Efficient Optimization of Performance Measures by Classifier Adaptation

Nan Li, Ivor W. Tsang, and Zhi-Hua Zhou, *Fellow*, *IEEE*

Abstract—In practical applications, machine learning algorithms are often needed to learn classifiers that optimize domain specific performance measures. Previously, the research has focused on learning the needed classifier in isolation, yet learning nonlinear classifier for nonlinear and nonsmooth performance measures is still hard. In this paper, rather than learning the needed classifier by optimizing specific performance measure directly, we circumvent this problem by proposing a novel two-step approach called CAPO, namely, to first train nonlinear auxiliary classifiers with existing learning methods and then to adapt auxiliary classifiers for specific performance measures. In the first step, auxiliary classifiers can be obtained efficiently by taking off-the-shelf learning algorithms. For the second step, we show that the classifier adaptation problem can be reduced to a quadratic program problem, which is similar to linear $\mathrm{SVM}^{\mathrm{perf}}$ and can be efficiently solved. By exploiting nonlinear auxiliary classifiers, CAPO can generate nonlinear classifier which optimizes a large variety of performance measures, including all the performance measures based on the contingency table and AUC, while keeping high computational efficiency. Empirical studies show that CAPO is effective and of high computational efficiency, and it is even more efficient than linear $\mathrm{SVM}^{\mathrm{perf}}$.

Index Terms—Optimize performance measures, classifier adaptation, ensemble learning, curriculum learning

1 Introduction

In real-world applications, different user requirements often employ different domain specific performance measures to evaluate the success of learning algorithms. For example, F1-score and Precision-Recall Breakeven Point (PRBEP) are usually employed in text classification; Precision and Recall are often used in information retrieval; Area Under the ROC Curve (AUC) and Mean Average Precision (MAP) are important to ranking. Ideally, to achieve good prediction performance, learning algorithms should train classifiers by optimizing the concerned performance measures. However, this is usually not easy due to the nonlinear and nonsmooth nature of many performance measures like F1-score and PRBEP.

During the past decade, many algorithms have been developed to optimize frequently used performance measures, and they have shown better performance than conventional methods [18], [6], [19], [15], [5], [4]. By now, the research has focused on training the needed classifier in isolation. But, in general, it is still challenging to design general-purpose learning algorithms to train nonlinear classifiers optimizing nonlinear and nonsmooth performance measures, though it is very needed in practice. For example, $\mathrm{SVM}^{\mathrm{perf}}$ proposed by Joachims [15] can efficiently optimize a large variety of performance measures in the linear case, but its nonlinear kernelized extension suffers from computational problems [31], [17].

In this paper, rather than directly designing sophisticated algorithms to optimize specific performance measures, we take a different strategy and present a novel two-step approach called CAPO to cope with this problem. Specifically, we first train auxiliary classifiers by exploiting existing off-the-shelf learning algorithms, and then adapt the obtained auxiliary classifiers to optimize the concerned performance measure. Note that in the literature many algorithms have been proposed that can train the auxiliary classifiers quite efficiently, even on large-scale data; thus the first step can be easily performed. For the second step, to make use of the auxiliary classifiers, we consider the classifier adaptation problem under the function-level adaptation framework [29], and formulate it as a quadratic program problem which is similar to linear $\mathrm{SVM}^{\mathrm{perf}}$ [15] and can also be efficiently solved. Hence, in total, CAPO can work efficiently.

A prominent advantage of CAPO is that it is a flexible framework which can handle different types of auxiliary classifiers and a large variety of performance measures, including all the performance measures based on the contingency table and AUC. By exploiting nonlinear auxiliary classifiers, CAPO can train nonlinear classifiers optimizing the concerned performance measure with low computational cost. This is very helpful because nonlinear classifiers are preferred in many real-world applications but training such a nonlinear classifier is often of high computational cost (e.g., nonlinear kernelized $\mathrm{SVM}^{\mathrm{perf}}$). In empirical studies, we perform experiments on datasets from different domains. It is found that CAPO is more effective and more

- *N. Li is with the National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China, and the School of Mathematical Sciences, Soochow University, Suzhou 215006, China. E-mail: lin@lamda.nju.edu.cn.*
- *I.W. Tsang is with the School of Computer Engineering, Nanyang Technological University, Singapore 639798. E-mail: IvorTsang@ntu.edu.sg.*
- *Z.-H. Zhou is with the National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: zhouzh@lamda.nju.edu.cn.*

Manuscript received 4 Jan. 2012; revised 7 July 2012; accepted 20 July 2012; published online 2 Aug. 2012.
Recommended for acceptance by A. Gretton.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2012-01-0012.
Digital Object Identifier no. 10.1109/TPAMI.2012.172.

efficient than state-of-the-art methods; also, it scales well with respect to training data size and is robust with the parameters. It is worth mentioning that the classifier adaptation procedure of CAPO is even more efficient than linear SVM^{perf}, though it employs the same cutting-plane algorithm to solve the classifier adaptation problem.

The rest of this paper is organized as follows: Section 2 briefly describes some background, including the problem studied here and SVM^{perf}. Section 3 presents our proposed CAPO approach. Section 4 gives some discussions on related work. Section 5 reports on our empirical studies, followed by the conclusion in Section 6.

2 Optimizing Performance Measures

In this section, we first present the problem of optimizing performance measures, and then introduce SVM^{perf} [15] and its kernelized extension.

2.1 Preliminaries and Background

In machine learning tasks, given a set of n training examples $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \{-1, +1\}$ are input pattern and its class label, respectively, our goal is to learn a classifier $f(\mathbf{x})$ that minimizes the expected risk on new data sample $S = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_m, y'_m)\}$, i.e.,

$$R^{\Delta}(f) = \mathbb{E}_S[\Delta((y'_1, \ldots, y'_m), (f(\mathbf{x}'_1), \ldots, f(\mathbf{x}'_m)))],$$

where $\Delta((y'_1, \ldots, y'_m), (f(\mathbf{x}'_1), \ldots, f(\mathbf{x}'_m)))$ is the loss function which quantifies the loss of f on S. Subsequently, we use the notation $\Delta(f; S)$ to denote $\Delta((y'_1, \ldots, y'_m), (f(\mathbf{x}'_1), \ldots, f(\mathbf{x}'_m)))$ for convenience. Since it is intractable to compute the expectation $\mathbb{E}_S[\cdot]$, discriminative learning methods usually approximate the expected risk $R^{\Delta}(f)$ using the empirical risk

$$\hat{R}^{\Delta}_D(f) = \Delta(f; D),$$

which measures $f(\mathbf{x})$'s loss on the training data D, and then train classifiers by minimizing empirical risk or regularized risk. In practice, domain specific performance measures are usually employed to evaluate the success of learned classifiers. Thus, good performance can be expected if the classifiers are trained by directly optimizing the concerned performance measures. Here, we are interested in regarding the loss function Δ as practical performance measures (e.g., F1-score and PRBEP) instead of some kinds of surrogate functions (e.g., hinge loss and exponential loss). In this situation, the loss function Δ can be nonlinear and nonsmooth function of training examples in D; thus it is computationally challenging to optimize the empirical risk Δ in practice.

In the literature, some methods have been developed to optimize frequently used performance measures, such as AUC [12], [14], F1-score [24], NDCG, and MAP [32], [28], [27]. Among existing methods that try to optimize performance measures directly, the SVM^{perf} proposed by Joachims [15] is a representative example. One of its attractive advantages is that by employing the multivariate prediction framework, it can directly handle a large variety of performance measures, including AUC and all measures that can be computed from the contingency table, while most of other methods are specially designed for one specific performance measure. Subsequently, we describe it and also show its limitation.

2.2 SVM^{perf} and Its Kernelized Extension

Since many performance measures cannot be decomposed over individual predictions, SVM^{perf}[15] takes a multivariate prediction formulation and considers mapping a tuple of n patterns $\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to a tuple of n class labels $\bar{y} = (y_1, \ldots, y_n)$ by

$$\bar{f} : \mathcal{X}^n \mapsto \mathcal{Y}^n,$$

where $\mathcal{Y}^n \subseteq \{-1, +1\}^n$ is the set of all admissible label vectors. To implement this mapping, it exploits a discriminant function and makes prediction as

$$\bar{f}(\bar{\mathbf{x}}) = \arg\max_{\bar{y}' \in \mathcal{Y}^n} \mathbf{w}^\top \Psi(\bar{\mathbf{x}}, \bar{y}'), \tag{1}$$

where $\mathbf{w}$ is a parameter vector and $\Psi(\bar{\mathbf{x}}, \bar{y}')$ is a feature vector relating $\bar{\mathbf{x}}$ and $\bar{y}'$. Obviously, the computational efficiency of the inference (1) highly depends on the form of the feature vector $\Psi(\bar{\mathbf{x}}, \bar{y}')$.

2.2.1 Linear Case

In [15], the feature vector $\Psi(\bar{\mathbf{x}}, \bar{y}')$ is restricted to being

$$\Psi(\bar{\mathbf{x}}, \bar{y}') = \sum_{i=1}^{n} y'_i \mathbf{x}_i,$$

and thus the argmax in (1) can be achieved by assigning y'_i to $\text{sign}(\mathbf{w}^\top \mathbf{x}_i)$, leading to a linear classifier

$$f(\mathbf{x}) = \text{sign}[\mathbf{w}^\top \mathbf{x}].$$

To learn the parameter $\mathbf{w}$, the following optimization problem is formulated:

$$\begin{array}{ll} \min\limits_{\mathbf{w}, \xi \geq 0} & \frac{1}{2}\|\mathbf{w}\|^2 + C\xi \\ \text{s.t.} & \forall \bar{y}' \in \mathcal{Y}^n \setminus \bar{y}: \\ & \mathbf{w}^\top [\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')] \geq \Delta(\bar{y}, \bar{y}') - \xi, \end{array} \tag{2}$$

where $\Delta(\bar{y}, \bar{y}')$ is the loss of mapping $\bar{\mathbf{x}}$ to $\bar{y}'$ while its true label vector is $\bar{y}$. It is not hard to find that $\Delta(\bar{y}, \bar{y}')$ can incorporate many types of performance measures, and (2) optimizes an upper bound of the empirical risk [15].

While there are a huge number of constraints in (2), the cutting-plane algorithm in Algorithm 1 can be used to solve it, and this algorithm has been shown to need at most $O(1/\epsilon)$ iterations to converge to an ϵ-accurate solution [15], [16]. In each iteration, it needs to find the most violated constraint by solving

$$\arg\max_{\bar{y}' \in \mathcal{Y}^n} \{\Delta(\bar{y}, \bar{y}') + \mathbf{w}^\top \Psi(\bar{\mathbf{x}}, \bar{y}')\}. \tag{3}$$

It has been shown that if the discriminant function $\mathbf{w}^\top \Psi(\bar{\mathbf{x}}, \bar{y}')$ can be written in the form $\sum_{i=1}^{n} y'_i f(\mathbf{x}_i)$, the inference (3) can be solved for many performance measures in polynomial time, that is, $O(n^2)$ for contingency table-based performance measures (such as F1-score) and $O(n \log n)$ for AUC [15]. Hence, Algorithm 1 can train SVM^{perf} in polynomial time.

