A Sequential Dual Method for Structural SVMs

P. Balamurugan* Shirish Shevade† S. Sundararajan‡ S. Sathiya Keerthi§

Abstract

In many real world prediction problems the output is a structured object like a sequence or a tree or a graph. Such problems range from natural language processing to computational biology or computer vision and have been tackled using algorithms, referred to as structured output learning algorithms. We consider the problem of structured classification. In the last few years, large margin classifiers like support vector machines (SVMs) have shown much promise for structured output learning. The related optimization problem is a convex quadratic program (QP) with a large number of constraints, which makes the problem intractable for large data sets. This paper proposes a fast sequential dual method (SDM) for structural SVMs. The method makes repeated passes over the training set and optimizes the dual variables associated with one example at a time. The use of additional heuristics makes the proposed method more efficient. We present an extensive empirical evaluation of the proposed method on several sequence learning problems. Our experiments on large data sets demonstrate that the proposed method is an order of magnitude faster than state of the art methods like cutting-plane method and stochastic gradient descent method (SGD). Further, SDM reaches steady state generalization performance faster than the SGD method. The proposed SDM is thus a useful alternative for large scale structured output learning.

1 Introduction

Support Vector Machines (SVMs) have gained wide popularity over the last decade. They have exhibited good generalization performance on problems in numerous applications including text categorization, face detection, speaker identification and many others. SVMs were originally designed for binary classification problems where the idea was to design an optimal separating hyperplane which separates the two classes with maximum margin [1]. Learning an SVM classifier amounts to solving a convex quadratic programming problem. In several data mining applications, the training set has a large number of labeled examples residing in a high dimensional space. Many efficient algorithms have been proposed to solve such large scale classification problems [9, 7].

In recent years significant advances have been made in extending SVMs to other supervised learning problems such as multi-category classification, regression and ordinal regression. In all these problems, the labels of the training set are discrete or univariate. There, however, exist many prediction problems in which the output is structured. Consider the parsing problem in natural language processing. Here, the input is a sentence and the output is a parse tree. In gene finding problem in computational biology, the input is a DNA sequence while the output is a sequence indicating the presence of genes. Such problems are referred to as structured classification or structured prediction problems. Note that in such problems, the components of the output vector belong to a finite set and are not independent. Structured output framework has become very popular and has been adopted in several other applications such as document summarization or image segmentation.

In this work, we focus on the problem of training structural SVMs. Learning structural SVMs amounts to solving a convex quadratic program (QP) with a huge number of constraints. The number of constraints is typically exponential, which makes the problem intractable by general purpose optimization methods. Some algorithms use polynomial-size reformulation of the training problem [23] and use decomposition methods similar to sequential minimal optimization (SMO) [19]. Such algorithms are however restricted to applications where the polynomial size reformulation exists. Particularly relevant to the work in this paper are the algorithms which work directly with the original QP with exponential number of constraints. These algorithms use polynomially-sized subset of constraints from the original QP and solve this restricted QP to attain a solution of sufficient accuracy. They are based on cutting-plane method [25, 10]. Stochastic gradient method [22] (SGD) can also be used for this purpose. The algorithm in [25] uses an efficient method to construct cutting-planes. It however needs to solve QPs of increasing size as the data set size grows.

In a recent work, Joachims et al [10] proposed an extension of the cutting-plane method, presented in [9], for training linear structural SVMs. It was demonstrated that on large data sets, this method is several orders of magnitude faster than conventional cutting-plane method [25]. On a benchmark data set

*Computer Science and Automation, Indian Institute of Science, Bangalore, India. balamurugan@csa.iisc.ernet.in

†Computer Science and Automation, Indian Institute of Science, Bangalore, India. shirish@csa.iisc.ernet.in

‡Yahoo! Labs, Bangalore, India. ssrajan@yahoo-inc.com

§Yahoo! Labs, Santa Clara, USA. selvarak@yahoo-inc.com

with about 35000 examples and 1.8×10^7 features, this method required about an hour to construct a decision function on a reasonably fast machine. Thus, there is a need to design a fast algorithm for structured output learning.

Contributions: In this work, we present a fast sequential dual method (SDM) for training structural SVMs. The method sequentially traverses through all the examples and optimizes the dual variables associated with one example at a time. The method is general purpose and can be effectively used for any structured output learning problem. The use of additional heuristics makes our method even faster. Experiments on several sequence learning problems demonstrate that our method is faster than the extended cutting-plane method of Joachims et al [10] and stochastic gradient descent method [3] by an order of magnitude. Further, SDM reaches steady state generalization performance faster than the SGD method.

The rest of this paper is organized as follows. The next Section details the structural SVM problem formulation. The related work is described in Section 3. In Section 4 we present the proposed SDM for structural SVM. Section 5 compares our method with other state-of-the-art methods for sequence learning on different benchmark data sets. Section 6 concludes the paper.

2 Structural support vector machines

In this section, we describe the structured prediction problem and give details of the dual solution. Let $\mathscr{X}$ denote the input space and $\mathscr{Y}$ denote the space of structured outputs. The structured output prediction problem is to learn a function, h,

$$h : \mathscr{X} \rightarrow \mathscr{Y}.$$

For example, in the case of gene finding, $\mathscr{X}$ is the space of DNA sequences, while $\mathscr{Y}$ is the space of sequences that indicate the presence of genes. We assume that a training set, S, of input-output pairs is available. Let $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n \in (\mathscr{X} \times \mathscr{Y})^n$ where $\mathbf{x}_i \in \mathscr{X}$ and $\mathbf{y}_i \in \mathscr{Y}$ for every i. Depending on the application a pair $(\mathbf{x}, \mathbf{y})$ can represent different kinds of objects. For example, in sequence learning $\mathbf{y}$ is the output sequence associated with the input sequence $\mathbf{x}$. In parse tree learning $\mathbf{x}$ corresponds to a sequence of words and $\mathbf{y}$ denotes the associated parse tree. If $\mathbf{x}$ is a sequence of length L, then $\mathbf{y}$ can be any sequence of length L generated from a given set of alphabets. This makes the cardinality of $\mathscr{Y}$ grow exponentially with the size of $\mathbf{x}$. The goal of structured output learning is to use the training set S to learn a discriminant function $g : \mathscr{X} \times \mathscr{Y} \rightarrow \mathbb{R}$ such that the prediction for a input $\mathbf{x}$ is given by

$$h(\mathbf{x}) = \arg\max_{\mathbf{y} \in \mathscr{Y}} g(\mathbf{x}, \mathbf{y}). \tag{2.1}$$

The function g, in some sense, measures how good a prediction $\mathbf{y}$ is for a given input $\mathbf{x}$. In this work, we assume that $g(\mathbf{x}, \mathbf{y})$ takes the form of a linear function

$$g(\mathbf{x}, \mathbf{y}) = \mathbf{w}^T f(\mathbf{x}, \mathbf{y})$$

where $\mathbf{w}$ is a parameter vector and $f(\mathbf{x}, \mathbf{y})$ is a feature vector relating input $\mathbf{x}$ and $\mathbf{y}$. The feature vector $f(\mathbf{x}, \mathbf{y})$ has a crucial effect on the performance of the designed structured classifier [13].

Clearly, the arg max computation in (2.1) plays a key role in structural SVMs. As we will see below, a slightly modified version of it plays a crucial role in the training process. For sequential and tree-structured output variables dynamic programming approaches such as the Viterbi algorithm and the Cook-Kasami-Younger (CKY) algorithm can be employed to do these computations efficiently.

