MULTI-PARAMETRIC SOLUTION-PATH ALGORITHM FOR INSTANCE-WEIGHTED SUPPORT VECTOR MACHINES

Masayuki Karasuyama[1]*, Naoyuki Harada*[2]*, Masashi Sugiyama*[1] *and Ichiro Takeuchi*[2]

[1]Department of Computer Science, Tokyo Institute of Technology, Tokyo 152-8552, Japan
[2]Department of Engineering, Nagoya Institute of Technology, Nagoya, Aichi 466-8555, Japan

ABSTRACT

An *instance-weighted* variant of the support vector machine (SVM) has attracted considerable attention recently since they are useful in various machine learning tasks such as non-stationary data analysis, heteroscedastic data modeling, transfer learning, learning to rank, and transduction. An important challenge in these scenarios is to overcome the computational bottleneck—instance weights often change dynamically or adaptively, and thus the weighted SVM solutions must be repeatedly computed. In this paper, we develop an algorithm that can efficiently and exactly update the weighted SVM solutions for arbitrary change of instance weights. Technically, this contribution can be regarded as an extension of the conventional *solution-path* algorithm for a single regularization parameter to multiple instance-weight parameters. However, this extension gives rise to a significant problem that *breakpoints* (at which the solution path turns) have to be identified in high-dimensional space. To facilitate this, we introduce a parametric representation of instance weights which allows us to find the breakpoints in high-dimensional space easily. Despite its simplicity, our parametrization covers various important machine learning tasks and it widens the applicability of the solution-path algorithm. Through extensive experiments on various practical applications, we demonstrate the usefulness of the proposed algorithm.

1. INTRODUCTION

The most fundamental principle of machine learning would be the *empirical risk minimization*, i.e., the sum of empirical losses over training instances is minimized:

$$\min \sum_i L_i,$$

where L_i denotes the empirical loss for the ith training instance. This empirical risk minimization approach was proved to produce *consistent* estimators [1]. On the other hand, one may also consider an *instance-weighted* variant of empirical risk minimization:

$$\min \sum_i C_i L_i,$$

where C_i denotes the weight for the ith training instance. This weighted variant plays an important role in various machine learning tasks such as

- Non-stationary data analysis
- Heteroscedastic data modeling
- Covariate shift adaptation, transfer learning, and multi-task learning
- Learning to rank and ordinal regression
- Transduction and semi-supervised learning

A common challenge in the research of instance-weighted learning has been to overcome the computational issue. In many of these tasks, instance weights often change dynamically or adaptively, and thus the instance-weighted solutions must be repeatedly computed. For example, in non-stationary data analysis, when training instances are provided in a sequential manner under changing environment, smaller weights are often assigned to older instances for imposing some 'forgetting' effect. In such a situation, every time when a new instance is observed, all the instance weights must be updated in such a way that newer instances have larger weights and older instances have smaller weights. Model selection in instance-weighted learning also poses a considerable computational burden. In many of the above scenarios, we only have qualitative knowledge about instance weights. For example, in the aforementioned ranking problem, we only know that higher-ranked items should have larger weights than lower-ranked items, but it is often difficult to know how large or small these weights should be. The problem of selecting the optimal weighting patterns is an instance of model selection, and many instance-weighted solutions with various weighting patterns must be computed in the model selection phase. The goal of this paper is to alleviate the computational bottleneck of instance-weighted learning.

In this paper, we focus on the *support vector machine* (SVM) [1], which is a popular classification algorithm minimizing a regularized empirical risk:

$$\min R + C \sum_i L_i,$$

where R is a regularization term and $C \geq 0$ controls the trade-off between the regularization effect and the empirical risk minimization. We consider an instance-weighted variant of SVM, which we refer to as the *weighted SVM* (WSVM) (e.g., [2]):

$$\min R + \sum_i C_i L_i.$$

For ordinary SVM, the *solution path algorithm* was proposed [3], which allows efficient computation of SVM solutions for all C by utilizing the piecewise-linear structure of the solutions w.r.t. C. This technique is known as *parametric programming* in the optimization community [4], and has been applied to various machine learning tasks recently [5, 6, 7]; the *incremental-decremental SVM algorithm*, which efficiently follows the piecewise-linear solution path when some training instances are added or removed from the

training set, is also based on the same parametric programming technique [8, 9].

The solution path algorithms described above have been developed for problems with a *single* hyper-parameter. Recently, attention has been paid to studying solution-path tracking in two-dimensional hyper-parameter space. For example, [10] developed a path-following algorithm for regularization parameter C and an insensitive zone thickness ε in *support vector regression*. [11] studied a path-following algorithm for regularization parameter λ and quantile parameter τ in *kernel quantile regression* [12]. However, these works are highly specialized to specific problem structure of bivariate path-following, and it is not straightforward to extend them to more than two hyper-parameters. Thus, the existing approaches may not be applicable to path-following of WSVM, which contains n-dimensional instance-weight parameters $\boldsymbol{c} = [C_1, \ldots, C_n]^\top$, where n is the number of training instances.