Algorithm 1. Cutting-plane algorithm for training linear SVM^{perf}[15]

1: Input: $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, C, ϵ
2: $\mathcal{W} \leftarrow \emptyset$
3: **repeat**
4: $(\mathbf{w}, \xi) \leftarrow \arg\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\|\mathbf{w}\|^2 + C\xi$
s.t. $\forall\ \bar{y}' \in \mathcal{W}$:
$\mathbf{w}^\top[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')] \geq \Delta(\bar{y}, \bar{y}') - \xi$,
5: find the most violated constraint by
$\bar{y}' \leftarrow \arg\max_{\bar{y}''\in\mathcal{Y}^n}\{\Delta(\bar{y}, \bar{y}'') + \mathbf{w}^\top\Psi(\bar{\mathbf{x}}, \bar{y}'')\}$
6: $\mathcal{W} \leftarrow \mathcal{W} \cup \{\bar{y}'\}$
7: **until** $\Delta(\bar{y}, \bar{y}') - \mathbf{w}^\top[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')] \leq \xi + \epsilon$

2.2.2 Kernelized Extension

Using kernel trick, the linear SVM^{perf} described above can be extended to the nonlinear case [16]. It is easy to obtain the dual of (2) as

$$\begin{aligned} \max_{\boldsymbol{\alpha}\geq 0}\ & -\frac{1}{2}\boldsymbol{\alpha}^\top\mathbf{H}\boldsymbol{\alpha} + \sum_{\bar{y}'\in\mathcal{Y}^n}\alpha_{\bar{y}'}\Delta(\bar{y}, \bar{y}') \\ \text{s.t.}\ & \sum_{\bar{y}'\in\mathcal{Y}^n}\alpha_{\bar{y}'} = C, \end{aligned} \tag{4}$$

where $\boldsymbol{\alpha}$ is the column vector of $\alpha_{\bar{y}'}$s and $\mathbf{H}$ is the Gram matrix with the entry $\mathbf{H}(\bar{y}', \bar{y}'')$ equal to

$$[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')]^\top[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}'')].$$

By replacing the primal problem with its dual in Line 4, it is easy to get the dual variant of Algorithm 1, which can solve the problem (4) in at most $O(1/\epsilon)$ iterations [16], [17]. In the solution, each $\alpha_{\bar{y}'}$ corresponds to a constraint in $\mathcal{W}$, and the discriminant function $\mathbf{w}^\top\Psi(\bar{\mathbf{x}}, \bar{y}')$ in (1) can be written as

$$\sum_{\bar{y}''\in\mathcal{W}}\alpha_{\bar{y}''}[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}'')]^\top\Psi(\bar{\mathbf{x}}, \bar{y}').$$

Obviously, the inner product $\Psi(\bar{\mathbf{x}}, \bar{y}')^\top\Psi(\bar{\mathbf{x}}, \bar{y}'')$ can be computed via a kernel $K(\bar{\mathbf{x}}, \bar{y}', \bar{\mathbf{x}}, \bar{y}'')$. However, if so, it can be found that the argmax in (1) and (3) will become computationally intractable. Hence, feature vectors of the following form are used:

$$\Psi(\bar{\mathbf{x}}, \bar{y}') = \sum_{i=1}^n y_i'\Phi(\mathbf{x}_i),$$

where $\Phi(\mathbf{x}_i)^\top\Phi(\mathbf{x}_j)$ can be computed via a kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top\Phi(\mathbf{x}_j)$. Then, the discriminant function becomes

$$\mathbf{w}^\top\Psi(\bar{\mathbf{x}}, \bar{y}') = \sum_{i=1}^n y_i'\sum_{j=1}^n\beta_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{5}$$

where $\beta_j = \sum_{\bar{y}''\in\mathcal{W}}\alpha_{\bar{y}''}(y_j - y_j'')$. In this case, the argmax in (1) can be achieved by assigning each y_i' with

$$\text{sign}\left[\sum_{j=1}^n\beta_j K(\mathbf{x}_i, \mathbf{x}_j)\right],$$

which produces the kernelized classifier

$$f(\mathbf{x}) = \text{sign}\left[\sum_{i=1}^n\beta_i K(\mathbf{x}, \mathbf{x}_i)\right].$$

However, in each iteration, the Gram matrix $\mathbf{H}$ needs to be updated by adding a new row/column for the new constraint. Suppose $\bar{y}^+$ is added; for every $\bar{y}' \in \mathcal{W}$, it requires computing $\mathbf{H}(\bar{y}', \bar{y}^+)$ as

$$\sum_{i=1}^n\sum_{j=1}^n(y_i - y_i')(y_j - y_j^+)K(\mathbf{x}_i, \mathbf{x}_j).$$

Thus, let m denote the number of constraints in $\mathcal{W}$ and n denote the data size; it takes $O(mn^2)$ kernel evaluations in each iteration. Also, it should be noted that computing the discriminative function (5) also requires $O(n^2)$ kernel evaluations, and this adds to the computational cost of the inference (3). These issues cause the kernelized extension of SVM^{perf} to suffer from computational problems, even on a reasonably sized dataset. However, as we know, nonlinear classifiers are quite necessary in many practical applications. Hence, training a nonlinear classifier that optimizes a specific performance measure becomes central to this work.

3 Classifier Adaptation for Performance Measures

In this section, we introduce our proposed approach CAPO, which is short for Classifier Adaptation for Performance measures Optimization.

3.1 Motivation and Basic Idea

Notice the fact that it is generally not straightforward to design learning algorithms which optimize specific performance measures, while there have been many well-developed learning algorithms in the literature and some of them can train complex nonlinear classifiers quite efficiently. Our intuitive motivation of this work is to exploit these existing algorithms to help in training the needed classifier that optimizes the concerned performance measure.

Specifically, denote $f^*(\mathbf{x})$ as the ideal classifier which minimizes the empirical risk $\Delta(f; D)$; it is generally not easy to design algorithms which can efficiently find $f^*(\mathbf{x})$ in the function space by minimizing $\Delta(f; D)$ due to its nonlinear and nonsmooth nature, especially when we are interested in complex nonlinear classifiers. Meanwhile, by using many off-the-shelf learning algorithms, we can get a certain classifier $f'(\mathbf{x})$ quite efficiently, even on a large-scale dataset. Obviously, $f'(\mathbf{x})$ can differ from the ideal classifier $f^*(\mathbf{x})$ since it may optimize a different loss from $\Delta(f; D)$. However, since many performance measures are closely related, for example, both F1-score and PRBEP are functions of precision and recall, the average AUC is an increasing function of accuracy [8], $f'(\mathbf{x})$ can be regarded as a rough estimated classifier of $f^*(\mathbf{x})$, then we conjecture that $f'(\mathbf{x})$ will be helpful to finding $f^*(\mathbf{x})$ in the function space, for example, it can reduce the computational cost of searching the whole function space. Subsequently, $f'(\mathbf{x})$ is called the auxiliary classifier and $f^*(\mathbf{x})$ the target classifier.

To implement this motivation, we take classifier adaptation techniques [21], [30] which have achieved success

in domain adaptation [9]. Specifically, after getting the auxiliary classifier $f'(\mathbf{x})$, we adapt it to a new classifier $f(\mathbf{x})$ and it is expected that the adapted classifier $f(\mathbf{x})$ can achieve good performance in terms of the performance measure concerned. For the classifier adaptation procedure, it is expected that

- the adapted classifier outperforms the auxiliary classifier in terms of concerned performance measure;
- the adaptation procedure is more efficient than directly training a new classifier for concerned performance measure;
- the adaptation framework can handle different types of auxiliary classifiers and different performance measures.

Since many existing algorithms can train auxiliary classifiers efficiently, we focus on the classifier adaptation procedure in the remainder of the paper.

3.2 Classifier Adaptation Procedure

For the aim of this work, we study the classifier adaptation problem under the function-level adaptation framework, which was originally proposed for domain adaption in [30], [29].

3.2.1 Single Auxiliary Classifier

The basic idea is to directly modify the decision function of auxiliary classifier which can be of any type. Concretely, given one auxiliary classifier $f'(\mathbf{x})$, we construct the new classifier $f(\mathbf{x})$ by adding a delta function $f_\delta(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x})$, i.e.,

$$f(\mathbf{x}) = \operatorname{sign}\big[f'(\mathbf{x}) + \mathbf{w}^\top \Phi(\mathbf{x})\big],$$

where $\mathbf{w}$ is the parameter of $f_\delta(\mathbf{x})$ and $\Phi(\cdot)$ is a feature mapping. It should be noted that $f'(\mathbf{x})$ is the auxiliary classifier directly producing $+1/-1$ predictions, and it can be of any type (e.g., SVM, neural network (NN), decision tree (DT), etc.) because it is treated as a "black-box" in CAPO, while $f_\delta(\mathbf{x})$ is a real-valued function which is added to modify the decision of $f'(\mathbf{x})$ such that $f(\mathbf{x})$ can achieve good performance in terms of our concerned performance measure. Obviously, our task is reduced to learning the delta function $f_\delta(\mathbf{x})$, and hence the classifier $f(\mathbf{x})$.

Based on the principle of regularized risk minimization, the following problem should be considered:

$$\min_{\mathbf{w}} \Omega(\mathbf{w}) + C \cdot \Delta(\bar{y}, \bar{y}^*), \tag{6}$$

where $\Omega(\mathbf{w})$ is a regularization term, $\Delta(\bar{y}, \bar{y}')$ is the empirical risk on training data D with $\bar{y} = (y_1, \ldots, y_n)$ are the true class labels, and $\bar{y}^* = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))$ are the predictions of $f(\mathbf{x})$, and C is the regularization parameter. In practice, the problem (6) is not easy to solve, mainly due to the following two issues:

1. For some multivariate performance measures like F1-score, Δ cannot be decomposed over individual predictions, i.e., they cannot be written in the form of $\Delta(\bar{y}, \bar{y}') = \sum_{i=1}^n \ell(y_i, h(\mathbf{x}_i))$.
2. The empirical risk Δ can be nonconvex and nonsmooth.