A structured output problem can be posed as an optimization problem that is similar to multi-class SVMs [5]. Let $\Delta f_i(\mathbf{y}) = f(\mathbf{x}_i, \mathbf{y}_i) - f(\mathbf{x}_i, \mathbf{y})$. Then the primal optimization problem associated with the training of structural SVMs is (**OP1**):

$$\begin{aligned} \min_{\mathbf{w}, \xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_i \xi_i \\ \text{s.t.} \quad & \mathbf{w}^T \Delta f_i(\mathbf{y}) \geq l_i(\mathbf{y}) - \xi_i \;\; \forall\, i, \mathbf{y} \end{aligned} \tag{2.2}$$

where $C > 0$ is a regularization parameter. $l_i(\mathbf{y})$ in (2.2) is a loss function that quantifies the loss associated with predicting $\mathbf{y}$ when the correct output is $\mathbf{y}_i$. It is appropriate to require l to satisfy $l_i(y_i) = 0$. In sequence learning problem, a natural choice for the loss function is the Hamming distance,

$$l_i(\mathbf{y}) = \sum_{j=1}^{L} I(y_i^j \neq y^j)$$

where $I(\cdot)$ is the indicator function and $\mathbf{y} = (y^1, y^2, \ldots, y^L)^T$. The objective function in (2.2) is an extension of the one used in SVMs for classification. The constraints ensure that for each example $(\mathbf{x}_i, \mathbf{y}_i)$, $\mathbf{w}^T f(\mathbf{x}_i, \mathbf{y}_i)$ must be greater than $\mathbf{w}^T f(\mathbf{x}_i, \mathbf{y})$ for every incorrect $\mathbf{y} \in \mathscr{Y}$ by a required margin $l_i(\mathbf{y})$. Note also that the constraint corresponding to the correct label $\mathbf{y}_i$ for the i-th example ensures non-negativity of ξ_i.

The dual problem of (OP1) involves dual variables $\boldsymbol{\alpha}_i(\mathbf{y}), \mathbf{y} \in \mathscr{Y}\ \forall\, i$. The $\mathbf{w}$ vector used in the prediction is defined as

$$\mathbf{w}(\boldsymbol{\alpha}) = C \sum_{i, \mathbf{y}} \boldsymbol{\alpha}_i(\mathbf{y}) \Delta f_i(\mathbf{y}). \tag{2.3}$$

For notational convenience, we write this simply as $\mathbf{w}$. A scaled version of the dual of (OP1) is (**OP2**):

$$\begin{array}{ll} \min & \frac{C}{2}\| \sum_{i,\mathbf{y}} \boldsymbol{\alpha}_i(\mathbf{y}) \Delta f_i(\mathbf{y})\|^2 - \sum_{i,\mathbf{y}} \boldsymbol{\alpha}_i(\mathbf{y}) l_i(\mathbf{y}) \\ \text{s.t.} & \sum_{\mathbf{y}} \boldsymbol{\alpha}_i(\mathbf{y}) = 1 \; \forall \; i \\ & \boldsymbol{\alpha}_i(\mathbf{y}) \geq 0 \; \forall \; i, \mathbf{y} \end{array}$$

At optimality the objective function of (OP1) is $-C$ times the objective function in (OP2). (OP2) is a convex quadratic programming problem with linear constraints. Optimality of $\boldsymbol{\alpha}$ for (OP2) can be checked using the quantity

$$\psi_i = \max_{\mathbf{y}} F_i(\mathbf{y}) - \min_{y:\boldsymbol{\alpha}_i(\mathbf{y})>0} F_i(\mathbf{y}) \tag{2.4}$$

where

$$F_i(\mathbf{y}) = l_i(\mathbf{y}) - \mathbf{w}^T \Delta f_i(\mathbf{y}). \tag{2.5}$$

$F_i(\mathbf{y})$ is the negative of the derivative of the objective function in (OP2) with respect to $\boldsymbol{\alpha}_i(\mathbf{y})$. From (2.4) it is clear that $\psi_i \geq 0$. Further, at optimality,

$$\psi_i = 0 \;\; \forall \; i.$$

For practical purposes we can approximately check the dual optimality using a positive tolerance parameter, τ:

$$\psi_i < \tau \;\; \forall \; i. \tag{2.6}$$

We refer to the above condition as τ-optimality.

Clearly, the computation of $\max_{\mathbf{y}} F_i(\mathbf{y})$ (needed for checking optimality via (2.4)) plays a crucial role in the training of structural SVMs. The computation is a slight modification of (2.1); under mild conditions dynamic programming based algorithms such as Viterbi and CKY can be employed for efficiently doing this computation too. Still, the computation is comparatively expensive and hence it has to be used sparsely.

3 Related Work

We now describe various approaches for training structural SVMs.

In general finding the solution of (OP1) is not trivial. The key challenge in solving (OP1) is the extremely large number, $O(n|\mathscr{Y}|)$, of constraints. The size of the output space $\mathscr{Y}$, that is the number of possible assignments $\mathbf{y}$ given $\mathbf{x}$, grows exponentially in the description length of $\mathbf{y}$. For example, in the sequence learning problem, $\mathscr{Y}$ consists of all possible output sequences of length L for the input $\mathbf{x}$ of the same length. This makes (OP1) intractable using any off-the-shelf technique.

Making use of the fact that a polynomially-sized subset of the constraints from the quadratic program (OP1) is sufficient for obtaining a solution of arbitrary accuracy, algorithms were developed which work directly with (OP1). Tsochantaridis et al [25] exploited the special structure of (2.2) and proposed an efficient cutting plane algorithm which aims at finding a small set of constraints that ensures a sufficiently approximate solution in polynomial time. The algorithm merely requires a separation oracle that efficiently finds the most violated constraint, $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} F_i(\mathbf{y})$. For each training set example, the algorithm maintains a working set $\mathscr{W}_i$ which is empty initially. Iterating through the training set examples, the algorithm finds the most violated constraint involving some output $\hat{\mathbf{y}}$, checks if the constraint is violated by more than some desired precision ϵ, and adds it to the respective working set. The above optimization problem is solved with respect to the working set $\mathscr{W} = \cup_{i=1}^{n} \mathscr{W}_i$. This procedure is repeated until no constraint is added to the set $\mathscr{W}$ in one loop through the training set examples. Although the learning problem is exponential in size, this algorithm has been proved to have polynomial time convergence on a large class of problems [25]. In practice, the cutting plane model size grows linearly with the data set size [25], which in turn requires solutions of Quadratic Programs (QPs) of increasing size. This causes the algorithm to have superlinear runtime [10].

Joachims et al [10] proposed an equivalent reformulation of the problem (2.2) and presented a cutting-plane algorithm whose runtime is $O(n)$ and is significantly faster than the method proposed by Tsochantaridis et al [25] on large scale problems. They considered the following problem:

$$\begin{aligned} \min_{\mathbf{w},\xi} \;\; & \frac{1}{2}\|\mathbf{w}\|^2 + C\xi \\ \text{s.t.} \;\; & \frac{1}{n}\mathbf{w}^T \sum_{i=1}^{n} \Delta f_i(\bar{\mathbf{y}}_i) \geq \frac{1}{n} \sum_{i=1}^{n} l_i(\bar{\mathbf{y}}_i) - \xi, \\ & \qquad \forall \; (\bar{\mathbf{y}}_1, \ldots, \bar{\mathbf{y}}_n) \in \mathscr{Y}^n \end{aligned} \tag{3.1}$$

Note that in the above formulation, there is only one slack variable ξ shared across all constraints. But, the number of constraints is $|\mathscr{Y}|^n$. This formulation is also referred to as "1-slack structural SVM (with margin rescaling)". Joachims et al [10] showed that the dual problem of (3.1) has a sparse solution, i.e., the solution has a small number of non-zero dual variables. It was also shown that the number of cutting-planes and the number of iterations are independent of the training set size n. Empirically, it was observed that the size of the QPs that need to be solved is very small, even for very large data sets.