In order to go beyond the limitation of the existing approaches, we derive a general solution path algorithm for efficiently computing the solution path of *multiple* instance-weight parameters $\boldsymbol{c}$ in WSVM. This extension involves a significant problem that *breakpoints* (at which the solution path turns) have to be identified in high-dimensional space. To facilitate this, we introduce a parametric representation of instance weights which allows us to find the breakpoints in high-dimensional space easily. Using this parametrization, we can construct an algorithm similarly to one-dimensional regularization path algorithms. Despite its simplicity and usefulness, it has not been exploited so far in machine learning literature. We will illustrate that our approach covers various important machine learning problems and greatly widens the applicability of the path-following approach. We also provide a geometric interpretation of weight space using a notion of *critical region* from the studies of *multi-parametric programming* [13]. A critical region is a polyhedron in which the current *affine* solution remains to be optimal (see Figure 1). This enables us to find breakpoints at intersections of the solution path and the boundaries of polyhedrons.

This paper is organized as follows. Section 2 reviews the definition of WSVM and its optimality conditions. Then we derive a path-following algorithm for WSVM in Section 3. Section 4 is devoted to experimentally illustrating advantages of our algorithm on a toy problem, on-line time-series analysis and covariate shift adaptation. Extensions to regression, ranking, and transduction scenarios are discussed in Section 4.3. Finally, we conclude in Section 5. Although we only discuss classification problems in this paper, a similar algorithm can also be derived for regression problems (see a longer version of this paper on arXiv [14]).

2. PROBLEM FORMULATION

In this section, we review the definition of the *weighted support vector machine* (WSVM) and its optimality conditions. For the moment, we focus on binary classification scenarios. Later in we extend our discussion to regression in Section 4.3, and more general scenarios including ranking in a separate technical report [14].

2.1. WSVM

Let us consider a binary classification problem. Denote n training instances as $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^p$ is an input and $y_i \in \{-1, +1\}$ is an output label.

SVM [1] is a learning algorithm of a linear decision boundary $f(\boldsymbol{x}) = \boldsymbol{w}^\top \Phi(\boldsymbol{x}) + b$ in a feature space $\mathcal{F}$, where $\Phi : \mathcal{X} \to \mathcal{F}$ is a map from the input space $\mathcal{X}$ to the feature space $\mathcal{F}$, $\boldsymbol{w} \in \mathcal{F}$ is a coefficient vector, $b \in \mathbb{R}$ is a bias term, and $^\top$ denotes the transpose. The parameters $\boldsymbol{w}$ and b are learned as

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^n [1 - y_i f(\boldsymbol{x}_i)]_+, \tag{1}$$

where $\frac{1}{2}\|\boldsymbol{w}\|_2^2$ is the regularization term, $\|\cdot\|$ denotes the Euclidean norm, C is the trade-off parameter, and $[z]_+ = \max\{0, z\}$. $[1 - y_i f(\boldsymbol{x}_i)]_+$ is the so-called *hinge-loss* for the ith training instance.

WSVM is an extension of the ordinary SVM so that each training instance possesses its own weight [2]:

$$\min_{\boldsymbol{w},b} \ \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^n C_i [1 - y_i f(\boldsymbol{x}_i)]_+, \tag{2}$$

where C_i is the weight for the ith training instance. WSVM includes the ordinary SVM as a special case when $C_i = C$ for $i = 1, \ldots, n$.

The goal of this paper is to derive an algorithm that can efficiently compute the sequence of WSVM solutions for arbitrary weighting patterns of $\boldsymbol{c} = [C_1, \ldots, C_n]^\top$.

2.2. Optimization in WSVM

Here we review basic optimization issues of WSVM which are used in the following section. The dual formulation of the problem (2) is given as follows:

$$\max_{\{\alpha_i\}_{i=1}^n} -\frac{1}{2}\sum_{i,j=1}^n \alpha_i \alpha_j Q_{ij} + \sum_{i=1}^n \alpha_i \ \text{s.t.} \begin{cases} \sum_{i=1}^n y_i \alpha_i = 0, \\ 0 \le \alpha_i \le C_i, \end{cases}$$

where $Q_{ij} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_j)$ is a *reproducing kernel*. The discriminant function $f : \mathcal{X} \to \mathbb{R}$ is represented in the following form: $f(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b$.