To cope with these issues, inspired by SVM^{perf}[15], we take the multivariate prediction formulation. That is, instead of learning $f(\mathbf{x}) : \mathcal{X} \mapsto \mathcal{Y}$ directly, we consider $\bar{f} : \mathcal{X}^n \mapsto \mathcal{Y}^n$, which maps a tuple of n patterns $\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to n class labels $\bar{y} = (y_1, \ldots, y_n)$. Specifically, the mapping is implemented by maximizing a discriminant function $F(\bar{\mathbf{x}}, \bar{y})$, i.e.,

$$\bar{y} = \arg\max_{\bar{y}' \in \mathcal{Y}^n} F(\bar{\mathbf{x}}, \bar{y}'). \tag{7}$$

In this work, $F(\bar{\mathbf{x}}, \bar{y}) = \sum_{i=1}^n y_i f(\mathbf{x}_i)$ is used, so the argmax in (7) can be easily obtained by assigning y'_i with $f(\mathbf{x})$. In this way, (7) becomes

$$\bar{y} = \arg\max_{\bar{y}' \in \mathcal{Y}^n} \begin{bmatrix} 1 \\ \mathbf{w} \end{bmatrix}^\top \Upsilon(\bar{\mathbf{x}}, \bar{y}),$$

where

$$\Upsilon(\bar{\mathbf{x}}, \bar{y}) = \sum_{i=1}^n y_i \begin{bmatrix} f'(\mathbf{x}_i) \\ \Phi(\mathbf{x}_i) \end{bmatrix}.$$

Instead of directly minimizing $\Delta(\bar{y}, \bar{y}')$, we consider its convex upper bound as follows.

Proposition 1. *Given training data D and the discriminative function $F(\mathbf{x}, \bar{y})$, the risk function*

$$R(\mathbf{w}; D) = \max_{\bar{y}' \in \mathcal{Y}^n} [F(\bar{\mathbf{x}}, \bar{y}') - F(\bar{\mathbf{x}}, \bar{y}) + \Delta(\bar{y}, \bar{y}')] \tag{8}$$

is a convex upper bound of the empirical risk $\Delta(\bar{y}, \bar{y}^*)$ with $\bar{y}^* = \arg\max_{\bar{y}' \in \mathcal{Y}^n} F(\bar{\mathbf{x}}, \bar{y})$.

Proof. The convexity of (8) with respect to $\mathbf{w}$ is due to the fact that F is linear in $\mathbf{w}$ and a maximum of linear functions is convex. Since $\bar{y}^* = \arg\max_{\bar{y}' \in \mathcal{Y}^n} F(\bar{\mathbf{x}}, \bar{y})$, it follows that

$$R(\mathbf{w}; D) \geq F(\mathbf{x}, \bar{y}^*) - F(\mathbf{x}, \bar{y}) + \Delta(\mathbf{y}, \bar{y}^*) \geq \Delta(\mathbf{y}, \bar{y}^*).$$

Thus, $R(\mathbf{w}; D)$ is a convex upper bound. □

Consequently, by taking $\Omega(\mathbf{w}) = \|\mathbf{w}\|^2$ and the convex upper bound $R(\mathbf{w}; D)$, the problem (6) becomes

$$\begin{array}{ll} \min\limits_{\mathbf{w}, \xi \geq 0} & \frac{1}{2}\|\mathbf{w}\|^2 + C\xi \\ \text{s.t.} & \forall \bar{y}' \in \mathcal{Y}^n \setminus \bar{y}: \\ & \begin{bmatrix} 1 \\ \mathbf{w} \end{bmatrix}^\top [\Upsilon(\bar{\mathbf{x}}, \bar{y}) - \Upsilon(\bar{\mathbf{x}}, \bar{y}')] \geq \Delta(\bar{y}, \bar{y}') - \xi, \end{array} \tag{9}$$

where ξ is a slack variable introduced to hide the max in (8). Although the regularization term $\|\mathbf{w}\|^2$ has the same form as that of SVM^{perf} in (2), it has a different meaning, as stated in following proposition.

Proposition 2. *By minimizing the regularization term $\|\mathbf{w}\|^2$ in the problem (9), the adapted classifier $f(\mathbf{x})$ is made to be near the auxiliary classifier $f'(\mathbf{x})$ in reproducing kernel Hilbert space.*

Proof. The Lagrangian function of (9) is

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + \left(C - \gamma - \sum_{\bar{y}' \in \mathcal{Y}^n} \alpha_{\bar{y}'}\right)\xi - \sum_{\bar{y}' \in \mathcal{Y}^n} \alpha_{\bar{y}'} \left(\begin{bmatrix} 1 \\ \mathbf{w} \end{bmatrix}^\top [\Upsilon(\bar{\mathbf{x}}, \bar{y}) - \Upsilon(\bar{\mathbf{x}}, \bar{y}')] - \Delta(\bar{y}, \bar{y}') \right),$$

where $\alpha_{\bar{y}}$ and γ are Lagrangian multipliers. By setting the derivative of L w.r.t. $\mathbf{w}$ to zero, we obtain

$$\mathbf{w} = \sum_{i=1}^{n} \beta_i \Phi(\mathbf{x}_i) \text{ and } f_\delta(\mathbf{x}) = \sum_{i=1}^{n} \beta_i K(\mathbf{x}_i, \mathbf{x}),$$

where $\beta_i = \sum_{\bar{y}' \in \mathcal{Y}^n} \alpha_{\bar{y}'}(y_i - y'_i)$ and $K(\mathbf{x}_i, \mathbf{x}) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x})$. Since $f(\mathbf{x}) = f'(\mathbf{x}) + f_\delta(\mathbf{x})$, the distance between f and f' in RKHS is $\|f - f'\|^2 = \|f_\delta\|^2$, which is computed as $\langle f_\delta, f_\delta \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_i \beta_j K(\mathbf{x}_i, \mathbf{x}_j)$. Meanwhile, since $\mathbf{w} = \sum_{i=1}^{n} \beta_i \Phi(\mathbf{x}_i)$, we have

$$\|\mathbf{w}\|^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_i \beta_i \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_i).$$

By computing $\Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_i)$ via $K(\mathbf{x}_i, \mathbf{x}_j)$, we obtain $\|f - f'\|^2 = \|\mathbf{w}\|^2$, which completes the proof. □

In summary, by solving the problem (9), CAPO finds the adapted classifier $f(\mathbf{x})$ near the auxiliary classifier $f'(\mathbf{x})$ such that $f(\mathbf{x})$ minimizes an upper bound of the empirical risk, and the parameter C balances these two goals.

3.2.2 Multiple Auxiliary Classifiers

If there are multiple auxiliary classifiers, rather than choosing one, we learn the target classifier by leveraging all the auxiliary classifiers. A straightforward idea is to construct an ensemble of them; then the ensemble is treated as a single classifier to be adapted. Suppose we have m auxiliary classifiers $f^1(\mathbf{x}), \ldots, f^m(\mathbf{x})$, the target classifier $f(\mathbf{x})$ can be formulated as

$$f(\mathbf{x}) = \text{sign}\left[\sum_{i=1}^{m} a_i f^i(\mathbf{x}) + \mathbf{w}^\top \Phi(\mathbf{x})\right], \tag{10}$$

where a_i is the weight of the auxiliary classifier $f^i(\mathbf{x})$, and $f_\delta(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x})$ is the delta function as above. We learn the ensemble weights $\mathbf{a} = [a_1, \ldots, a_m]^\top$ and the parameter $\mathbf{w}$ of $f_\delta(\mathbf{x})$ simultaneously. Let $\mathbf{f}_i = [f^1(\mathbf{x}_i), \ldots, f^m(\mathbf{x}_i)]^\top$ and

$$\Psi(\bar{\mathbf{x}}, \bar{y}) = \sum_{i=1}^{n} y_i \begin{bmatrix} \mathbf{f}_i \\ \Upsilon(\mathbf{x}_i) \end{bmatrix}.$$

Following the same strategy as above, the following problem is formulated:

$$\begin{array}{ll} \min\limits_{\mathbf{a},\mathbf{w},\xi \geq 0} & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2} B \|\mathbf{a}\|^2 + C\xi \\ \text{s.t.} & \forall\, \bar{y}' \in \mathcal{Y}^n \setminus \bar{y}: \\ & \begin{bmatrix} \mathbf{a} \\ \mathbf{w} \end{bmatrix}^\top [\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')] \geq \Delta(\bar{y}, \bar{y}') - \xi, \end{array} \tag{11}$$

where $\|\mathbf{a}\|^2$ penalizes large weights on the auxiliary classifiers. It prevents the target classifier $f(\mathbf{x})$ from having too much reliance on the auxiliary classifiers because they do not directly optimize the target performance measure. The term $\|\mathbf{w}\|^2$ measures the distance between $f(\mathbf{x})$ and $\sum_{i=1}^{m} a_i f^i(\mathbf{x})$ in the function space. Thus, minimizing $\frac{1}{2}\|\mathbf{w}\|^2$ finds the final classifier $f(\mathbf{x})$ near the ensemble of auxiliary classifiers $\sum_{i=1}^{m} a_i f^i(\mathbf{x})$ in the function space. The two goals are balanced by the parameter B. Hence, in summary, it learns an ensemble of auxiliary classifiers, and seeks the target classifier near the ensemble such that the risk in terms of concerned performance measure is minimized.

3.2.3 Efficient Learning via Feature Augmentation

Obviously, in CAPO the auxiliary classifier $f'(\mathbf{x})$ can be nonlinear classifiers such as SVM and neural network; thus the adapted classifier $f(\mathbf{x})$ is nonlinear even if the delta function $f_\delta(\mathbf{x})$ is linear. Empirical studies in Section 5 show that using linear delta function $f_\delta(\mathbf{x})$ achieves good performance while keeping computational efficiency.

Consider linear delta function, i.e., $\Phi(\mathbf{x}) = \mathbf{x}$ and $f_\delta(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$, and take CAPO with multiple auxiliary classifiers for example, if we augment the original features with outputs of auxiliary classifiers and let

$$\mathbf{v} = \begin{bmatrix} \sqrt{B}\mathbf{a} \\ \mathbf{w} \end{bmatrix} \text{ and } \mathbf{x}'_i = \begin{bmatrix} \frac{1}{\sqrt{B}} \mathbf{f}_i \\ \mathbf{x}_i \end{bmatrix}, \tag{12}$$

the adaptation problem (11) can be written as

$$\begin{array}{ll} \min\limits_{\mathbf{v},\xi \geq 0} & \frac{1}{2}\|\mathbf{v}\|^2 + C\xi \\ \text{s.t.} & \forall\, \bar{y}' \in \mathcal{Y}^n \setminus \bar{y}: \\ & \mathbf{v}^\top \left[\sum_{i=1}^{n} y_i \mathbf{x}'_i - \sum_{i=1}^{n} y'_i \mathbf{x}'_i\right] \geq \Delta(\bar{y}, \bar{y}') - \xi. \end{array} \tag{13}$$

For CAPO with one auxiliary classifier, it is easy to find that there exists a constant B such that the adaptation problem (9) can also be transformed into problem (13) if we define

$$\mathbf{v} = \begin{bmatrix} \sqrt{B} \\ \mathbf{w} \end{bmatrix} \text{ and } \mathbf{x}'_i = \begin{bmatrix} \frac{1}{\sqrt{B}} \mathbf{f}_i \\ \mathbf{x}_i \end{bmatrix}. \tag{14}$$

Note that the problem (13) is the same as that of linear SVM^{perf} in (2). Thus, after obtaining auxiliary classifiers, if we augment the original data features with the outputs of auxiliary classifiers according to (12) or (14), the classifier adaptation problem of CAPO can be efficiently solved by the cutting plane algorithm in Algorithm 1. Obviously, as does linear SVM^{perf}, CAPO can also handle all the performance measures based on the contingency table and AUC.

