAR	91.15	88.60	89.86
VP	94.38	94.78	94.58
all	94.28	94.07	94.17

Table 7: Our chunk prediction results: with enhanced features

(4) is performed thirty times repeatedly over the data. The training time is approximately the same as that of the regularized Winnow method.

Clearly the regularized Winnow method has indeed enhanced the performance of the original Winnow method. The improvement is more or less consistent over all chunk types. It can also be seen that the improvement is appreciable but not dramatic in this case. This is not too surprising since the data is very close to linearly separable. Even on the test set, the multi-class classification accuracy is around 96%. On average, the binary classification accuracy on the training set (note that we train one binary classifier for each chunk type) is close to 100%. This means that the training data is close to linearly separable. Since the benefit of regularized Winnow is more significant with noisy data, the improvement in this case is not dramatic. We should mention that for some other more noisy problems which we have tested on, the improvement of regularized Winnow method over the original Winnow method can be much more significant.

testdata	precision	recall	$F_{\beta=1}$
ADJP	77.06	70.55	73.66
ADVP	79.33	78.87	79.10
CONJP	45.45	55.56	50.00
INTJ	50.00	50.00	50.00
LST	0.00	0.00	0.00
NP	93.08	93.39	93.24
PP	96.71	97.11	96.91
PRT	82.05	60.38	69.57
SBAR	83.12	85.61	84.35
UCP	0.00	0.00	0.00
VP	93.27	93.19	93.23
all	92.56	92.80	92.68

Table 8: Chunk prediction results with original Winnow: with basic features

6.3 The Impact of Features

The experiment in this section illustrates the necessity of using many features. We use the same parameter values as in Section 6.2. However, we only use first-order basic features. The following results are obtained with regularized Winnow on the full training set and tested on the standard test set. The number of non-zero features per datum is 12. The total number of binary features is 4.8×10^4. There are about 1.4×10^4 non-zero linear weight components per chunk-type. Although the training time is significantly faster (about three minutes), the performance is much worse. This shows that using a large number of features improves the prediction accuracy.

We would also like to mention that with only the first-order basic features, the original Winnow algorithm achieves an overall precision of 90.65, recall of 88.36, and $F_{\beta=1}$ of 89.49. It is clear that the difference between the original Winnow and our modification is more

testdata	precision	recall	$F_{\beta=1}$
ADJP	83.63	64.16	72.61
ADVP	80.56	79.45	80.00
CONJP	83.33	55.56	66.67
INTJ	50.00	50.00	50.00
LST	0.00	0.00	0.00
NP	91.83	92.71	92.27
PP	96.22	97.28	96.74
PRT	80.81	75.47	78.05
SBAR	85.90	85.42	85.66
VP	92.09	92.77	92.43
all	92.07	92.36	92.22

Table 9: Chunk prediction results with only first-order basic features

significant. Since the data become less separable in this case, the result is explained by the fact that the original Winnow does not handle noisy data very well.

6.4 Error Analysis

In this section, we would like to study errors made by the original Winnow and the generalized Winnow on the test set. The analysis can reveal limitations of our system and suggest possible future improvements. We will only examine outputs with the basic features.

We find that there are about 1800 word-level errors that are made both by the original Winnow and by the generalized Winnow. There are about 500 word-level errors made by the original Winnow only, and about 200 errors made by the generalized Winnow only. Therefore the generalized Winnow reduced the word-level error rate by about 10%. Clearly their errors are highly (but not totally) correlated. This implies that hard cases for one algorithm are also likely to be hard for the other. Also since the errors are not entirely correlated, it is possible to combine different classifiers to reduce error.

We shall classify the errors into four types:

1. Errors corresponding to chunks that are inherently ambiguous.

2. Errors corresponding to chunks that are ambiguous based on features used by the system.

3. Errors corresponding to cases that are not sufficiently represented in the training data.

4. Errors that may be avoided with a better learning algorithm.

Clearly the first type of error is inherent to the problem and cannot be avoided. In general, this type of errors may be defined as the variation among different human labelers who use the same set of chunking rules for possible ambiguity resolution. They also include chunks not tagged in a consistent manner, as well as human errors. However, this type of error does not appear to dominate our test data. As an example of this type of error, we observed that in the sentence segment "Mr. Meador takes responsibility for development and property management.", "development" and "property management" were tagged as two separate noun phrases. However, both algorithms tagged the whole sub-string "development and property management" as a single noun phrase.