The optimality conditions of the dual problem, called the *Karush-Kuhn-Tucker (KKT) conditions*, are summarized as follows:

$$y_i f(\boldsymbol{x}_i) \ge 1, \quad \text{if} \quad \alpha_i = 0, \tag{3a}$$
$$y_i f(\boldsymbol{x}_i) = 1, \quad \text{if} \quad 0 < \alpha_i < C_i, \tag{3b}$$
$$y_i f(\boldsymbol{x}_i) \le 1, \quad \text{if} \quad \alpha_i = C_i, \tag{3c}$$
$$\sum_{i=1}^n y_i \alpha_i = 0. \tag{3d}$$

We define the following three index sets for later use:

$$\mathcal{O} = \{i \mid \alpha_i = 0\}, \tag{4a}$$
$$\mathcal{M} = \{i \mid 0 < \alpha_i < C_i\}, \tag{4b}$$
$$\mathcal{I} = \{i \mid \alpha_i = C_i\}, \tag{4c}$$

where $\mathcal{O}$, $\mathcal{M}$, and $\mathcal{I}$ stand for 'Outside the margin' ($y_i f(\boldsymbol{x}_i) \ge 1$), 'on the Margin' ($y_i f(\boldsymbol{x}_i) = 1$), and 'Inside the margin' ($y_i f(\boldsymbol{x}_i) \le 1$), respectively.

In what follows, the subscript by an index set such as $\boldsymbol{v}_\mathcal{I}$ for a vector $\boldsymbol{v} \in \mathbb{R}^n$ indicates a sub-vector of $\boldsymbol{v}$ whose elements are indexed by $\mathcal{I}$. For example, for $\boldsymbol{v} = (a, b, c)^\top$ and $\mathcal{I} = \{1, 3\}$,

$\boldsymbol{v}_{\mathcal{I}} = (a, c)^\top$. Similarly, the subscript by two index sets such as $\boldsymbol{M}_{\mathcal{M},\mathcal{O}}$ for a matrix $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ denotes a sub-matrix whose rows and columns are indexed by $\mathcal{M}$ and $\mathcal{O}$, respectively. The principal sub-matrix such as $\boldsymbol{M}_{\mathcal{M},\mathcal{M}}$ is abbreviated as $\boldsymbol{M}_{\mathcal{M}}$.

3. SOLUTION-PATH ALGORITHM FOR WSVM

The path-following algorithm for the ordinary SVM [3] computes the entire solution path for the single regularization parameter C. In this section, we develop a path-following algorithm for the vector of weights $\boldsymbol{c} = [C_1, \ldots, C_n]^\top$. Our proposed algorithm keeps track of the optimal α_i and b when the weight vector $\boldsymbol{c}$ is changed.

3.1. Analytic Expression of WSVM Solutions

Let $\boldsymbol{\alpha} = [\alpha_i, \ldots, \alpha_n]^\top$, $\boldsymbol{y} = [y_i, \ldots, y_n]^\top$, and $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ be a matrix whose (i, j) element is Q_{ij}. Then, using the index sets (4b) and (4c), we can expand one of the KKT conditions, (3b), as

$$\boldsymbol{Q}_{\mathcal{M}}\boldsymbol{\alpha}_{\mathcal{M}} + \boldsymbol{Q}_{\mathcal{M},\mathcal{I}}\boldsymbol{c}_{\mathcal{I}} + \boldsymbol{y}_{\mathcal{M}} b = \mathbf{1}, \tag{5}$$

where $\mathbf{1}$ denotes the vector with all ones. Similarly, another KKT condition (3d) is expressed as

$$\boldsymbol{y}_{\mathcal{M}}^\top \boldsymbol{\alpha}_{\mathcal{M}} + \boldsymbol{y}_{\mathcal{I}}^\top \boldsymbol{c}_{\mathcal{I}} = 0. \tag{6}$$

Let us define a matrix:

$$\boldsymbol{M} = \begin{bmatrix} 0 & \boldsymbol{y}_{\mathcal{M}}^\top \\ \boldsymbol{y}_{\mathcal{M}} & \boldsymbol{Q}_{\mathcal{M}} \end{bmatrix}.$$

Arranging (5) and (6), we obtain the the following system of $|\mathcal{M}| + 1$ linear equations, where $|\mathcal{M}|$ denotes the number of elements in the set $\mathcal{M}$:

$$\begin{bmatrix} b \\ \boldsymbol{\alpha}_{\mathcal{M}} \end{bmatrix} = -\boldsymbol{M}^{-1} \begin{bmatrix} \boldsymbol{y}_{\mathcal{I}}^\top \\ \boldsymbol{Q}_{\mathcal{M},\mathcal{I}} \end{bmatrix} \boldsymbol{c}_{\mathcal{I}} + \boldsymbol{M}^{-1} \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}, \tag{7}$$

where we implicitly assumed that $\boldsymbol{M}$ is invertible[1]. Since b and $\boldsymbol{\alpha}_{\mathcal{M}}$ are *affine* w.r.t. $\boldsymbol{c}_{\mathcal{I}}$, we can calculate the change of b and $\boldsymbol{\alpha}_{\mathcal{M}}$ by (7) as long as the weight vector $\boldsymbol{c}$ is changed continuously. By the definition of $\mathcal{I}$ and $\mathcal{O}$, the remaining parameters $\boldsymbol{\alpha}_{\mathcal{I}}$ and $\boldsymbol{\alpha}_{\mathcal{O}}$ are merely given by

$$\boldsymbol{\alpha}_{\mathcal{I}} = \boldsymbol{c}_{\mathcal{I}} \text{ and } \boldsymbol{\alpha}_{\mathcal{O}} = \mathbf{0}. \tag{8}$$