In practice, CAPO is an efficient approach for training nonlinear classifiers optimizing specific performance measures because both of its steps can be efficiently performed. Moreover, because auxiliary classifiers can be seen as estimation of the needed classifier, it can be expected that Algorithm 1 needs fewer iterations to converge, i.e., fewer times of solving the inference (3), and hence its classifier adaptation procedure can be more efficient than linear SVM^{perf}, which searches the function space directly. This has been validated by the experimental results in Section 5.2.

4 Discussion with Related Work

The most famous work that optimizes performance measures is SVM^{perf}[15]. By taking a multivariate prediction formulation, it finds the classifier in the function space directly. Our proposed CAPO works in a different manner and employs auxiliary classifiers to help find the target classifier in the function space. Furthermore, CAPO is a

framework that can use different types of auxiliary classifiers. If nonlinear auxiliary classifier is used, the obtained classifier will also be nonlinear. This is very helpful because nonlinear classifier is preferred in many applications, while training nonlinear SVM^{perf} is computationally expensive. In summary, compared with SVM^{perf}, CAPO can provide the needed nonlinearity while keeping and even improving computational efficiency.

Another related work is A-SVM [30], which learns a new SVM classifier by adapting auxiliary classifiers trained in other related domains. CAPO differs from A-SVM in several aspects: 1) CAPO aims to optimize specific performance measures, while A-SVM considers hinge loss; 2) the auxiliary classifiers of CAPO are used to help find the target classifier in the function space, while A-SVM is proposed for domain adaptation [9] and it employs auxiliary classifier to extract knowledge from related domains; similar ideas can be found in [10]. Generally speaking, classifier adaptation techniques which try to obtain a new classifier based on existing classifiers were mainly used for domain adaptation in previous studies [30], [10]. Here, we use classifier adaptation to optimize specific performance measures, which is quite different.

Ensemble learning is the learning paradigm which employs multiple learners to solve one task [33], and it achieves state-of-the-art performance in many practical applications. In current work, the final classifier generated by CAPO is an ensemble consisting of auxiliary classifiers and the delta function. But, differently from conventional ensemble methods, the component classifiers of CAPO are of two kinds and are generated in two steps: First, auxiliary classifiers are trained; then a delta function which is designed to correct the decision of auxiliary classifiers is added such that the concerned performance measure is optimized.

From the feature augmentation perspective, the nonlinear auxiliary classifiers construct nonlinear features that are augmented to the original features so that the final classifier can have nonlinear generalization performance. This is like *constructive induction* [22], which tries to change the representation of data by creating new features.

Curriculum learning [2] is a learning paradigm which circumvents a challenging learning task by starting with relatively easier subtasks; then, with the help of learned subtasks, the target task can be effectively solved. It was first proposed for training neural networks in [11], and is closely related to the idea of "twice learning" proposed in [34], where a neural network ensemble was trained to help induce a decision tree. The study in [2] shows promising empirical results of curriculum learning. Our proposed CAPO is similar to curriculum learning since it also tries to solve a difficult problem by starting with relatively easier subtasks, but they are quite different because we do not provide a curriculum learning strategy.

5 Empirical Studies

In this section, we perform experiments to evaluate the performance and efficiency of CAPO.

TABLE 1
Datasets Used in the Experiments

Data set	#Feature	#Train	#Test
IJCNN1	22	49,990	91,701
Mitfaces	361	6,977	24,045
Reuters	8,315	7,770	3,299
Splice	60	1,000	2,175
USPS*	256	7,291	2,007

5.1 Configuration

The following five datasets from different application domains are used in our experiments:

- IJCNN1: This dataset is from the IJCNN 2001 neural network competition (task 1); here we use the winner's transformation in [7].
- Mitfaces: The face detection dataset from CBCL at MIT [1].
- Reuters: Text classification data which are to discriminate the `money-fx` documents from others in the Reuters-21578 collection.
- Splice: The task is to recognize two classes of splice junctions in a DNA sequence.
- USPS*: This dataset is to classify the digits "01234" against the digits "56789" on the USPS handwritten digits recognition data.

Table 1 summarizes the information of datasets. On each dataset, we optimize four performance measures (accuracy, F1-score, PRBEP, and AUC) so there are 20 tasks in total. For each task, we train classifiers on training examples, and then evaluate their performances on test examples. The experiments are run on an Intel Xeon E5520 machine with 8 GB memory.

5.2 Comparison with State-of-the-Art Methods

First, we compare the performance and efficiency of CAPO with state-of-the-art methods. Specifically, we compare three methods which can optimize different performance measures, including SVM^{perf}, classification SVM incorporating with a cost model [23], and our proposed CAPO. Detailed implementations of these methods are described as follows:

- CAPO: We use three kinds of classifiers as auxiliary classifiers, including Core Vector Machine (CVM)[1] [26], RBF Neural Network [3], and C4.5 Decision Tree [25], and corresponding CAPOs are denoted as CAPO_{cvm}, CAPO_{nn}, and CAPO_{dt}, respectively. In CAPO_{cvm}, the CVM is with RBF kernel $k(\mathbf{x}_i; \mathbf{x}_j) = \exp(\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, where γ is set to the default value (inverse squared averaged distance between examples) and the parameter C is set to 1. In CAPO_{nn} and CAPO_{dt}, NN and DT are implemented by WEKA [13] with default parameters. Furthermore, we also implement CAPO*, which exploits all three auxiliary classifiers. The parameter C is selected from $C \in$

1. http://www.cs.ust.hk/~ivor/cvm.html. Here, we use the option "-c 1 -e 0.001" for all auxiliary CVMs.

TABLE 2
Performance of Compared Methods, Where the Best Performance for Each Task Is Bolded and the Methods That Cannot Be Completed in 24 Hours Are Indicated by "N/A"

	Task	CAPO_{cvm}	CAPO_{dt}	CAPO_{nn}	CAPO*	$\text{SVM}^{\text{perf}}_{\text{lin}}$	$\text{SVM}^{\text{perf}}_{\text{rbf}}$	$\text{SVM}^{\text{light}}_{\text{lin}}$	$\text{SVM}^{\text{light}}_{\text{rbf}}$
IJCNN1	*Accuracy*	.9540 (.9521)	.9702 (.9702)	.9150 (.8914)	**.9703**	.9193	.9658	N/A	N/A
	F1	.7620 (.7544)	**.8473** (.8471)	.5753 (.2643)	.8468	.5565	N/A	N/A	N/A
	PRBEP	.7723 (.7376)	.8470 (.8364)	.5692 (.3222)	**.8605**	.6016	N/A	N/A	N/A
	AUC	.9607 (.8839)	.9734 (.9464)	.9198 (.8658)	**.9810**	.9180	N/A	N/A	N/A
Mitfaces	*Accuracy*	**.9842** (.9839)	.9458 (.9302)	.9696 (.9067)	.9841	.9727	.9840	.9733	N/A
	F1	**.4658** (.4665)	.1605 (.1342)	.2281 (.1768)	.4514	.2056	N/A	.2015	N/A
	PRBEP	**.5127** (.4979)	.1864 (.1822)	.2500 (.1059)	.4873	.2140	N/A	.2309	N/A
	AUC	**.9148** (.9148)	.7991 (.7201)	.8368 (.7979)	.9137	.8533	N/A	.8450	N/A
Reuters	*Accuracy*	**.9745** (.9745)	.9664 (.9660)	.9715 (.9315)	.9739	.9727	.9727	.9724	.9721
	F1	.7730 (.7729)	.6973 (.6890)	.7455 (.1439)	**.7731**	.7375	N/A	.7599	.7540
	PRBEP	.7654 (.7709)	.7207 (.6871)	.7151 (.3743)	**.7765**	.7598	N/A	.7709	.7598
	AUC	.9870 (.9363)	.9842 (.9144)	.9868 (.8322)	.9838	**.9878**	N/A	.9872	.9873
Splice	*Accuracy*	.8947 (.8947)	.9347 (.9347)	.9651 (.9651)	**.9664**	.8451	.8947	.8446	.8975
	F1	.8955 (.8943)	.9371 (.9362)	**.9659** (.9659)	.9512	.8451	N/A	.8487	.8990
	PRBEP	.8762 (.8691)	.9363 (.9355)	.9576 (.9558)	**.9584**	.8532	N/A	.8523	.9036
	AUC	.9457 (.8992)	.9760 (.9307)	.9836 (.9667)	**.9852**	.9304	N/A	.9267	.9639
USPS*	*Accuracy*	.9691 (.9689)	.9233 (.9233)	.8520 (.7798)	.9676	.8411	**.9706**	N/A	N/A
	F1	.9611 (.9613)	.9060 (.9053)	.8188 (.7486)	**.9617**	.8012	N/A	N/A	N/A
	PRBEP	.9500 (.9488)	.9000 (.8898)	.8195 (.7500)	**.9573**	.7963	N/A	N/A	N/A
	AUC	.9731 (.9658)	.9557 (.9179)	.9137 (.7582)	**.9843**	.9052	N/A	N/A	N/A

For CAPO, the raw performance of the auxiliary classifier is shown in brackets following the entry of the corresponding CAPO.

$\{2^{-7}, \ldots, 2^{7}\}$ by 5-fold cross validation on training data, and the parameter B of CAPO* is simply set to 1.

- SVM^{perf}: We use the codes of SVM^{perf} provided by Joachims.[2] Both linear kernel and RBF kernel are used; the corresponding methods are denoted as $\text{SVM}^{\text{perf}}_{\text{lin}}$ and $\text{SVM}^{\text{perf}}_{\text{rbf}}$, respectively. The parameter C for both methods and the kernel width γ for $\text{SVM}^{\text{perf}}_{\text{rbf}}$ are selected from $C \in \{2^{-7}, \ldots, 2^{7}\}$ and $\gamma \in \{2^{-2}\gamma_0, \ldots, 2^{2}\gamma_0\}$ by five-fold cross validation on training data, where γ_0 is the inverse squared averaged distance between examples.
- SVM with cost model: We implement the SVM with cost model with $\text{SVM}^{\text{light}}$,[3] where the parameter j is used to set different costs for different classes. Specifically, we use $\text{SVM}^{\text{light}}_{\text{lin}}$ and $\text{SVM}^{\text{light}}_{\text{rbf}}$, where linear kernel and RBF kernel are used. The parameter C and j for both methods and the kernel width γ for $\text{SVM}^{\text{light}}_{\text{rbf}}$ are selected from $C \in \{2^{-7}, \ldots, 2^{7}\}$, $j \in \{2^{-2}, \ldots, 2^{6}\}$, and $\gamma \in \{2^{-2}\gamma_0, \ldots, 2^{2}\gamma_0\}$ by five-fold cross validation.