Taskar [24] proposed a structured SMO algorithm to solve (OP2). Instead of working with the exponentially large number of dual variables $\boldsymbol{\alpha_i}$, he suggested

to use polynomially many "marginal" variables. These variables are used to identify a pair $\alpha_i(\mathbf{y}_1)$ and $\alpha_i(\mathbf{y}_2)$ for joint optimization, based on KKT violation. After the SMO update the dual variables are projected back to the marginal variables. This procedure is repeated until KKT optimality conditions are satisfied.

While the linear models discussed so far are margin based, there also exist probabilistic linear discriminative models. Such models are based on extensions of logistic regression. Particularly relevant to the work in this paper is the conditional random field (CRF) model [15], originally proposed by Lafferty et al [15] for sequence learning and later extended to more complex graphical models. The close connection to logistic regression can be utilized effectively for training a CRF.

Memisevic [17] proposed an SMO algorithm to solve the dual of the kernelized version of the following primal problem:

$$\begin{array}{ll} \min & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \log\sum_{\mathbf{y}} \exp(-\xi^i_{\mathbf{y}}) \\ \text{s.t.} & \xi^i_{\mathbf{y}} = \mathbf{w}^T\Delta f_i(\mathbf{y}) \;\; \forall\, i, \mathbf{y} \end{array}$$

SMO algorithm requires a strategy to select two dual variables for optimization at any point of time. The author [17] proposed a method for choosing these variables and it performed well on multi-class classification problems.

Gradient based online methods like stochastic gradient descent (SGD) and others [4, 20] can be used for solving the primal problem. SGD methods are fast and are especially useful when the training data set size is large [3]. A thorny issue associated with online methods is the choice of the learning rate value. The sequential dual method that we propose in this paper has an online feel to it, even though it is fundamentally a batch algorithm. Adjustments of weight parameters are automatically inbuilt into it. As we will see later SDM reaches steady state generalization performance much faster than SGD. OLaRank [2] is a semi-online algorithm which revisits old examples and improves their dual variables. It is mainly set up for achieving very good performance within one pass over a given set of examples. On hard data sets OLaRank does not achieve steady state generalization performance after a single pass over the training set.

The work in this paper can be viewed as an extension, to structural SVMs, of the ideas given in [7, 12] for binary and multi-class problems. The extension is challenging due to the complex nature of the subproblem associated with each example.

In this work, we compare our method with state-of-the-art approaches like the cutting plane method proposed in [11] and the stochastic gradient descent method [3] whose implementations are available online.

4 SDM for Structural SVM

We now present our sequential dual method for structural SVMs. The idea is to traverse through the training set examples sequentially and solve the sub-problem of (OP2) restricted to the i-th example. This procedure is repeated until KKT optimality conditions are satisfied. We first give the basic algorithm of SDM and then propose different heuristics to enhance its speed.

A generic step is to pick a single example i, and solve the following optimization problem which is the restriction of OP2 to the block of dual variables $\boldsymbol{\alpha_i}$, associated with that example:

$$\begin{array}{ll} \min\limits_{\boldsymbol{\delta\alpha_i}} & \frac{C}{2}\|\sum\limits_{\mathbf{y}} \boldsymbol{\delta\alpha_i}(\mathbf{y})\Delta f_i(\mathbf{y})\|^2 - \sum\limits_{\mathbf{y}} \boldsymbol{\delta\alpha_i}(\mathbf{y})F_i(\mathbf{y}) \\ \text{s.t.} & \sum\limits_{\mathbf{y}} \boldsymbol{\delta\alpha_i}(\mathbf{y}) = 0 \\ & \boldsymbol{\delta\alpha_i}(\mathbf{y}) \geq -\boldsymbol{\alpha}_i(\mathbf{y}) \;\forall \mathbf{y} \end{array} \tag{4.1}$$

where the vector $\boldsymbol{\delta\alpha_i}$ denotes the change in $\boldsymbol{\alpha_i}$. Note that the constraints in the above problem ensure dual feasibility. The solution of (4.1) is done till τ-optimality in (2.6) is achieved. Algorithm 1 presents the details of the overall algorithm. The value $\tau = 0.25$ is a good choice for a practical implementation of SDM for structural SVMs.

Algorithm 1 *SDM Algorithm to solve* **OP2**

Input $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, C, τ
Initialize $\boldsymbol{\alpha}_i(\mathbf{y}) = 0 \;\forall\, i, \mathbf{y}$
repeat
  for $i = 1, \ldots, n$ **do**
    Solve (4.1)
    $\boldsymbol{\alpha}_i(\mathbf{y}) := \boldsymbol{\alpha}_i(\mathbf{y}) + \boldsymbol{\delta\alpha_i}(\mathbf{y})$
    $\mathbf{w} := \mathbf{w} + C\sum_{i,\mathbf{y}} \boldsymbol{\delta\alpha_i}(\mathbf{y})\Delta f_i(\mathbf{y})$
  end for
  Compute $\psi_i \;\forall\, i$ using (2.4)
  $\psi_{max} = \max_{1\leq i\leq n} \psi_i$
until $\psi_{max} \leq \tau$

Note that in Algorithm 1, the solution of (4.1) and computation of $\psi_i \;\forall\, i$ are needed repeatedly. Both are computationally expensive especially when the number of possible $\mathbf{y}$'s is large.

We first address the problem of solving (4.1). It is most often the case that, at optimality the set $V_i = \{\mathbf{y} : \boldsymbol{\alpha}_i(\mathbf{y}) > 0\}$ is quite small in size; in many cases V_i is just a singleton. Thus, instead of solving the large dimensional problem (4.1) involving all the variables it is more efficient to maintain a small set V_i and solve the

following optimization sub-problem (**OP-SUB**)[1]:

$$\begin{aligned}\min_{\boldsymbol{\delta\alpha_i}} \quad & \frac{C}{2}\|\sum_{\mathbf{y}\in V_i}\boldsymbol{\delta\alpha_i}(\mathbf{y})\Delta f_i(\mathbf{y})\|^2 - \sum_{\mathbf{y}\in V_i}\boldsymbol{\delta\alpha_i}(\mathbf{y})F_i(\mathbf{y}) \\ \text{s.t.} \quad & \sum_{\mathbf{y}\in V_i}\boldsymbol{\delta\alpha_i}(\mathbf{y}) = 0 \\ & \boldsymbol{\delta\alpha_i}(\mathbf{y}) \geq -\boldsymbol{\alpha}_i(\mathbf{y}) \;\forall\, \mathbf{y}\in V_i\end{aligned}$$

One could employ any off-the-shelf QP solver for solving this problem. Alternatively, a simple decomposition algorithm like Sequential Minimal Optimization (SMO) could be deployed to solve (OP-SUB). We prefer this method since it is efficient, simpler to implement and avoids dependence on external QP solvers. Note the presence of a linear equality constraint in (OP-SUB). The SMO algorithm breaks the problem down into two-dimensional sub-problems that may be solved analytically. The variables involved in the two-dimensional optimization sub-problem can be chosen by using the "maximum violating pair" strategy. Let K denote a matrix whose (p,q)-th entry is the inner product of $\Delta f_i(y_p)$ and $\Delta f_i(y_q)$, and let $K_{q,\cdot}$ denote its q-th row. The size of K is $|V_i| \times |V_i|$. Also, let

$$\begin{aligned} M_q &= C(K_{q,\cdot}\boldsymbol{\delta\alpha_i}) - F_i(\mathbf{y}_q) \\ &\quad \forall\, q \text{ s.t. } \mathbf{y}_q \in V_i, -\alpha_i(\mathbf{y}_q) < \delta\alpha_i(\mathbf{y}_q) \\ m_p &= C(K_{p,\cdot}\boldsymbol{\delta\alpha_i}) - F_i(\mathbf{y}_p) \;\forall\, p \ni \mathbf{y}_p \in V_i \end{aligned} \quad (4.2)$$