A change of the index sets $\mathcal{M}$, $\mathcal{O}$, and $\mathcal{I}$ is called an *event*. As long as no event occurs, the WSVM solutions for all $\boldsymbol{c}$ can be computed by (7)–(8) since all the KKT conditions (3a)–(3d) are still satisfied. However, when an event occurs, we need to check the violation of the KKT conditions. Below, we address the issue of event detection when $\boldsymbol{c}$ is changed.

3.2. Event Detection

Suppose we want to change the weight vector from $\boldsymbol{c}^{(\mathrm{old})}$ to $\boldsymbol{c}^{(\mathrm{new})}$ (see Figure 1). This can be achieved by moving the weight vector $\boldsymbol{c}^{(\mathrm{old})}$ toward the direction of $\boldsymbol{c}^{(\mathrm{new})} - \boldsymbol{c}^{(\mathrm{old})}$.

[1]The invertibility of the matrix $\boldsymbol{M}$ is assured if and only if the submatrix $\boldsymbol{Q}_{\mathcal{M}}$ is positive definite in the subspace $\{\boldsymbol{z} \in \mathbb{R}^{|\mathcal{M}|} \mid \boldsymbol{y}_{\mathcal{M}}^\top \boldsymbol{z} = 0\}$. We assume this technical condition here. A notable exceptional case is that $\mathcal{M}$ is empty— we discuss how to cope with this case in detail in [14].

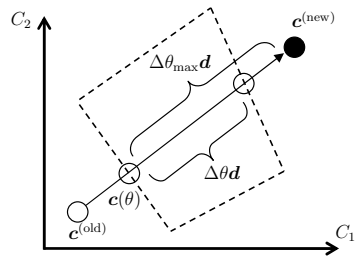

Fig. 1. The schematic illustration of path-following in the space of $\boldsymbol{c} \in \mathbb{R}^2$, where the WSVM solution is updated from $\boldsymbol{c}^{(\mathrm{old})}$ to $\boldsymbol{c}^{(\mathrm{new})}$. Suppose we are currently at $\boldsymbol{c}(\theta)$. The vector $\boldsymbol{d}$ represents the update direction $\boldsymbol{c}^{(\mathrm{new})} - \boldsymbol{c}^{(\mathrm{old})}$, and the polygonal region enclosed by dashed lines indicates the current critical region. Although $\boldsymbol{c}(\theta) + \Delta\theta_{\max}\boldsymbol{d}$ seems to directly lead the solution to $\boldsymbol{c}^{(\mathrm{new})}$, the maximum possible update from $\boldsymbol{c}(\theta)$ is $\Delta\theta\boldsymbol{d}$; otherwise the KKT conditions are violated. To go beyond the border of the critical region, we need to update the index sets $\mathcal{M}$, $\mathcal{I}$, and $\mathcal{O}$ to fulfill the KKT conditions.

Let us write the line segment between $\boldsymbol{c}^{(\mathrm{old})}$ and $\boldsymbol{c}^{(\mathrm{new})}$ in the following parametric form

$$\boldsymbol{c}(\theta) = \boldsymbol{c}^{(\mathrm{old})} + \theta \left(\boldsymbol{c}^{(\mathrm{new})} - \boldsymbol{c}^{(\mathrm{old})} \right), \ \theta \in [0, 1],$$

where θ is a parameter. This parametrization allows us to derive a path-following algorithm between arbitrary $\boldsymbol{c}^{(\mathrm{old})}$ and $\boldsymbol{c}^{(\mathrm{new})}$ by considering the change of the solutions when θ is moved from 0 to 1. This parametrization may also be interpreted as one-dimensional parametric programming with respect to the scalar θ, and our construction of the algorithm follows a similar line to one-dimensional regularization path algorithms. However, an important difference from one-dimensional regularization path algorithms is that the weight vector in high dimensional space is moved in our algorithm. We will later illustrate that the above parametrization covers various important machine learning problems and greatly widens the applicability of the path-following approach.