The second type of error is certainly caused by non-optimal feature construction. By comparing results corresponding to first order features only as in Table 9, basic features as in Table 6, and enhanced features as in Table 7, it is clear that appropriate feature choices can have significant impact on the system performance. We observe that some of the chunk labels can only be resolved with long-range features. Since we have only used local context of window size two, this leads to errors that cannot be resolved using our features. For example, in the sentence segment "These include, among other parts, ..., torque box, *fixed leading edges* for the wings and aft keel beam.", both algorithms tagged the string "fixed leading edges" as "B-VP B-NP I-NP", which is reasonable given our local context with a window size of two. Clearly, from the whole sentence, one can see that the correct tags would have been "B-NP I-NP I-NP". As another example, consider the sentence segment "..., is an aerospace *, electronics, automotive* and graphics concern.". Clearly based on the whole sentence, we shall tag the string "an aerospace , electronics , automotive and graphics concern" as a single noun-phrase. However, both algorithms tagged the sub-string ", electronics , automotive" as "O B-NP O B-NP", which is reasonable based only on our local features of window size two. Unfortunately this kind of error cannot be avoided without using global features.

The third and fourth types of errors are related. Too few cases for one algorithm could still be sufficient for another (better) algorithm. However, we know that in general giving more data will improve the performance of a statistical machine learning method. For example, in the sentence segment "Improving profitability of U.S. operations is an ...", the sub-string "Improving profitability" should be tagged as "B-VP B-NP". However, both algorithms tagged it as "B-NP I-NP", which is reasonable if we do not consider the semantics of the word "Improving". This kind of error could be avoided if we have many examples of the word "Improving" so that its proper usage can be learned.

It is also interesting to check errors that are made by the original Winnow but are avoided by the generalized Winnow. In the sentence segment "... a special charge of $ 5 million representing general and administrative expenses from ...", the sub-string "general and administrative expenses" was correctly tagged as a single noun phrase by the generalized Winnow. It was tagged as "B-ADJP I-ADJP I-ADJP B-NP" by the original Winnow. As another example, the sentence segment "... to stash away that kind of money." was correctly tagged as "B-VP I-VP B-ADVP B-NP I-NP ..." by the generalized Winnow but was incorrectly tagged as "B-VP I-VP B-NP I-NP I-NP ..." by the original Winnow. We can see that errors like this may be avoided or reduced if we use a better learning algorithm.

7. Conclusion

In this paper, we described a text chunking system based on a generalization of Winnow we call regularized Winnow. The advantage of the new method compared with the original Winnow is its ability to handle linearly non-separable data and its ability to provide reliable confidence estimates. Such confidence estimates are required in the statistical sequential modeling approach to the text chunking problem which we presented in this paper.

Since the Winnow family of algorithms 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 more efficient than other systems reported in the literature that achieved performance close to ours.

From our experiments, it is clear that good features can significantly improve system performance. Therefore the good performance of our system is in part due to the features we have used. Our error analysis also shows that many errors are inherently ambiguous using local features as in this paper. Therefore to further improve the system's performance, it is necessary to use long range features. However we do not know how to construct good long range features for text chunking. This is an interesting open problem.

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.

Appendix A. Dual Formulation for Generalized Winnow

Note that when $w \to \infty$ the objective value of (7) goes to ∞. We thus know that $\hat{w}$ is finite. Since both (7) and (8) have strictly convex objective functions, both have unique solutions. The purpose of this section is to show the following theorem which justifies the validity of the dual formulation in Section 4. A more general treatment of this type of duality were given by Zhang (2002).