For parameter selection, we extend the search space if the most frequently selected parameter was on a boundary. Note that both SVM with cost model and SVM^{perf} are strong baselines to compare against. Lewis [20] won the TREC-2001 batch filtering evaluation by using the former, and Joachims [15] showed that SVM^{perf} performed better. We apply these methods to the 20 tasks mentioned above, and report their performance. Since time efficiency is also concerned, we report the CPU time used for parameter selection. Note that if one task is not completed in 24 hours, we would stop it and mark it with "N/A."

2. http://svmlight.joachims.org/svm_perf.html.
3. http://svmlight.joachims.org.

Table 2 presents the performance of compared methods as well as the raw performance of auxiliary classifiers (in the brackets following the entries of corresponding CAPO methods), where the best result for each task is bolded. It is obvious that CAPO and $\text{SVM}^{\text{perf}}_{\text{lin}}$ succeed in finishing all tasks in 24 hours. We can observe that CAPO achieves performance improvements over auxiliary classifiers on most tasks, and many of the performance improvements are quite large. For example, on Reuters the best AUC achieved by auxiliary classifiers is 0.9363, while CAPO methods achieve AUC higher than 0.98. This result shows that CAPO is effective in improving the performance with respect to the concerned performance measure. More results for the case of multiple auxiliary classifiers are given in Section 5.3. Moreover, we could see from the results that CAPO methods perform much better than linear methods, i.e., $\text{SVM}^{\text{perf}}_{\text{lin}}$ and $\text{SVM}^{\text{light}}_{\text{lin}}$, especially when optimizing multivariate performance measures like F1-score and PRBEP. For example, CAPO* achieves PRBEP 0.8605, but $\text{SVM}^{\text{perf}}_{\text{lin}}$ achieves only 0.6016 on IJCNN1; CAPO_{nn} achieves F1-score 0.9659, but those of $\text{SVM}^{\text{perf}}_{\text{lin}}$ and $\text{SVM}^{\text{light}}_{\text{lin}}$ are both less than 0.85 on Splice. This can be explained by the fact that CAPO methods exploit the nonlinearity provided by auxiliary classifiers. Meanwhile, it is interesting that all methods achieve similar performances on Reuters; this coincides with the common knowledge that linear classifier is strong enough for text classification tasks. For kernelized methods, i.e., $\text{SVM}^{\text{perf}}_{\text{rbf}}$ and $\text{SVM}^{\text{light}}_{\text{rbf}}$, it is easy to see that they fail to finish in 24 hours on most tasks. On the smallest dataset, Splice, $\text{SVM}^{\text{light}}_{\text{rbf}}$ succeeds in finishing all tasks, its performance is better than linear methods ($\text{SVM}^{\text{perf}}_{\text{lin}}$ and $\text{SVM}^{\text{light}}_{\text{lin}}$); this can be explained by the fact that $\text{SVM}^{\text{light}}_{\text{rbf}}$ exploits nonlinearity by using RBF kernel. Meanwhile, it is easy to see that the performances of CAPO methods,

TABLE 3
CPU Time for Parameter Selection (in Seconds), Where the Tasks Not Completed in 24 Hours Are Indicated by "N/A"

Task		CAPO_{cvm}	CAPO_{dt}	CAPO_{nn}	CAPO*	$\text{SVM}^{\text{perf}}_{\text{lin}}$	$\text{SVM}^{\text{perf}}_{\text{rbf}}$	$\text{SVM}^{\text{light}}_{\text{lin}}$	$\text{SVM}^{\text{light}}_{\text{rbf}}$
IJCNN1	*Accuracy*	9.3	11.1	9.9	11.2	10.0	96.6	N/A	N/A
	F1	9,451.5	9,011.5	14,809.3	6,652.8	12,281.3	N/A		
	PRBEP	1,507.9	1,033.3	2,276.2	1,005.1	2,034.0	N/A		
	AUC	88.0	38.0	124.0	40.6	112.6	N/A		
Mitfaces	*Accuracy*	9.5	11.2	23.7	9.0	27.2	27,089.3	6,114.7	N/A
	F1	465.6	802.5	1,211.5	379.0	1,189.4	N/A		
	PRBEP	126.9	183.4	241.6	119.6	234.4	N/A		
	AUC	37.7	48.5	74.0	30.6	79.3	N/A		
Reuters	*Accuracy*	5.7	2.1	2.6	3.9	2.3	39,813.1	283.1	53,113.8
	F1	68.7	67.4	64.3	67.6	60.2	N/A		
	PRBEP	10.8	13.1	11.9	10.6	11.4	N/A		
	AUC	18.9	8.6	8.7	3.9	8.1	N/A		
Splice	*Accuracy*	4.0	484.5	697.1	2.0	3,602.4	2,187.1	16,297.6	464.2
	F1	168.2	592.3	3,373.9	58.4	10,201.5	N/A		
	PRBEP	11.8	17.0	27.3	6.8	82.6	N/A		
	AUC	2.0	3.3	7.3	1.2	42.0	N/A		
USPS*	*Accuracy*	24.6	35.4	215.3	15.6	221.5	24,026.7	N/A	N/A
	F1	2,199.0	2,605.4	5,429.9	1,514.8	5,225.9	N/A		
	PRBEP	626.2	566.1	938.9	404.4	895.2	N/A		
	AUC	155.6	139.9	424.3	76.1	452.5	N/A		

For CAPO, the CPU time for training auxiliary classifiers is not counted, and they are shown in Table 4.

especially CAPO*, are superior to $\text{SVM}^{\text{light}}_{\text{rbf}}$. It can be understood that RBF kernel may not be suitable for this data, while CAPO* exploits nonlinearity introduced by different kinds of auxiliary classifiers. By comparing CAPO* with other CAPO methods with one auxiliary classifier, it can be found there are many cases where CAPO* performs better. This is not hard to understand because CAPO* exploits more nonlinearity by using different kinds of auxiliary classifiers.

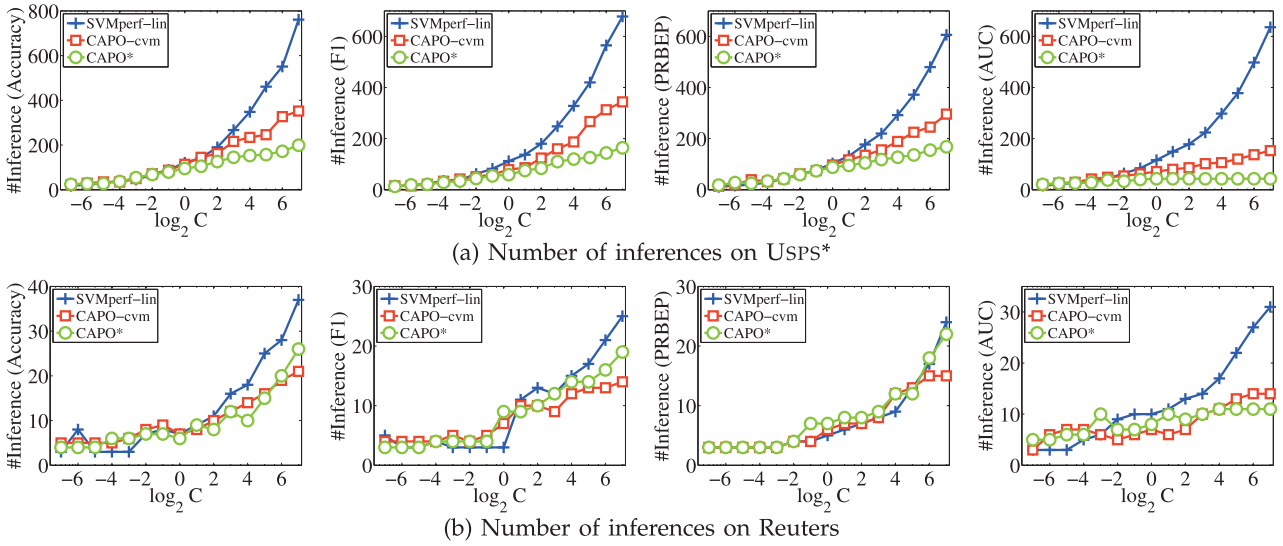
Table 3 shows the CPU time used for parameter selection via cross validation. On each dataset, we employ the same auxiliary classifiers for four different measures, so the times used for training auxiliary classifiers on one dataset are identical, which is shown in Table 4. Also, because four tasks of $\text{SVM}^{\text{light}}$ on one dataset have the same cross-validation process, they have the same cross-validation time. From Tables 3 and 4, we can see kernelized nonlinear methods ($\text{SVM}^{\text{perf}}_{\text{rbf}}$ and $\text{SVM}^{\text{light}}_{\text{rbf}}$) fail to finish in 24 hours on most tasks. This can be understood by the fact that the Gram matrix updating in $\text{SVM}^{\text{perf}}_{\text{rbf}}$ costs much time, as described in Section 2.2, and $\text{SVM}^{\text{light}}_{\text{rbf}}$ has many parameters to tune. Meanwhile, it can be found that CAPO methods are more efficient than others, even after adding the time used for training auxiliary classifiers.

TABLE 4
CPU Time for Training Auxiliary Classifiers (in Seconds)

Data set	CVM	DT	NN
IJCNN1	1.6	19.9	20.2
Mitfaces	2.8	66.1	63.6
Reuters	2.1	1,689.7	1,771.0
Splice	0.1	0.4	0.9
USPS*	2.3	45.9	37.1

Moreover, it is interesting to find that the classifier adaptation procedure of CAPO costs much less time than $\text{SVM}^{\text{perf}}_{\text{lin}}$ except on Reuters, though it employs the latter to solve the adaptation problem. For example, when optimizing F1-score on Splice, CAPO* consumes only 58.4 seconds for cross validation, while $\text{SVM}^{\text{perf}}_{\text{lin}}$ costs more than 10,000 seconds. To understand this phenomenon, we record the number of inferences of the most violated constraint (i.e., solving the argmax in (3) when training $\text{SVM}^{\text{perf}}_{\text{lin}}$, CAPO_{cvm}, and CAPO*. Concretely, on two representative datasets, Reuters and USPS* , the number of inferences under different C values is recorded and Fig. 1 shows the results. From Fig. 1a, we can find that on USPS* , CAPO*, and CAPO_{cvm} have fewer inferences than $\text{SVM}^{\text{perf}}_{\text{lin}}$, especially when C is large. Since the training cost of Algorithm 1 is dominated by the inference, the high efficiency of CAPO* and CAPO_{cvm} is due to fewer number of inferences. This can be understood by the fact that auxiliary classifiers provide estimates of the target classifier and CAPO searches them, while $\text{SVM}^{\text{perf}}_{\text{lin}}$ searches in the whole function space. On Reuters, where three methods have similar time efficiency, we can find from Fig. 1b that the numbers of inferences are small and similar. This can be understood by the fact that linear classifier is strong enough for text classification tasks. Moreover, the adaptation procedure of CAPO* is more efficient than CAPO_{cvm}, and Fig. 1a also shows CAPO* has fewer number of inferences. This indicates that it may be easier to find the target classifier by using multiple auxiliary classifiers, coinciding with the fact that an ensemble can provide a better estimate of the target classifier.