Suppose we are currently at $\boldsymbol{c}(\theta)$ on the path, and the current solution is $(b, \boldsymbol{\alpha})$. Let $\boldsymbol{\Delta c} = \Delta\theta \left(\boldsymbol{c}^{(\mathrm{new})} - \boldsymbol{c}^{(\mathrm{old})} \right)$ where the operator Δ represents the amount of change of each variable from the current value and $\Delta\theta \geq 0$. If $\Delta\theta$ is increased from 0, we may encounter a point at which some of the KKT conditions (3a)–(3c) do not hold. This can be checked by investigating the following conditions.

$$\begin{cases} y_i f(\boldsymbol{x}_i) + y_i \Delta f(\boldsymbol{x}_i) \geq 1, & i \in \mathcal{O}, \\ \alpha_i + \Delta\alpha_i > 0, & i \in \mathcal{M}, \\ \alpha_i + \Delta\alpha_i - (C_i + \Delta C_i,) < 0, & i \in \mathcal{M}, \\ y_i f(\boldsymbol{x}_i) + y_i \Delta f(\boldsymbol{x}_i) \leq 1, & i \in \mathcal{I}. \end{cases} \tag{9}$$

The set of inequalities (9) defines a convex polyhedron, called a *critical region* in the multi-parametric programming literature [13]. The event points lie on the border of critical regions, as illustrated in Figure 1.

We detect an event point by checking the conditions (9) along the solution path as follows. Using (7), we can express the changes of b and $\boldsymbol{\alpha}_{\mathcal{M}}$ as

$$\begin{bmatrix} \Delta b \\ \boldsymbol{\Delta\alpha}_{\mathcal{M}} \end{bmatrix} = \Delta\theta\boldsymbol{\phi}, \tag{10}$$

where $\boldsymbol{\phi} = -\boldsymbol{M}^{-1}[\boldsymbol{y}_{\mathcal{I}}\ \boldsymbol{Q}_{\mathcal{I},\mathcal{M}}]^\top\ (\boldsymbol{c}_{\mathcal{I}}^{(\mathrm{new})} - \boldsymbol{c}_{\mathcal{I}}^{(\mathrm{old})})$. Furthermore, $y_i \Delta f(\boldsymbol{x}_i)$ is expressed as

$$y_i \Delta f(\boldsymbol{x}_i) = \Delta\theta \psi_i, \tag{11}$$

where $\psi_i = [y_i\ \boldsymbol{Q}_{i,\mathcal{M}}]\ \boldsymbol{\phi} + \boldsymbol{Q}_{i,\mathcal{I}}(\boldsymbol{c}_{\mathcal{I}}^{(\mathrm{new})} - \boldsymbol{c}_{\mathcal{I}}^{(\mathrm{old})})$. Let us denote the elements of the index set $\mathcal{M}$ as $\mathcal{M} = \{m_1, \ldots, m_{|\mathcal{M}|}\}$. Substituting (10) and (11) into the inequalities (9), we can obtain the maximum step-length with no event occurrence. Based on the largest possible $\Delta\theta$, we can compute $\boldsymbol{\alpha}$ and b along the solution path by (10).

At the border of the critical region, we need to update the index sets $\mathcal{M}$, $\mathcal{O}$, and $\mathcal{I}$. For example, if α_i ($i \in \mathcal{M}$) reaches 0, we need to move the element i from $\mathcal{M}$ to $\mathcal{O}$. Then the above path-following procedure is carried out again for the next critical region specified by the updated index sets $\mathcal{M}$, $\mathcal{O}$, and $\mathcal{I}$, and this procedure is repeated until $\boldsymbol{c}$ reaches $\boldsymbol{c}^{(\mathrm{new})}$.

3.3. Computational Complexity

The computational complexity at each iteration of our path-following algorithm is the same as that for the ordinary SVM (i.e., the single-C formulation) [3]. Thus, our algorithm inherits a superior computational property of the original path-following algorithm.

The update of the linear system (10) from the previous one at each event point can be carried out efficiently with $O(|\mathcal{M}|^2)$ computational cost based on the *Cholesky decomposition rank-one update* [15] or the *block-matrix inversion formula*. Thus, the computational cost required for identifying the next event point is $O(n|\mathcal{M}|)$.

It is difficult to state the number of iterations needed for complete path-following because the number of events depends on the sensitivity of the model and the data set. Several empirical results suggested that the number of events linearly increases w.r.t. the data set size [3, 10]; our experimental analysis given in Section 4 also showed the same tendency. This implies that path-following is computationally highly efficient—indeed, in Section 4, we will experimentally demonstrate that the proposed path-following algorithm is faster than an alternative approach in one or two orders of magnitude.

4. EXPERIMENTS

In this section, we illustrate the empirical performance of the proposed WSVM path-following algorithm in three real-world applications. We compared the computational cost of the proposed path-following algorithm with the *sequential minimal optimization* (SMO) algorithm [16] when the instance weights of WSVM are changed in various ways. In particular, we investigated the CPU time of updating solutions from some $\boldsymbol{c}^{(\mathrm{old})}$ to $\boldsymbol{c}^{(\mathrm{new})}$.