Using these variables, we define

$$q^* = \arg\max_q M_q \;\text{ and }\; p^* = \arg\min_p m_p. \quad (4.3)$$

Then, KKT optimality conditions for (OP-SUB) can be written as

$$\tilde{\psi}_i \triangleq M_{q^*} - m_{p^*} = 0. \quad (4.4)$$

For practical implementation, one can use the following $\tilde{\tau}$-optimality condition:

$$\tilde{\psi}_i \leq \tilde{\tau}. \quad (4.5)$$

where $\tilde{\tau}$ is a suitable tolerance parameter, chosen to be smaller than τ. The SMO algorithm finds q^* and p^* and solves the restricted subproblem (OP-SUB) with respect to $\delta\alpha_i(y_{p^*})$ and $\delta\alpha_i(y_{q^*})$. This procedure is repeated until the $\tilde{\tau}$-optimality condition (4.5) is satisfied. If δ denotes the change in $\delta\alpha_i(y_{p^*})$, then the change in $\delta\alpha_i(y_{q^*})$ will be $-\delta$ subject to the feasibility given by the constraints. Therefore, the restricted sub-problem in (OP-SUB) reduces to a one-dimensional linearly constrained quadratic programming problem in δ. It is easy to verify that the optimal solution of this restricted sub-problem is δ^* given by

$$\max(-\alpha_i(y_{p^*}), \min(\alpha_i(y_{q^*}), \frac{F_i(y_{p^*}) - F_i(y_{q^*})}{\|\Delta f_i(y_{p^*}) - \Delta f_i(y_{q^*})\|^2})). \quad (4.6)$$

Algorithm 2 describes this procedure.

Algorithm 2 *SMO Algorithm to solve* **OP-SUB**

1: Input $(\mathbf{x}_i, \mathbf{y}_i)$, V_i, C, $\tilde{\tau}$
2: Initialize $\boldsymbol{\delta\alpha_i}(\mathbf{y}) = 0 \;\forall\, \mathbf{y}\in V_i$
3: Find p^* and q^* using (4.3) and $\tilde{\psi}_i$ using (4.4)
4: **while** the condition (4.5) is not satisfied **do**
5: Calculate δ^* using (4.6)
6: $\delta\alpha_i(y_{p^*}) := \delta\alpha_i(y_{p^*}) + \delta^*$
7: $\delta\alpha_i(y_{q^*}) := \delta\alpha_i(y_{q^*}) - \delta^*$
8: Find p^* and q^* using (4.3) and $\tilde{\psi}_i$ using (4.4)
9: **end while**

Note that the optimality condition (4.5) is with respect to the set V_i and does not ensure that the optimality condition in (2.6) is satisfied. The condition (2.6) requires computation of ψ_i, which in turn needs $\max_{\mathbf{y}} F_i(\mathbf{y})$ (see (2.4)), a quantity that is computationally expensive to compute; therefore, from the efficiency viewpoint doing this computation in every iteration should be avoided.

We therefore resort to a two-loop approach that is described in Algorithm 3. The difference between the two loops is that the type-I loop (indicated by GetMaxY = 1) computes $\hat{\mathbf{y}}$ using $\arg\max_{\mathbf{y}\in\mathscr{Y}} F_i(\mathbf{y})$ (step 9 in Algorithm 3) and adds it to the set V_i, if necessary, before solving the restricted sub-problem (OP-SUB) (step 15). On the other hand, the type-II loop (GetMaxY = 0) uses the sets V_i to solve the problem (OP-SUB) (step 19). As mentioned earlier, the computation of $\max_{\mathbf{y}\in\mathscr{Y}} F_i(\mathbf{y})$ is expensive and therefore, $\hat{\mathbf{y}}$ should be found as and when needed. The proposed SDM spends more time in executing the type-II loop compared to the type-I loop, thereby avoiding the frequent computation of $\hat{\mathbf{y}}$.

The type-I loop iterates through all the examples sequentially. For each example i, it finds $\hat{\mathbf{y}}$ and computes ψ_i.[2] If $\psi_i > \tau$, it modifies V_i as follows.

$$\begin{aligned} \hat{\mathbf{y}} &= \arg\max_{\mathbf{y}} F_i(\mathbf{y}) \\ \alpha_i(\hat{\mathbf{y}}) &= 0 \text{ if } \hat{\mathbf{y}} \notin V_i \\ V_i &= V_i \cup \{\hat{\mathbf{y}}\} \end{aligned} \quad (4.7)$$

[1] This approach of solving the optimization sub-problem with respect to the variables associated with the set V_i makes the proposed method significantly different from the sequential dual method for multi-class SVMs.

[2] The KKT conditions at optimality ensure that one can compute ψ_i as mentioned in step 10 (Algorithm 3) instead of using (2.4).

Algorithm 3 *SDM Algorithm (with heuristics) to solve* **OP2**

1: Input $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, C
2: $\mathbf{w} = 0, V_i = \{\mathbf{y}_i\}$, $\alpha_i(\mathbf{y}_i) = 1 \ \ \forall i = 1, 2, \cdots, n$
3: $\tau = .25, \tilde{\tau} = .15, \kappa_1 = 10, \kappa_2 = 5$
4: $iter = 0, GetMaxY = 1, stopflag = 0, \tau_v = 0$
5: **repeat**
6: ChangedInOuterLoop=0
7: **for** $i = 1, \ldots, n$ **do**
8: **if** GetMaxY == 1 **then**
9: $\hat{\mathbf{y}}_i = \arg\max_{\mathbf{y}} F_i(\mathbf{y})$.
10: $\psi_i = F_i(\hat{\mathbf{y}}_i) - \min_{\mathbf{y} \in V_i} F_i(\mathbf{y})$.
11: **if** $\psi_i > \tau$ **then**
12: **if** $\hat{\mathbf{y}}_i \notin V_i$ **then**
13: $V_i = V_i \cup \{\hat{\mathbf{y}}_i\}$, $\boldsymbol{\alpha}_i(\hat{\mathbf{y}}_i) = 0$.
14: **end if**
15: Solve (OP-SUB) using Algorithm 2.
16: ChangedInOuterLoop := 1
17: **end if**
18: **else**
19: Solve (OP-SUB) using Algorithm 2.
20: **end if**
21: $\boldsymbol{\alpha}_i(\mathbf{y}) := \boldsymbol{\alpha}_i(\mathbf{y}) + \boldsymbol{\delta\alpha_i(y)} \ \forall \ \mathbf{y} \in V_i$
22: $\mathbf{w} := \mathbf{w} + C \sum_{\mathbf{y} \in V_i} \boldsymbol{\delta\alpha_i(y)} \Delta f_i(\mathbf{y})$
23: $V_i = V_i \setminus \{y : \boldsymbol{\alpha}_i(\mathbf{y}) = 0\}$
24: **end for**
25: Calculate $\hat{\psi}_i \ \forall \ i$ using (4.9)
26: $\hat{\psi}_{max} = \max_i \hat{\psi}_i$
27: **if** GetMaxY == 1 **then**
28: **if** ChangedInOuterLoop == 0 **then**
29: stopflag=1
30: **else**
31: **if** $iter \geq \kappa_1$ **and** $\hat{\psi}_{max} > \tau$ **then**
32: $\tau_v = \frac{\hat{\psi}_{max}}{2}$
33: GetMaxY=0
34: **end if**
35: **end if**
36: **else**
37: **if** $\hat{\psi}_{max} < \tau_v$ **or** $(iter - \kappa_1)\%\kappa_2 == 0$ **then**
38: GetMaxY=1
39: **end if**
40: **end if**
41: $iter := iter + 1$
42: **until** $stopflag == 1$