Therefore, we can see that the auxiliary classifiers not only inject nonlinearity, but also make the classifier adaptation procedure more efficient.



(a) Number of inferences on USPS*

(b) Number of inferences on Reuters

Fig. 1. Number of inferences of the most violated constraints (#Inference) when training $\text{SVM}_{\text{lin}}^{\text{perf}}$, CAPO_{cvm}, and CAPO* on USPS* and Reuters, where the x- and y-axes show the C values and #Inference, respectively.

5.3 Effect of Delta Function

To show the effect of adding delta function on auxiliary classifiers, we compare the performance of CAPO with that of the weighted ensemble of auxiliary classifiers, which does not include a delta function. In detail, we train five CVMs as auxiliary classifiers due to its high efficiency. Each CVM is with one of the following five kernels:

1. RBF kernel $k(\mathbf{x}_i;\mathbf{x}_j) = \exp(\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$;
2. polynomial kernel $k(\mathbf{x}_i;\mathbf{x}_j) = (\gamma\mathbf{x}_i^\top\mathbf{x}_j + c_0)^d$;
3. Laplacian kernel $k(\mathbf{x}_i;\mathbf{x}_j) = \exp(\gamma\|\mathbf{x}_i - \mathbf{x}_j\|)$;
4. inverse distance kernel $k(\mathbf{x}_i;\mathbf{x}_j) = \frac{1}{\sqrt{\gamma}\|\mathbf{x}_i-\mathbf{x}_j\|+1}$; and
5. inverse squared distance kernel $k(\mathbf{x}_i;\mathbf{x}_j) = \frac{1}{\gamma\|\mathbf{x}_i-\mathbf{x}_j\|^2+1}$,

where all kernels are with default parameters ($c_0 = 0$ and $d = 3$ in the polynomial kernel, γ is the inverse squared averaged distance between examples in all kernels).

Then, CAPO employs these five CVMs as auxiliary classifiers, and the weighted ensemble learns a set of weights to combine them such that the empirical risk is minimized. Both methods select C from $\{2^{-7},\ldots,2^7\}$ by fivefold cross validation on training data, and B of CAPO is fixed to 1.

Table 5 presents the performances of two methods. It can be seen that CAPO achieves better performance than the weighted ensemble. For example, the weighted ensemble achieves PRBEP 0.8000 on IJCNN1 while CAPO achieves 0.8472; the weighted ensemble achieves AUC 0.9022 but CAPO achieves 0.9486 on Splice. Noting that their difference is that CAPO exploits the delta function, we can see that by adding the delta function, CAPO achieves performance improvement w.r.t. concerned performance measure.

5.4 Effect of Auxiliary Classifier Selection

In the above experiments, we directly use common learning algorithms to train auxiliary classifiers; it is obvious that these auxiliary classifiers are not specially improved according to the concerned performance measure. Then, a straightforward question is how CAPO performs if the auxiliary classifiers are specially improved w.r.t. the concerned performance measure or, in other words, how CAPO performs if we train auxiliary classifiers according to the concerned performance measure. Subsequently, we perform experiments to answer this question. Specifically, rather than training five CVMs with five different kernels with default parameters, we train a set of 50 CVMs and select five from them as auxiliary classifiers based on the concerned performance measure. In detail, these 50 CVMs are trained by independently using the five kernels mentioned above, and the parameter γ for each kernel is set as $\gamma = 1.5^\theta\gamma_0$, where $\theta \in \{-0.5, 0, 0.5, \ldots, 4\}$ and γ_0 is the

TABLE 5
Performance of CAPO and Weighted Ensemble, Where Both Methods Exploit Five CVMs with Different Kernels

TASK		CAPO	Ensemble
IJCNN1	*Accuracy*	.9712	.9632
	F1	.8438	.8439
	PRBEP	.8472	.8000
	AUC	.9892	.9837
Mitfaces	*Accuracy*	.9842	.9837
	F1	.4563	.4446
	PRBEP	.4831	.4767
	AUC	.9097	.9097
Reuters	*Accuracy*	.9715	.9715
	F1	.7429	.7181
	PRBEP	.7598	.7318
	AUC	.9847	.7979
Splice	*Accuracy*	.8952	.8938
	F1	.9024	.9024
	PRBEP	.9010	.8912
	AUC	.9486	.9022
USPS*	*Accuracy*	.9706	.9701
	F1	.9659	.9644
	PRBEP	.9634	.9622
	AUC	.9823	.9705

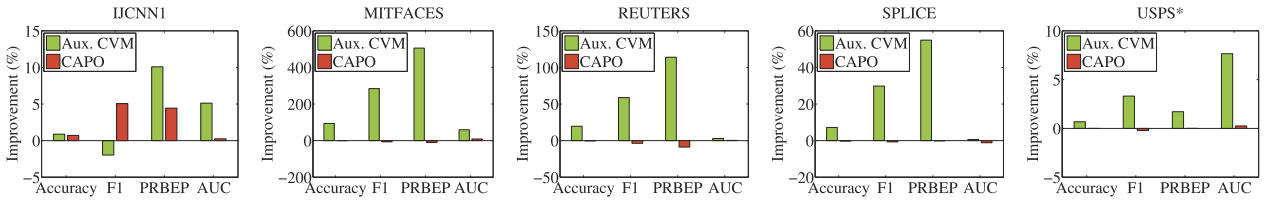


Fig. 2. Comparison between relative improvement of averaged performance of auxiliary classifiers and relative performance improvement of CAPO after auxiliary classifier selection.

default value, and then five CVMs which performs best in terms of the concerned performance measure are selected as auxiliary classifiers. For example, if we want to train classifier optimizing F1-score, then the five CVMs which achieve the highest F1-score are selected. As above, we choose the parameter $C \in \{2^{-7}, \ldots, 2^{7}\}$ by five-fold cross validation and fix B to be 1.

On each task, we compute the relative improvement of the averaged performance of auxiliary classifiers and that obtained by CAPO, and report them in Fig. 2. The relative performance improvement is computed as the performance improvement caused by the auxiliary classifier selection divided by the performance before selection. From Fig. 2, it is easy to see that although the averaged performance of auxiliary classifiers improves a lot after selection, yet the performance of CAPO remains similar in most cases and even degrades in some cases. This may suggest that it is enough to use common CVMs as auxiliary classifiers, and it is not necessary to specially design auxiliary classifiers according to the target performance measure. This can be explained by the fact that the auxiliary classifiers are used to provide approximate solutions to the problem which are combined and further refined by the delta function to obtain the final solution; thus, actually these approximate solutions are not required to be very accurate. Moreover, it is obvious that with respect to time efficiency, CAPO with auxiliary classifier selection has no superiority over the the

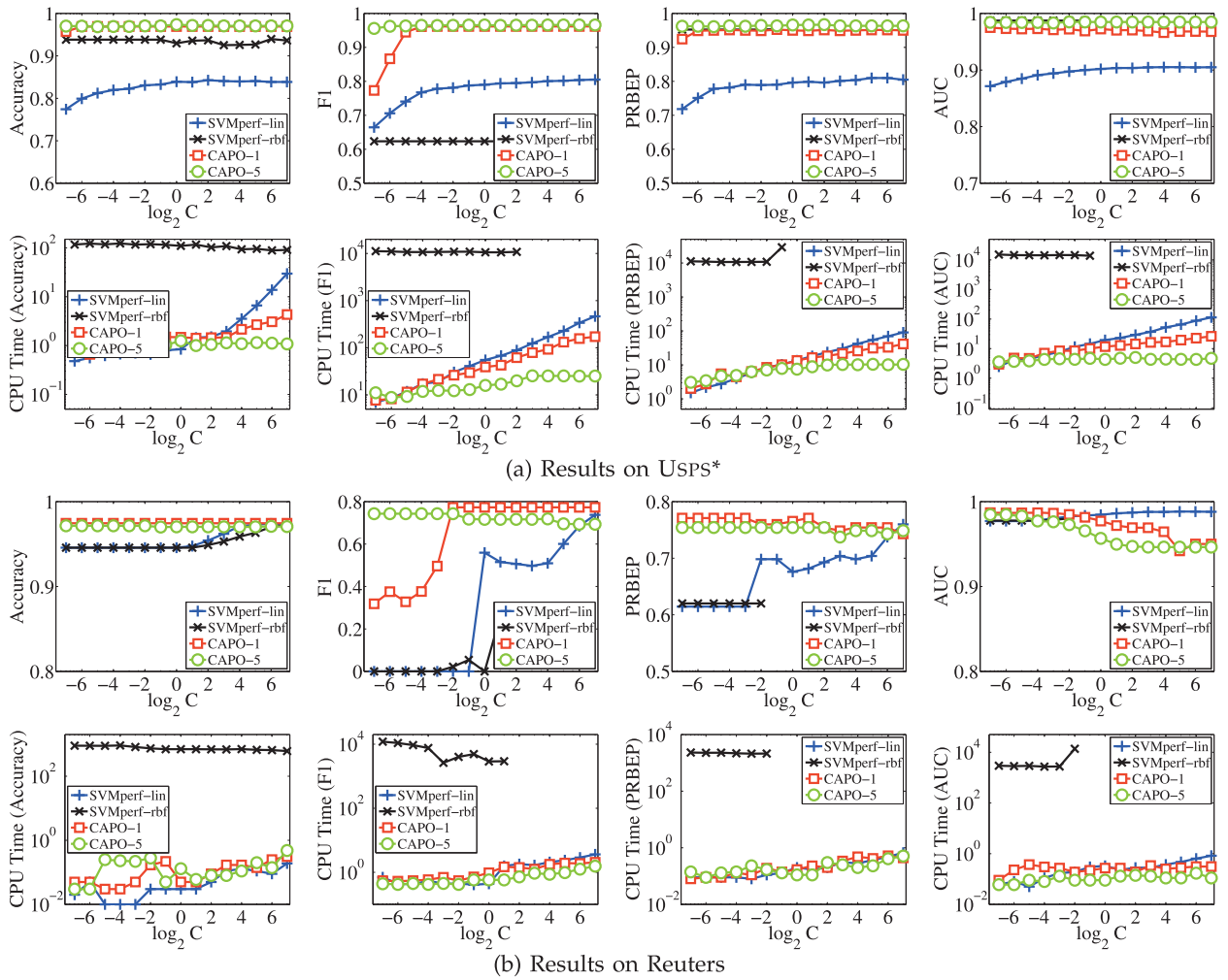


Fig. 3. Performance and CPU time (in seconds) with different Cs, (a) on USPS*; (b) on Reuters. Each subfigure shows performance in the first row and corresponding CPU time in the second row.