In the path-following algorithm, we assume that the optimal parameter $\boldsymbol{\alpha}$ as well as the Cholesky factor $\boldsymbol{L}$ of $\boldsymbol{Q}_{\mathcal{M}}$ for $\boldsymbol{c}^{(\mathrm{old})}$ has already been obtained. In SMO, we used the old optimal $\boldsymbol{\alpha}$ as the initial starting point (i.e., the 'hot' start) after making them *feasible* using the *alpha-seeding strategy* [17]. We set the tolerance parameter in the termination criterion of SMO to 10^{-3}. Our implementation of SMO algorithm is based on *LIBSVM* [18]. To circumvent possible numerical instability, we added small positive constant 10^{-6} to the diagonals of the matrix $\boldsymbol{Q}$. In all the experiments, we used the Gaussian kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{x}'\|^2/p\right)$, where γ is a hyper-parameter and p is the dimensionality of $\boldsymbol{x}$.

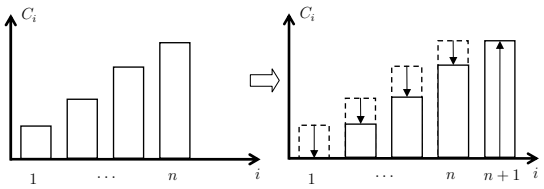

Fig. 2. A schematic illustration of the change of weights in time-series learning. The left plot shows the fact that larger weights are assigned to more recent instances. The right plot describes a situation where we receive a new instance ($i = n + 1$). In this situation, the oldest instance ($i = 1$) is deleted by setting its weight to zero, the weight of the new instance is set to be the largest, and the weights of the rest of the instances are decreased accordingly.

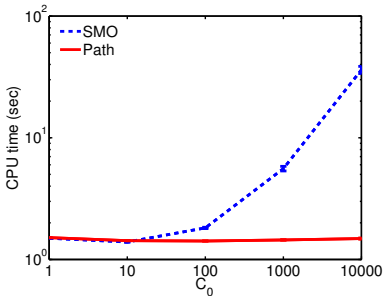

Fig. 3. CPU time comparison for online time-series learning using NASDAQ composite index.

4.1. On-line Time-series Learning

In online time-series learning, larger (resp. smaller) weights should be assigned to newer (resp. older) instances. For example, in [19], the following weight function is used: $C_i = C_0 \frac{2}{1+\exp(a - 2a \times \frac{i}{n})}$, where C_0 and a are hyper-parameters and the instances are assumed to be sorted along the time axis (the most recent instance is $i = n$). In online learning, we need to update parameters when new observations arrive, and all the weights must be updated accordingly (see Figure 2).

We investigated the computational cost of updating parameters when several new observations arrive. The experimental data are obtained from the *NASDAQ composite index* between January 2, 2001 and December 31, 2009. As [19], we transformed the original closing prices using the Relative Difference in Percentage (RDP) of the price and the exponential moving average (EMA). Our task is to predict whether the future average closing price increases or decreases from the current price (see [14] for details).

We have an initial set of training instances with size $n = 2515$. The inputs were normalized in $[0, 1]^p$, where p is the dimensionality of the input $\boldsymbol{x}$. We used the Gaussian kernel with $\gamma = 1$, and the weight parameter a in C_i was set to 3. We first trained WSVM using the initial set of instances. Then we added 5 instances to the previously trained WSVM and removed the oldest 5 instances by decreasing their weights to 0. This does not change the size of the training data set, but the entire weights need to be updated as illustrated in Figure 2. We iterated this process 5 times and compared the total computational costs of the path-following algorithm and SMO algorithm. For fair comparison, we cleared the cache of kernel values at each update before running the algorithms.

Figure 3 shows the average CPU time over 10 runs for $C_0 \in \{1, 10, 10^2, 10^3, 10^4\}$, showing that the path-following algorithm is much faster than SMO algorithm especially for large C_0.

4.2. Model Selection in Covariate Shift Adaptation

Covariate shift is a situation in supervised learning where the input distributions change between the training and test phases but the conditional distribution of outputs given inputs remains unchanged [20]. Under covariate shift, standard SVMs are biased, and the bias caused by covariate shift can be asymptotically canceled by weighting the loss function according to the *importance* (i.e., the ratio of training and test input densities).

Here, we apply importance-weighted SVMs to *brain-computer interfaces* (BCIs) [21]. A BCI is a system which allows for a direct communication from man to machine via brain signals. Strong non-stationarity effects have been often observed in brain signals between training and test sessions, which could be modeled as covariate shift [22]. We used the BCI datasets provided by the Berlin BCI group [23], containing 24 binary classification tasks. The input features are 4-dimensional preprocessed *electroencephalogram* (EEG) signals, and the output labels correspond to the 'left' and 'right' commands. The size of training datasets is around 500 to 1000, and the size of test datasets is around 200 to 300.