Then it solves (OP-SUB), and updates $\boldsymbol{\alpha}_i(\mathbf{y})$ and $\mathbf{w}$ as:

$$\begin{aligned} \boldsymbol{\alpha}_i(\mathbf{y}) &:= \boldsymbol{\alpha}_i(\mathbf{y}) + \boldsymbol{\delta\alpha_i(y)} \ \forall \ \mathbf{y} \in V_i \\ \mathbf{w} &:= \mathbf{w} + C \sum_{\mathbf{y} \in V_i} \boldsymbol{\delta\alpha_i(y)} \Delta f_i(\mathbf{y}). \end{aligned} \tag{4.8}$$

If the optimality conditions (2.6) are satisfied the algorithm exits. Otherwise it enters the type-II loop, iterates through all the examples and solves the problem (OP-SUB) until (4.5) is satisfied. The control then goes back to the type-I loop. The type-I loop terminates when $\psi_i \leq \tau$ (indicated by, ChangedInOuterLoop = 0, step 28 in Algorithm 3). Note also that if τ-optimality condition is satisfied with respect to the set V_i for every example i, that is, $\hat{\psi}_i \leq \tau$ (or $\hat{\psi}_{max} \leq \tau$) where

$$\hat{\psi}_i = \max_{\mathbf{y} \in V_i} F_i(\mathbf{y}) - \min_{\mathbf{y} \in V_i} F_i(\mathbf{y}) \ \forall \ i, \tag{4.9}$$

the algorithm executes the type-I loop again to ensure that $\psi_i \leq \tau$ for every example i before it terminates.

4.1 Heuristics for improving efficiency We now discuss various heuristics deployed in Algorithm 3 to make it more efficient.

The algorithm traverses through the training set examples in the order $i = 1, \ldots, n$ for updating the corresponding dual variables. Any systematic regularities in this order may lead to slow convergence of the algorithm. So, in our SDM implementation, we randomly permute the training set examples in every iteration.

The initial κ_1 iterations of the type-I loop are used to build the sets V_i to a reasonable size. During these iterations, the type-II loop iterations are not executed. This avoids fast switching between the type-I and type-II loops, especially during the initial stages of the algorithm. κ_1 was set to 10 in our experiments.

Also, the type-II loop iterations are not executed till $\tilde{\tau}$-optimality is achieved as this would require large number of type-II loop iterations. Therefore we terminate the type-II loop when τ_v-optimality is achieved, where $\tau_v = \frac{\hat{\psi}_{max}}{2}$ and $\hat{\psi}_{max}$ is the maximum violation when the algorithm enters the type-II loop. On some data sets, the dual objective function changes very slowly, resulting in a large number of type-II loop iterations that is disproportional to the change in the dual objective function value. Therefore we limit the number of type-II loop iterations to κ_2, thereby maintaining a balance between the number of type-I and type-II loop iterations. In our experiments, κ_2 was set to 5.

To concentrate the algorithm's effort on those $\boldsymbol{\alpha}_i(\mathbf{y})$ which are non-zero, we remove all those $\mathbf{y}$'s in the set V_i for which the associated $\boldsymbol{\alpha}_i(\mathbf{y})$ are zero. Some times, towards the end of the algorithm, the relative change in the dual objective function is small. In such cases, it may be a good idea to introduce an additional check in step 28 of Algorithm 3 which ensures that the type-I loop terminates even if $\hat{\psi}_{max} \leq \tau$.

5 Experiments

To demonstrate the efficacy of the proposed method for learning structural SVMs, we considered the problem

Table 1: **Summary of data sets**. n and n_{test} denote the sizes of the training and test data respectively, d is the input dimension, k denotes the number of alphabets and N is the feature vector dimension

Data set	n	n_{test}	d	k	N
POS	7200	1681	404990	42	17011344
WSJPOS	35531	1681	446180	42	18741324
BioNLP	18546	3856	513932	11	5653373
BioCreative	7500	5000	102409	3	307236
CoNLL	8936	2012	1679679	22	36953422

of sequence learning. Five data sets, Part-of-Speech (POS) [18], Wall Street Journal POS (WSJPOS) [16], BioNLP [14], BioCreative [6] and CoNLL [21], were used in our experiments. A summary of these data sets is given in Table 1. The $\arg\max_{\mathbf{y}} F_i(\mathbf{y})$ in Algorithm 3 was computed using the Viterbi algorithm. All experiments were run on a 1.81 GHz AMD processor with 10GB of shared main memory under Linux.

In the rest of this section, we first describe the feature functions used in our experiments. Comparison of SDM with the cutting-plane method and the SGD method will then be presented.

Feature Functions

In a sequence learning problem, each input, $\mathbf{x} = (x^1, x^2, \ldots, x^L)^T$ is a sequence of feature vectors and $\mathbf{y} = (y^1, y^2, \ldots, y^L)^T$ is a sequence of labels, where $x^t \in \mathbb{R}^d$ and $y^t \in \{1, 2, \ldots, k\}$. At each node t of the sequence, a bunch of feature functions $f(x^t, y^t)$ is formed. These are then combined to get the joint feature vector

$$f(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{L} f(x^t, y^t).$$

For our experiments, we used the combination of first order and token-independent second order functions to construct the joint feature vector [13]. In the case of first order functions the feature vector x^t is stacked into the position y^t (similar to the plain multi-class case) yielding

$$f_{multi}(x^t, y^t) = \begin{pmatrix} 0 \\ \vdots \\ x^t \\ \vdots \\ 0 \end{pmatrix}.$$

Second order token independent functions use the dependencies between y^t and y^{t-1} and result in the feature vector

$$f_{so}(y^t, y^{t-1}) = \begin{pmatrix} I(y^t = 1)I(y^{t-1} = 1) \\ I(y^t = 1)I(y^{t-1} = 2) \\ \vdots \\ I(y^t = k)I(y^{t-1} = k) \end{pmatrix}.$$

A dummy start node y^0 (which takes a fixed value) can be introduced so that $f_{so}(y^1, y^0)$ is defined. The joint feature vector can be written by combining these two feature vectors:

$$f(\mathbf{x}, \mathbf{y}) = \sum_{t=1}^{L} \begin{pmatrix} f_{multi}(x^t, y^t) \\ f_{so}(y^t, y^{t-1}) \end{pmatrix}.$$

The dimension N of this feature vector is $k^2 + dk$. Hamming distance was used as the loss function. Joachims et al [10] used the same set of features and loss function in their experiments on sequence learning.

Comparison with the Cutting Plane Method

Experiment setup We first compared SDM with the cutting-plane method on the five data sets mentioned above. For comparison we used the "1-slack" implementation available at [11]. The regularization parameter C in (OP1) was scaled so that the objective functions in (OP1) and (3.1) are same. The speed of convergence of the two methods was compared with respect to the relative function value difference, $\frac{\eta - \eta^*}{\eta^*}$ where η^* is the optimal dual objective function value given by the cutting-plane method and η denotes the dual objective function value attained by a method at any point of time. While comparing SDM with the cutting-plane method, SDM was terminated when it attained the dual objective function value of η^*. The two methods were also compared in terms of their ability to reach steady state test set performance fast. The regularization parameter C was set to 0.1 in our experiments.