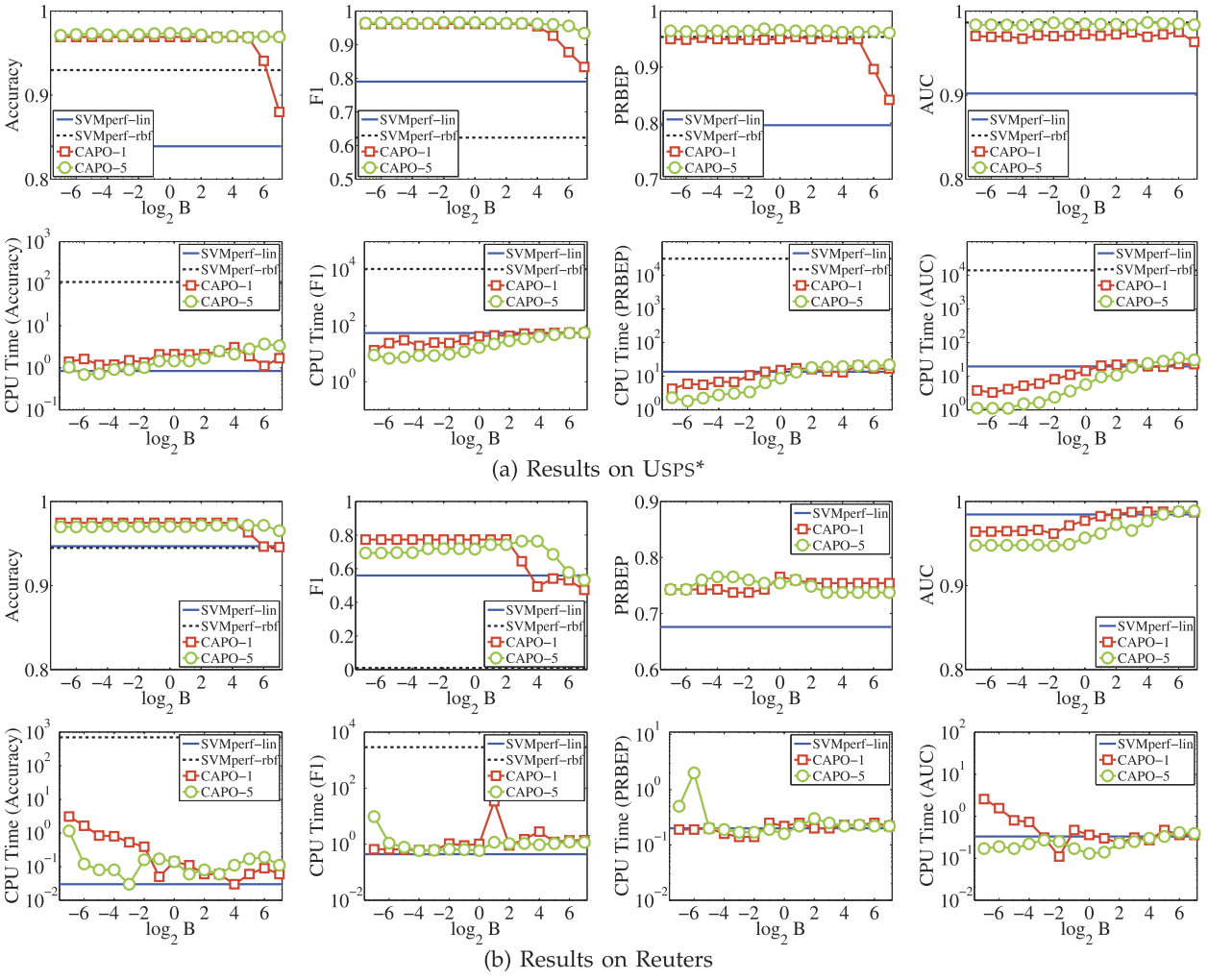


Fig. 4. Performance and CPU time (in seconds) with different Bs: (a) on USPS*, (b) on Reuters. Each subfigure shows performance in the first row and corresponding CPU time in the second row.

original one, especially after counting the time used for training 50 auxiliary CVMs.

5.5 Parameter Sensibility

To study the impact of parameters, we perform experiments on two medium-sized datasets, USPS* and Reuters. The two datasets are representative since nonlinear classifiers perform well on USPS* while linear classifiers work well on Reuters. We study the performance and time efficiency of CAPO_1 and CAPO_5 under different C and B values, where CAPO_1 uses one auxiliary CVM with RBF kernel and CAPO_5 uses five auxiliary CVMs with five different kernels as above; all kernels are with default parameters.

First, we vary C within $\{2^{-7}, 2^{-6}, \ldots, 2^{7}\}$ and fix B to be 1. For comparison, we also train $\text{SVM}^{\text{perf}}_{\text{lin}}$ and $\text{SVM}^{\text{perf}}_{\text{rbf}}$ with the same Cs. Fig. 3 shows the results. It can be found that CAPO_1 and CAPO_5 generally outperform SVM^{perf} at different Cs, except that $\text{SVM}^{\text{perf}}_{\text{rbf}}$ achieves comparable performance as CAPOfor PRBEP and AUC on USPS* and $\text{SVM}^{\text{perf}}_{\text{lin}}$ performs better for AUC at large Cs on Reuters. With respect to time efficiency, CAPO_1, CAPO_5, and $\text{SVM}^{\text{perf}}_{\text{lin}}$ cost comparable CPU time, which is much less than $\text{SVM}^{\text{perf}}_{\text{rbf}}$. Moreover, CAPO_1 and CAPO_5 scale better when C increases, and they are more efficient than $\text{SVM}^{\text{perf}}_{\text{lin}}$ at large Cs. Moreover, it is easy to find that our methods, especially CAPO_5, are more robust with C.

Second, we vary B within $\{2^{-7}, 2^{-6}, \ldots, 2^{7}\}$ with fixed $C = 1$ for CAPO_1 and CAPO_5. As comparisons, $\text{SVM}^{\text{perf}}_{\text{lin}}$ and $\text{SVM}^{\text{perf}}_{\text{rbf}}$ are trained with $C = 1$. The results are shown in Fig. 4, where $\text{SVM}^{\text{perf}}_{\text{lin}}$ and $\text{SVM}^{\text{perf}}_{\text{rbf}}$ are illustrated as straight lines because they do not have the parameter B. In general, CAPO_1 and CAPO_5 achieve better performance at different Bs in most cases, except for AUC on Reuters. Also, CAPO_1 and CAPO_5 have comparable efficiency with $\text{SVM}^{\text{perf}}_{\text{lin}}$, which is much better than $\text{SVM}^{\text{perf}}_{\text{rbf}}$. We can see that our methods are quite robust to parameters B, and comparatively speaking, CAPO_5 is more robust than CAPO_1.

Thus, we can see that our methods, especially CAPO_5, are robust to B and C. Comparatively speaking, CAPO_5 is more robust and more efficient than CAPO_1; this verifies our previous results.

5.6 Scalability w.r.t. Training Set Size

To evaluate scalability of CAPO, we perform experiments on the largest dataset IJCNN1. We first train CAPO_1 and CAPO_5 using $\{1/32, 1/16, 1/8, 1/4, 1/2, 1\}$ of all training examples, and then evaluate them on test examples. As comparisons, $\text{SVM}^{\text{perf}}_{\text{lin}}$ and $\text{SVM}^{\text{perf}}_{\text{rbf}}$ are also trained under the same

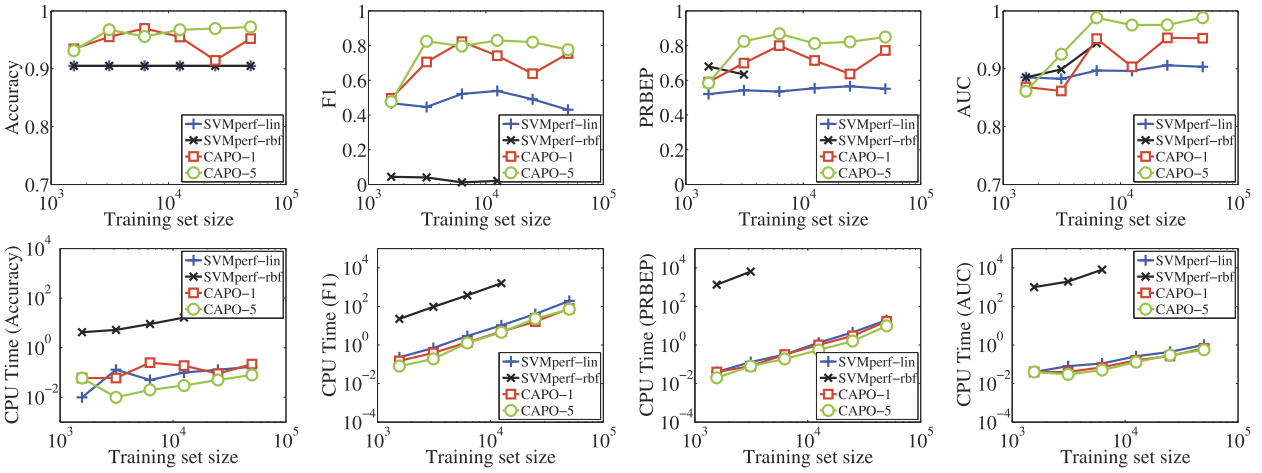


Fig. 5. Performance (first row) and CPU time (second row, in seconds) with different training set sizes on IJCNN1.

configuration. In this experiment, we simply fix both the parameters B and C to be 1. We report performance of compared methods and the corresponding used CPU time.

Fig. 5 shows the results of the achieved performance and the corresponding running time in the first and second rows, respectively. As we can see, all methods scale well except that $\mathrm{SVM}^{\mathrm{perf}}_{\mathrm{rbf}}$ has to be terminated early when the training set size increases. Moreover, compared with $\mathrm{SVM}^{\mathrm{perf}}_{\mathrm{lin}}$, it is easy to see that CAPO_5 achieves better performance but costs less time at every training set size.

5.7 Summary

Based on above empirical studies, we can see that CAPO is an effective and efficient approach to training classifier that optimizes performance measures. Compared with $\mathrm{SVM}^{\mathrm{perf}}$ and SVM with cost model, it can achieve better performances at lower time costs. As well, it has been shown that CAPO is robust to parameters and scales well w.r.t. the training data size. For practical implementation, training auxiliary classifiers by optimizing accuracy is a good choice because many efficient algorithms have been developed in the literature, and the experiments in Section 5.4 suggest that using auxiliary classifiers with higher target performances does not show significant superiority, especially when tuning auxiliary classifiers costs much time. Meanwhile, it can be better to use multiple diverse auxiliary classifiers.