Although the importance-weighted SVM tends to have lower bias, it in turns has larger estimation variance than the ordinary SVM [20]. Thus, in practice, it is desirable to slightly 'flatten' the instance weights so that the trade-off between bias and variance is optimally controlled. Here, we changed the instance weights from the uniform values to the importance values using the proposed path-following algorithm, i.e., the instance weights were changed from $C_i^{(\mathrm{old})} = C_0$ to $C_i^{(\mathrm{new})} = C_0 \frac{p_{\mathrm{test}}(\boldsymbol{x}_i)}{p_{\mathrm{train}}(\boldsymbol{x}_i)}$, $i = 1, \ldots, n$. The importance values $\frac{p_{\mathrm{test}}(\boldsymbol{x}_i)}{p_{\mathrm{train}}(\boldsymbol{x}_i)}$ were estimated by the method proposed in [7], which directly estimates the density ratio without going through density estimation of $p_{\mathrm{test}}(\boldsymbol{x})$ and $p_{\mathrm{train}}(\boldsymbol{x})$.

For comparison, we ran SMO algorithm at (i) each breakpoint of the solution path, and (ii) 100 weight vectors taken uniformly in $[C_i^{(\mathrm{old})}, C_i^{(\mathrm{new})}]$. We used the Gaussian kernel and the inputs were normalized in $[0, 1]^p$, where p is the dimensionality of $\boldsymbol{x}$.

Figure 4 shows the average CPU time and its standard deviation. We examined several settings of hyper-parameters $\gamma = 1$ and $C_0 \in \{1, 10, 10^2, \ldots, 10^4\}$. The horizontal axis of each plot represents C_0. The graphs show that our path-following algorithm is faster than SMO algorithm in all cases. While SMO algorithm tended to take longer time for large C_0, the CPU time of the path-following algorithm did not increase with respect to C_0.

An important advantage of the path algorithm is that the path of the validation error can also be traced as well. In the case of 0-1 loss, since it changes discretely when the sign of $f(\boldsymbol{x})$ changes, the validation error is a piecewise-constant function of θ (see Figure 5). Monitoring piecewise linear change of $f(\boldsymbol{x})$, we can exactly detect the changes of the validation error during the path following and it is thus possible to find the exact minimizer of validation error with respect to $\theta \in [0, 1]$.

4.3. Heteroscedastic Regression

So far, we focused on classification scenarios. Here we apply the proposed path-following algorithm to an instance-weighted variant of the *support vector regression* (WSVR) (see [14] for its derivation).

As an application of WSVR, we consider a heteroscedastic regression problem, where the variance of output noise depends on input points. In heteroscedastic data modeling, larger (resp. smaller) weights are usually assigned to instances with smaller (resp. larger) variances. Since the point-wise variances are often unknown in practice, they should also be estimated from data. A standard approach is to alternately estimate the weight vector $\boldsymbol{c}$ based on the current WSVR solution and update the WSVR solutions based on the new weight vector $\boldsymbol{c}$ [24].

We set the weights as $C_i = C_0 \frac{\widehat{\sigma}}{|e_i|}$ where $e_i = y_i - \widehat{f}(\boldsymbol{x}_i)$ is the residual of the instance $(\boldsymbol{x}_i, y_i)$ from the current fit $\widehat{f}(\boldsymbol{x}_i)$, and $\widehat{\sigma}$ is an estimate of the common standard deviation of the noise computed as $\widehat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^n e_i^2}$. We employed the following procedure for the heteroscedastic data modeling:

Step1: Training WSVR with uniform weights (i.e., $C_i = C_0, i = 1, \ldots, n$.).

Step2: Update weights by $C_i = C_0 \frac{\widehat{\sigma}}{|e_i|}$ and update the solution of WSVR accordingly. Repeat this step until $\frac{1}{n}\sum_{i=1}^n |(e_i^{(\mathrm{old})} - e_i)/e_i^{(\mathrm{old})}| \leq 10^{-3}$ holds, where $e^{(\mathrm{old})}$ is the previous training error.

We investigated the computational cost of Step2. We applied the above procedure to the *Boston housing* data set [25]. The sample size is 506 and the number of features is $p = 13$. The inputs were normalized in $[-1, 1]^p$. We randomly sampled $n = 404$ instances from the original data set, and the experiments were repeated 10 times. We used the Gaussian kernel with $\gamma = 1$. The insensitive zone thickness in WSVR was fixed to $\varepsilon = 0.05$.