Relative dual objective function difference and generalization performance Figure 4 (see the last page of the paper) gives plots of relative function value difference and test set accuracy as a function of training time for the five data sets. It can be seen from this figure that SDM is much faster than the cutting-plane method. On most of the data sets it is faster by an order of magnitude. Further, SDM also achieves the steady state test set performance faster.

Effect of the regularization parameter on the training time The two methods were also compared for different values of the regularization parameter C (0.1, 1 and 10) on two data sets, BioNLP and WSJPOS. The results are shown in Figures 2 and 3. It is clear from

Table 2: **Training time comparison of SDM and the Cutting-Plane (CP) method on different data sets**

Data set	Train time(sec) $C = 0.1$		Train time(sec) $C = 1$	
	SDM	*CP*	*SDM*	*CP*
WSJPOS	285	2771	710	4616
BioNLP	77	696	188	999
CoNLL	55	1015	110	1372
BioCreative	15	51	43	91
POS	77	844	160	1367

Table 3: **Training time comparison of SDM and SGD to achieve the same test set performance**

Data set	Train Time(sec)	
	SDM $C = 0.1$	SGD
WSJPOS	207	7656
BioNLP	67	700
CoNLL	65	865
BioCreative	31	79
POS	65	1928

these figures that SDM reaches the optimal objective function value and steady state test set performance faster than the cutting-plane method even at larger values of C.

Training time comparison The training times required by both the methods, to reach the optimal objective function value attained by the cutting plane method, are given in Table 2. SDM was observed to be an order of magnitude faster than the cutting-plane method on most of the data sets.

Comparison with the SGD Method

Experiment setup Stochastic gradient descent method is known to achieve steady state test set performance fast. In our experiments we used the SGD implementation available for CRFs at [3]. The default parameters in the implementation were used. For SDM C was set to the default value of 0.1. Since the two methods optimize different objective functions we did not compare them.

Generalization performance comparison Figure 1 compares SDM and SGD on the five data sets in terms of test set performance. This implementation of SGD does parameter tuning before doing online learning. During this time period, the test set performance is not evaluated and therefore, the plots shown start at a later point of time. From this figure it is clear that SDM achieves steady state test set performance faster than the SGD method on all the data sets.

Table 4: **Scalability of SDM on the POS data set**

Data set size	Train Time(sec)	
	$C = 0.1$	$C = 1$
450	12	14
900	18	28
1800	32	45
3600	53	92
7200	77	160

Training time comparison The SGD method was run for 50 (default) iterations and its training time and test set performance at the end of these iterations were noted. SDM was run until it attained the test set performance given by the SGD method. The training times of these methods are reported in Table 3. On most of the data sets, SDM was observed to be at least an order of magnitude faster than the SGD method.

Scalability

Next we examined the scalability of SDM. For this purpose, we used the POS data set and varied the training set size from 450 to 7200 as a multiple of 2. The training times (to reach τ-optimality) were measured as the size of the training set was increased. The results are reported in Table 4. From this table it is clear that SDM scales well under different parameter settings.

6 Conclusion

In this paper we proposed a sequential dual method for structural SVMs. It is based on the idea of sequentially looking at one example at a time and optimizing the dual variables associated with it. The proposed method is fast and easy to implement. Experiments on several benchmark data sets demonstrated that it is an order of magnitude faster than state-of-the-art approaches for structured learning. The proposed SDM also reaches steady state test set performance quite fast and scales well. Thus, it is a good alternative to the cutting-plane and SGD methods for structured output learning. In this work, SDM was evaluated on sequence learning problems. On the experiments conducted, it converged to a τ-optimal solution. More details related to the evaluation of SDM on other structured output problems and the convergence of this method will be reported in a future paper.

References

[1] B. E. Boser, I. Guyon and V. Vapnik. A training algorithm for optimal margin classifiers. COLT, 1992.

[2] A. Bordes, N. Usunier and L. Bottou, Sequence Labelling SVMs Trained in One Pass, ECML, 2008.

[3] L. Bottou, Stochastic gradient descent examples, `http://leon.bottou.org/projects/sgd`, 2007.

[4] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. Bartlett, Exponentiated Gradient Algorithms for

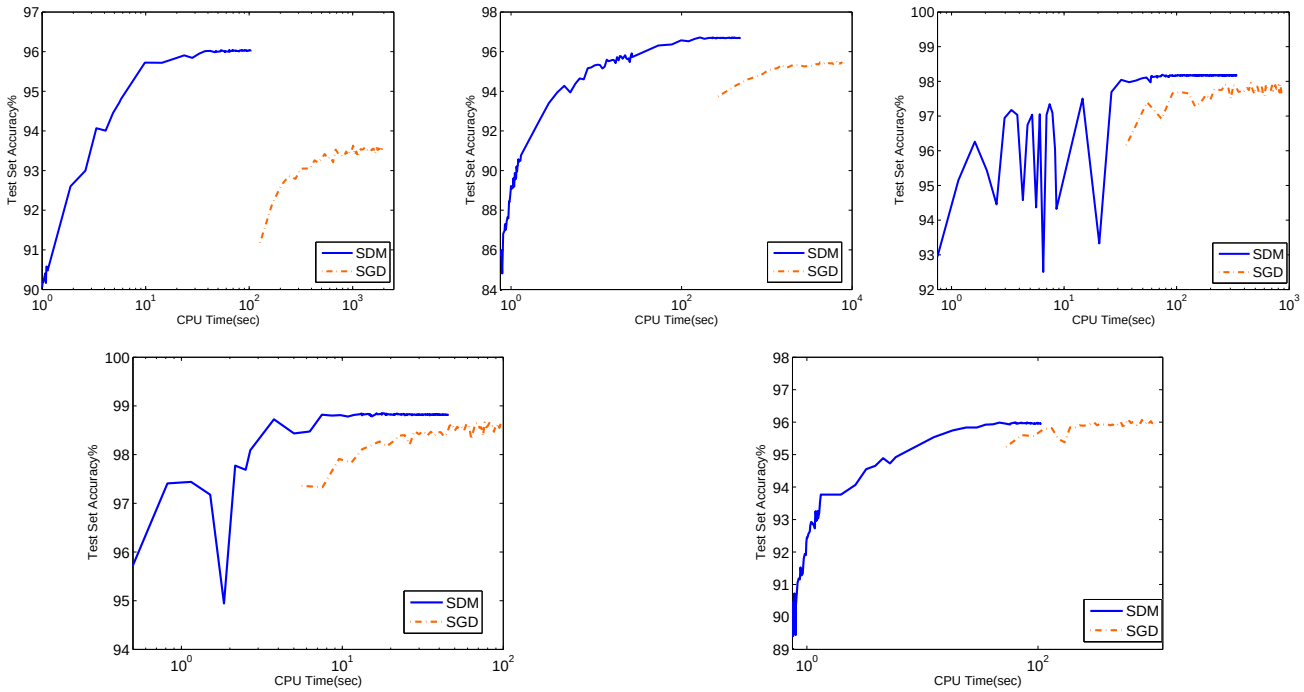

Figure 1: **Comparison of *SDM* and the *SGD* method on various data sets. Row 1: POS, WSJPOS and BioNLP data sets, Row 2: BioCreative and CoNLL data sets.**

Conditional Random Fields and Max-Margin Markov Networks, JMLR, 9:1775-1822, 2008.