Theorem 1 *The solution $\hat{\alpha}$ of (8) gives the solution $\hat{w}$ of (7) through (9).*

Proof By differentiating (7) with respect to w_j, we obtain the following first order condition which is satisfied at the optimal solution $\hat{w}$ for each j:

$$\left(\ln \frac{\hat{w}_j}{e\mu_j} + 1\right) + c\sum_{i=1}^{n} f'(\hat{w}^T x^i y^i) x_j^i y^i = 0.$$

That is,

$$\hat{w}_j = \mu_j \exp\left(\sum_{i=1}^{n} \bar{\alpha}^i x_j^i y^i\right), \tag{12}$$

where

$$\bar{\alpha}^i = -cf'(\hat{w}^T x^i y^i) = c\max(\min(1 - \hat{w}^T x^i y^i, 2), 0). \tag{13}$$

In the following, we only need to show that $\bar{\alpha}$ is the solution of (8). By definition, we have $\bar{\alpha}^i \in [0, 2c]$. We only need to show that $\bar{\alpha}$ satisfies the KKT condition of (8).

From (13), we obtain

$$\hat{w}^T x^i y^i = 1 - \frac{1}{c}\bar{\alpha}^i - \lambda_1^i + \lambda_2^i.$$

where $\lambda_1^i, \lambda_2^i \geq 0$, $\lambda_1^i(\bar{\alpha}^i - 2c) = 0$, and $\lambda_2^i \bar{\alpha}^i = 0$. Using (12), we obtain for all i:

$$\begin{aligned}
&1 - \frac{1}{c}\bar{\alpha}^i - \lambda_1^i + \lambda_2^i = \sum_j \mu_j \exp\left(\sum_{k=1}^n \bar{\alpha}^k x_j^k y^i\right) x_j^i y^i,\\
&\lambda_1^i(\bar{\alpha}^i - 2c) = 0,\\
&\lambda_2^i \bar{\alpha}^i = 0,\\
&\lambda_1^i \geq 0, \quad \lambda_2^i \geq 0.
\end{aligned}$$

It is easy to verify that the above equations are the set of KKT conditions of (8). It follows that $\bar{\alpha}$ solves (8). That is $\hat{\alpha} = \bar{\alpha}$. ■

Appendix B. Regularized Winnow as a Conditional Probability Estimator

Consider a binary classification problem as in Section 4. Assume that the in-class conditional probability is $P(y = 1|x) = q(x)$. We show that if $E_{x,y} f(w^T xy)$ is small, then $E_x ((1 + T(w^T x))/2 - q(x))^2$ is also small, where E denotes expectation, and $T(v)$ is the truncation of v into $[-1, 1]$. This implies that $(1 + T(w^T x))/2$ can be regarded as an approximation of the conditional in-class probability if we try to minimize the expected value of $f(w^T xy)$. The result justifies the choice of using loss function f.

Lemma 1 *Let $T(v) = \min(\max(v, -1), 1)$, then for all w:*

$$\begin{aligned}
E_{x,y} f(w^T xy) =& 2E_x\, q(x)(1 - q(x)) + \\
& E_x \frac{1}{2}(2q(x) - 1 - T(w^T x))^2 + E_x |2q(x) - 1 - T(w^T x)||w^T x - T(w^T x)|.
\end{aligned}$$

Proof Let q be a number in $[0, 1]$. Consider the following situations:

- $p \in [0, 1]$: it is easy to verify that
$$qf(p) + (1 - q)f(-p) = \frac{q}{2}(p - 1)^2 + \frac{1 - q}{2}(p + 1)^2 = 2q(1 - q) + \frac{1}{2}(2q - 1 - p)^2.$$
- $p \geq 1$: it is easy to verify that
$$qf(p) + (1 - q)f(-p) = (1 - q)(2p) = 2q(1 - q) + \frac{1}{2}(2q - 1 - 1)^2 + |2q - 1 - 1|(p - 1).$$
- $p \leq -1$: it is easy to verify that
$$qf(p) + (1 - q)f(-p) = q(-2p) = 2q(1 - q) + \frac{1}{2}(2q - 1 + 1)^2 + |2q - 1 + 1|(-p - 1).$$