6 Conclusion and Future Work

This paper presents a new approach, CAPO, to training classifier that optimizes specific performance measure. Rather than designing sophisticated algorithms, we solve the problem in two steps: First, we train auxiliary classifiers by taking existing off-the-shelf learning algorithms; then these auxiliary classifiers are adapted to optimize the concerned performance measure. We show that the classifier adaptation problem can be formulated as an optimization problem similar to linear $\mathrm{SVM}^{\mathrm{perf}}$ and can be efficiently solved. In practice, the auxiliary classifier (or ensemble of auxiliary classifiers) benefits CAPO in two aspects:

1. By using nonlinear auxiliary classifiers, it injects nonlinearity that is quite needed in practical applications.
2. It provides an estimate of the target classifier, making the classifier adaption procedure more efficient.

Extensive empirical studies show that the classifier adaptation procedure helps to find the target classifier for the concerned performance measure. Moreover, the learning process becomes more efficient than linear $\mathrm{SVM}^{\mathrm{perf}}$ due to fewer inferences in CAPO.

In this work, linear delta function is used for classifier adaptation. Although it achieves good performances, interesting and promising future work is to exploit nonlinear delta function for this problem.

Acknowledgments

The authors want to thank the anonymous reviewers and the associate editor for their helpful comments and suggestions. This research was supported by the National Fundamental Research Program of China (2010CB327903), the National Science Foundation of China (61073097, 61021062), and the Jiangsu Science Foundation (BK2011566). Z.-H. Zhou is the corresponding author of this paper.

References

[1] M. Alvira and R. Rifkin, "An Empirical Comparison of SNoW and SVMs for Face Detection," Technical Report 2001-004, CBCL, MIT, Cambridge, Mass., 2001.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum Learning," *Proc. Int'l Conf. Machine Learning*, pp. 41-48, 2009.

[3] C.M. Bishop, *Neural Networks for Pattern Recognition*. Oxford Univ. Press, 1995.

[4] C.J.C. Burges, R. Ragno, and Q. Le, "Learning to Rank with Nonsmooth Cost Functions," *Advances in Neural Information Processing Systems 20*, pp. 193-200, 2006.

[5] Y. Cao, J. Xu, T.Y. Liu, H. Li, Y. Huang, and H.W. Hon, "Adapting Ranking SVM to Document Retrieval," *Proc. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 186-193, 2006.

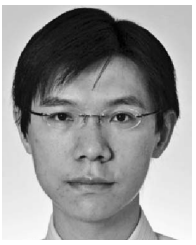
[6] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble Selection from Libraries of Models," *Proc. Int'l Conf. Machine Learning*, pp. 18-25, 2004.

[7] C.C. Chang and C.J. Lin, "IJCNN 2001 Challenge: Generalization Ability and Text Decoding," *Proc. Int'l Joint Conf. Neural Networks*, pp. 1031-1036, 2001.

[8] C. Cortes and M. Mohri, "AUC Optimization vs. Error Rate Minimization," *Advances in Neural Information Processing Systems 16*, pp. 313-320, 2004.

[9] H. Daumé III and D. Marcu, "Domain Adaptation for Statistical Classifiers," *J. Artificial Intelligence Research,* vol. 26, pp. 101-126, 2006.
[10] L. Duan, I.W. Tsang, D. Xu, and T.S. Chua, "Domain Adaptation from Multiple Sources via Auxiliary Classifiers," *Proc. Int'l Conf. Machine Learning,* pp. 289-296, 2009.
[11] J.L. Elman, "Learning and Development in Neural Networks: The Importance of Starting Small," *Cognition,* vol. 48, no. 6, pp. 781-799, 1993.
[12] C. Ferri, P. Flach, and J. Hernandez-Orallo, "Learning Decision Trees Using the Area under the ROC Curve," *Proc. Int'l Conf. Machine Learning,* pp. 139-146, 2002.
[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Exploration Newsletter,* vol. 11, no. 1, pp. 10-18, 2009.
[14] A. Herschtal and B. Raskutti, "Optimising Area under the ROC Curve Using Gradient Descent," *Proc. Int'l Conf. Machine Learning,* pp. 49-56, 2004.
[15] T. Joachims, "A Support Vector Method for Multivariate Performance Measures," *Proc. Int'l Conf. Machine Learning,* pp. 377-384, 2005.
[16] T. Joachims, T. Finley, and C.-N. J. Yu, "Cutting-Plane Training of Structural SVMs," *Machine Learning,* vol. 76, no. 1, pp. 27-59, 2009.
[17] T. Joachims and C.-N. J. Yu, "Sparse Kernel SVMs via Cutting-Plane Training," *Machine Learning,* vol. 76, nos. 2/3, pp. 179-193, 2009.
[18] J. Lafferty and C. Zhai, "Document Language Models, Query Models, and Risk Minimization for Information Retrieval," *Proc. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval,* pp. 111-119, 2001.
[19] J. Langford and B. Zadrozny, "Estimating Class Membership Probabilities Using Classifier Learners," *Proc. Int'l Workshop Artificial Intelligence and Statistics,* pp. 198-205, 2005.
[20] D.D. Lewis, "Applying Support Vector Machines to the TREC-2001 Batch Filtering and Routing Tasks," *Proc. Text REtrieval Conf.,* pp. 286-292, 2001.
[21] X. Li and J. Bilmes, "A Bayesian Divergence Prior for Classifier Adaptation," *Proc. Int'l Conf. Artificial Intelligence and Statistics,* pp. 275-282, 2007.
[22] C.J. Matheus and L.A. Rendell, "Constructive Induction on Decision Trees," *Proc. Int'l Joint Conf. Artificial Intelligence,* pp. 645-650, 1989.
[23] K. Morik, P. Brockhausen, and T. Joachims, "Combining Statistical Learning with a Knowledge-Based Approach—a Case Study in Intensive Care Monitoring," *Proc. Int'l Conf. Machine Learning,* pp. 268-277, 1999.
[24] D.R. Musicant, V. Kumar, and A. Ozgur, "Optimizing F-Measure with Support Vector Machines," *Proc. Int'l Florida Artificial Intelligence Research Soc. Conf.,* pp. 356-360, 2003.
[25] J.R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
[26] I.W. Tsang, J.T. Kwok, and P.-M. Cheung, "Core Vector Machines: Fast SVM Training on Very Large Data Sets," *J. Machine Learning Research,* vol. 6, pp. 363-392, 2005.
[27] H. Valizadegan, R. Jin, R. Zhang, and J. Mao, "Learning to Rank by Optimizing NDCG Measure," *Advances in Neural Information Processing Systems 22,* pp. 1883-1891, 2009,
[28] J. Xu, T.Y. Liu, M. Lu, H. Li, and W.Y. Ma, "Directly Optimizing Evaluation Measures in Learning to Rank," *Proc. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval,* pp. 107-114, 2008.
[29] J. Yang and A.G. Hauptmann, "A Framework for Classifier Adaptation and Its Applications in Concept Detection," *Proc. ACM SIGMM Int'l Conf. Multimedia Information Retrieval,* pp. 467-474, 2008.
[30] J. Yang, R. Yan, and A.G. Hauptmann, "Cross-Domain Video Concept Detection Using Adaptive SVMs," *Proc. Int'l Conf. Multimedia,* pp. 188-197, 2007.
[31] C.-N.J. Yu and T. Joachims, "Training Structural SVMs with Kernels Using Sampled Cuts," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 794-802, 2008.
[32] Y. Yue, T. Finley, F. Radlinski, and T. Joachims, "A Support Vector Method for Optimizing Average Precision," *Proc. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval,* pp. 271-278, 2007.
[33] Z.H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, 2012.
[34] Z.-H. Zhou and Y. Jiang, "NeC4.5: Neural Ensemble Based C4.5," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 6, pp. 770-773, June 2004.

Nan Li received the MSc degree in computer science from Nanjing University, China, in 2008. In the same year, he became a faculty member in the School of Mathematical Sciences at Soochow University, China. He is currently working toward the PhD degree in the Department of Computer Science & Technology of Nanjing University. His research interests include machine learning, data mining, and ubiquitous computing. He and other LAMDA members won the Grand Prize (Open Category) in the PAKDD 2012 Data Mining Competition. His coauthored paper won the Best Paper Runner-Up Award at MobiQuitous 2011. He also received the IBM PhD Fellowship in 2013.



Ivor W. Tsang received the PhD degree in computer science from the Hong Kong University of Science and Technology, Kowloon, Hong Kong, in 2007. He is currently an assistant professor with the School of Computer Engineering, Nanyang Technological University (NTU), Singapore. He is the deputy director of the Center for Computational Intelligence, NTU. He was the recpient of the prestigious *IEEE Transactions on Neural Networks* Outstanding 2004 Paper Award in 2006, the 2008 National Natural Science Award (Class II), China, in 2009, the Best Student Paper Award at CVPR in 2010, the Best Paper Award at ICTAI in 2011, the Best Poster Honorable Mention at ACML in 2012, the Microsoft Fellowship in 2005, and the ECCV 2012 Outstanding Reviewer Award.

Zhi-Hua Zhou received the BSc, MSc, and PhD degrees in computer science from Nanjing University, China, in 1996, 1998, and 2000, respectively, all with the highest honors. He joined the Department of Computer Science & Technology at Nanjing University as an assistant professor in 2001, and is currently a professor and the director of the LAMDA group. His research interests are mainly in artificial intelligence, machine learning, data mining, pattern recognition, and multimedia information retrieval. In these areas he has published more than 90 papers in leading international journals or conference proceedings, and he holds 12 patents. He has won various awards/honors, including the National Science & Technology Award for Young Scholars of China, the Fok Ying Tung Young Professorship 1st-Grade Award, the Microsoft Young Professorship Award, and eight international journals/conferences paper awards and competition awards. He serves/served as associate editor-in-chief of the *Chinese Science Bulletin*, an associate editor of the *IEEE Transactions on Knowledge and Data Engineering* and the *ACM Transactions on Intelligent Systems and Technology*, and as an editorial board member for more than 10 other journals. He is the founder and steering committee chair of ACML, and steering committee member of PAKDD and PRICAI. He serves/ed as general chair/cochair of ACML '12, ADMA '12, and PCM '13, program chair/cochair for PAKDD '07, PRICAI '08, ACML '09, and SDM '13, workshop chair of KDD '12, program vice chair or area chair of various conferences, and chaired many domestic conferences in China. He is the chair of the Machine Learning Technical Committee of the Chinese Association of Artificial Intelligence, chair of the Artificial Intelligence & Pattern Recognition Technical Committee of the China Computer Federation, vice chair of the Data Mining Technical Committee of IEEE Computational Intelligence Society, and the chair of the IEEE Computer Society Nanjing Chapter. He is a fellow of the IAPR, the IEEE, and the IET/IEE.