[5] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel based vector machines. JMLR, 2:265-292, 2001.

[6] L. Hirschman, A. Yeh, C. Blashcke, A. Valencia, Overview of BioCreAtivE:critical assessment of information extraction for Biology, BMC Informatics, 6(Suppl 1), 2005.

[7] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi and S. Sundararajan. A dual coordinate descent method for large scale linear SVM. ICML, 2008.

[8] T. Joachims. Learning to align sequences: A maximum-margin approach. On-line manuscript.

[9] T. Joachims. Training linear SVMs in linear time. ACM KDD, 2006.

[10] T. Joachims, T. Finley, C.-N. J. Yu, Cutting Plane Training of Structural SVMs, Machine Learning, 77:27-59, 2009.

[11] T. Joachims. SVM^{struct}: Support vector machine for complex output. Implementation available at `http://www.cs.cornell.edu/People/tj/svm_light/svm_struct.html`

[12] S. S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, C.-J. Lin, A Sequential Dual Method for Large Scale Multi-class Linear SVMs, KDD, 2008.

[13] S. S. Keerthi and S. Sundararajan. CRF versus SVM-struct for Sequence Labeling. On-line manuscript.

[14] J.-D. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, N. Collier, Introduction to the bio-entity recognition task at JNLPBA, IJWNLPBA, 2004.

[15] J. Lafferty, A. McCallum, F. Pereira, Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, ICML, 2001.

[16] M. Marcus, B. Santorini, M. A. Marcinkiewicz, Building a Large Annotated Corpus of English, Computational Linguistics, 1993.

[17] R. Memisevic. Dual optimization of conditional probability models. On-line manuscript.

[18] N. Nguyen, Y. Guo, Comparisons of Sequence-labeling Algorithms and Extensions, ICML, 2007.

[19] J. Platt, Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola (Eds.), Advances in kernel methods - support vector learning. Cambridge: MIT Press. Chap 12.

[20] N. Ratliff, J. A. Bagnell and M. Zinkevich, (Online) Subgradient methods for structured prediction, AISTATS, 2007.

[21] E. F. T. K. Sang and S. Buchholz, Introduction to the CoNLL-2000 shared task: Chunking, CoNLL, 2000.

[22] S. Shalev-Shwartz, Y. Singer and N. Srebro, PEGASOS: Primal Estimated sub-GrAdient SOlver for SVM, ICML, 2007.

[23] B. Taskar, C. Guestrin and D. Koller, Maximum-margin Markov networks, NIPS, 2003.

[24] B. Taskar, Learning Structured Prediction Models: A Large Margin Approach, Ph.D. Thesis, 2004.

[25] I. Tsochantaridis, T. Joachims, T. Hoffmann, Y. Altun, Large Margin Methods for Structured and Interdependent Output Variables, JMLR, 6:1453:1484, 2005.

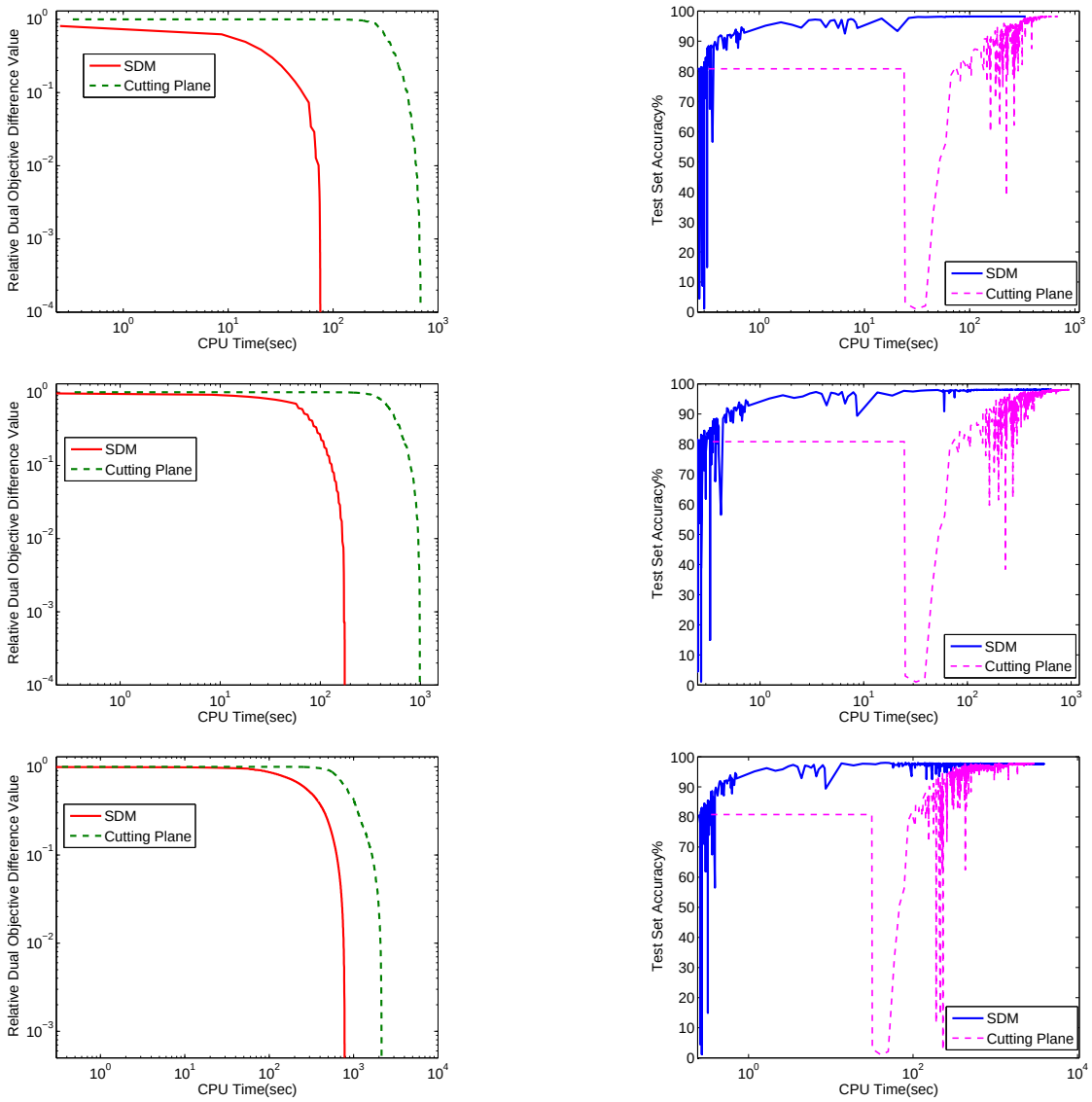

Figure 2: **Comparison of** *SDM* **and the** *Cutting plane* **method on the BioNLP data set for different** C **values. Row 1:** *C=0.1*, **Row 2:** *C=1.0*, **Row 3:** *C=10.0*.