Summarize the above, we have the following inequality for all $q \in [0,1]$ and all p:

$$qf(p) + (1-q)f(-p) = 2q(1-q) + (2q-1-T(p))^2 + |2q-1-T(p)||p-T(p)|.$$

The lemma follows from this equality and the observation that

$$E_{x,y}\, f(w^T xy) = E_x\, \left[q(x)f(w^T x) + (1-q(x))f(-w^T x)\right].$$

■

Theorem 2 *Assume that there exists $\bar{w}$ such that $E_x\,(2q(x)-1-T(\bar{w}^T x))^2 \le \epsilon_1$ and $\bar{w}^T x \le \kappa$ for all x where $\kappa \ge 1$. Assume further that we find $\hat{w}$ that approximately minimizes f as:*

$$E_{x,y}\, f(\hat{w}^T xy) \le \inf_w E_{x,y}\, f(w^T xy) + \epsilon_2,$$

then

$$E_x\,(2q(x)-1-T(\hat{w}^T x))^2 \le \epsilon_1 + (\kappa-1)\epsilon_1^{1/2} + \epsilon_2.$$

Proof Note that from the Schwartz inequality we obtain

$$E_x\,|2q(x)-1-T(\bar{w}^T x))||\bar{w}^T x - T(\bar{w}^T x)| \le (\kappa-1)(E_x\,(2q(x)-1-T(\bar{w}^T x))^2)^{1/2} \le (\kappa-1)\epsilon_1^{1/2}.$$

Now using Lemma 1, we have

$$E_{x,y}\, f(\bar{w}^T xy) - 2E_x\, q(x)(1-q(x)) \le \epsilon_1 + (\kappa-1)\epsilon_1^{1/2}.$$

Again, from Lemma 1 and the assumptions of the theorem, we obtain

$$\begin{aligned} E_x\,(2q(x)-1-T(\hat{w}^T x))^2 \le& E_{x,y}\, f(\hat{w}^T xy) - 2E_x q(x)(1-q(x)) \\ \le& E_{x,y}\, f(\bar{w}^T xy) - 2E_x q(x)(1-q(x)) + \epsilon_2 \\ \le& \epsilon_1 + (\kappa-1)\epsilon_1^{1/2} + \epsilon_2. \end{aligned}$$

■

This implies that by approximately minimizing the expected value $f(w^T xy)$ as in (7), we effectively try to obtain a weight $\hat{w}$ such that $T(\hat{w}^T x) \approx 2q(x) - 1$. Therefore $(1 + T(\hat{w}^T x))/2$ can be regarded as an approximation to the conditional in-class probability $q(x)$. In particular, if $q(x) = T(\bar{w}^T x)$ for some $\bar{w}$, then the theorem implies that by approximately minimizing the expected value of $f(w^T xy)$ as in (7), we can obtain $\hat{w}$ such that $T(\hat{w}^T x)$ is arbitrarily close to $q(x)$.

References

S. P. Abney. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 257–278. Kluwer, Dordrecht, 1991.

Eric Brill. Some advances in rule-based part of speech tagging. In *Proc. AAAI 94*, pages 722–727, 1994.

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

C. Gentile and M. K. Warmuth. Linear hinge loss and average margin. In *Proc. NIPS'98*, 1998.

A. Grove and D. Roth. Linear concepts and hidden variables. *Machine Learning*, 42: 123–141, 2001.

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

J. Kivinen and M.K. Warmuth. Additive versus exponentiated gradient updates for linear prediction. *Journal of Information and Computation*, 132:1–64, 1997.

Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *NAACL-2001*, 2001.

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

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

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

Ion Muslea. Extraction patterns for information extraction tasks: A survey. In *AAAI workshop on machine learning for information extraction*, pages 1–6, 1999.

Vasin Punyakanok and Dan Roth. The use of classifiers in sequential inference. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 995–1001. MIT Press, 2001.

Erik F. Tjong Kim Sang. Text chunking by system combination. In *Proc. CoNLL-2000 and LLL-2000*, 2000.

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

Hans van Halteren. Chunking with wpdv models. In *Proc. CoNLL-2000 and LLL-2000*, pages 154–156, 2000.

Tong Zhang. Regularized Winnow methods. In *Advances in Neural Information Processing Systems 13*, pages 703–709, 2001.

Tong Zhang. On the dual formulation of regularized linear systems. *Machine Learning*, 46: 91–129, 2002.

Tong Zhang, Fred Damerau, and David E. Johnson. Text chunking using regularized Winnow. In *39th Annual Meeting of the Association for Computational Linguistics*, pages 539–546, 2001.