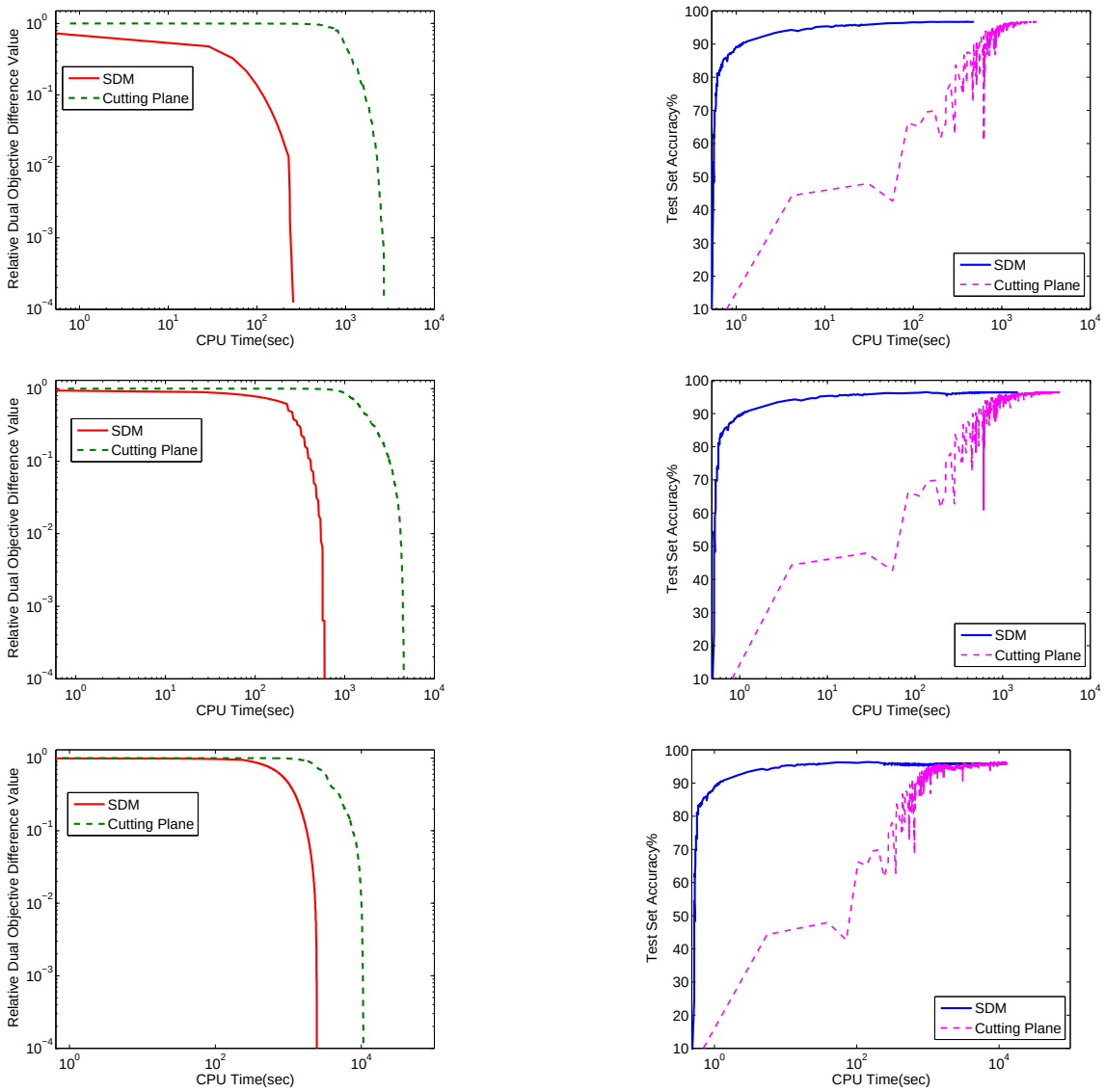

Figure 3: **Comparison of** *SDM* **and the** *Cutting plane* **method on the WSJPOS data set for different** C **values. Row 1:** *C=0.1*, **Row 2:** *C=1.0*, **Row 3:** *C=10.0*.

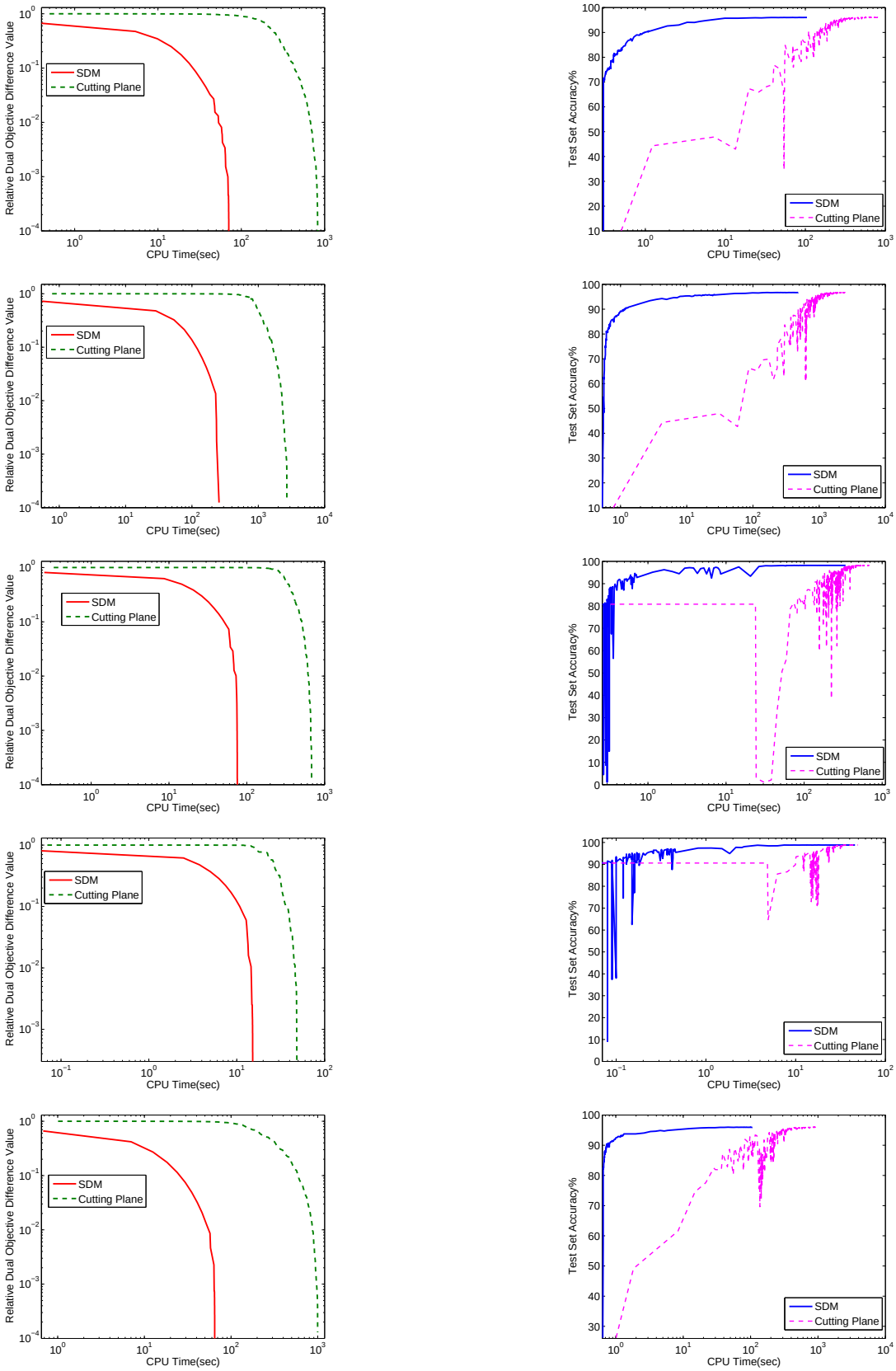

Figure 4: **Comparison of *SDM* and the *Cutting plane* method for C=0.1. The rows correspond to the data sets POS, WSJPOS, BioNLP, BioCreative and CoNLL in that order.**