Figure 6 shows the CPU time comparison for $C_0 \in \{1, 10, \ldots, 10^4\}$, where Step2 is iterated about $20 \sim 50$ times. The graph shows that our path-following approach is faster than SMO algorithm especially for large C_0.

5. CONCLUSION

In this paper, we developed an efficient algorithm for updating solutions of instance-weighted SVMs. Our algorithm was built upon multi-parametric programming techniques, which—to the best of our knowledge—have never been studied in machine learning literature. We experimentally demonstrated the computational advantage of the proposed algorithm on a wide range of practical applications including on-line time-series analysis, covariate shift adaptation, and heteroscedastic data modeling.

6. REFERENCES

[1] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, Berlin, Germany, 1995.

[2] Y. Lin, Y. Lee, and G. Wahba, "Support vector machines for classification in nonstandard situations," *Machine Learning*, vol. 46, no. 1/3, pp. 191–202, 2002.

[3] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu, "The entire regularization path for the support vector machine," *Journal of Machine Learning Research*, vol. 5, pp. 1391–1415, 2004.

[4] M. J. Best, "An algorithm for the solution of the parametric quadratic programming problem," Tech. Rep. 82-24, Faculty of Mathematics, University of Waterloo, 1982.

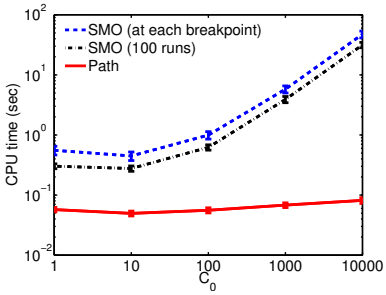

Fig. 4. CPU time comparison for covariate shift adaptation using BCI data.

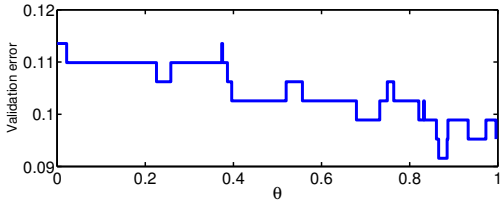

Fig. 5. An example of the validation error path for the BCI data ($C = 10$, $\gamma = 1$). The numbers of training and validation instances are 727 and 273, respectively. The minimum validation error is achieved at $\theta \simeq [0.866, 0.885]$.

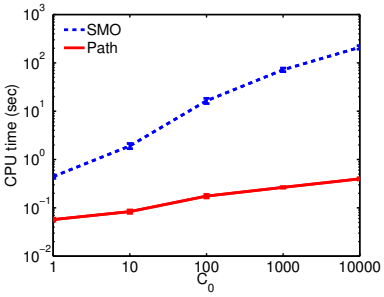

Fig. 6. CPU time comparison for heteroscedastic modeling using Boston housing data.

[5] F. R. Bach, D. Heckerman, and E. Horvitz, "Considering cost asymmetry in learning classifiers," *Journal of Machine Learning Research*, vol. 7, pp. 1713–1741, 2006.

[6] S. Rosset and J. Zhu, "Piecewise linear regularized solution paths," *Annals of Statistics*, vol. 35, pp. 1012–1030, 2007.

[7] T. Kanamori, S. Hido, and M. Sugiyama, "A least-squares approach to direct importance estimation," *Journal of Machine Learning Research*, vol. 10, pp. 1391–1445, Jul. 2009.

[8] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Advances in Neural Information Processing Systems 13*, Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, Eds., Cambridge, Massachussetts, 2001, vol. 13, pp. 409–415, The MIT Press.

[9] M. Karasuyama and I. Takeuchi, "Multiple incremental decremental learning of support vector machines," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 2009, pp. 907–915.

[10] G. Wang, D.-Y. Yeung, and F. H. Lochovsky, "A new solution path algorithm in support vector regression," *IEEE Transactions on Neural Networks*, vol. 19, no. 10, pp. 1753–1767, 2008.

[11] S. Rosset, "Bi-level path following for cross validated solution of kernel quantile regression," *Journal of Machine Learning Research*, vol. 10, pp. 2473–2505, 2009.

[12] I. Takeuchi, Q. V Le, T. D Sears, and A. J Smola, "Nonparametric quantile estimation," *Journal of Machine Learning Research*, vol. 7, pp. 1231–1264, 2006.

[13] E. N. Pistikopoulos, M. C. Georgiadis, and V. Dua, *Process Systems Engineering: Volume 1: Multi-Parametric Programming*, WILEY-VCH, 2007.

[14] M. Karasuyama, N. Harada, M. Sugiyama, and I. Takeuchi, "Multi-parametric solution-path algorithm for instance-weighted support vector machines," *CoRR, arXiv:1009.4791 [cs.LG]*, 2010.

[15] G. H. Golub and C. F. V. Loan, *Matrix computations*, Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[16] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds., Cambridge, MA, 1999, pp. 185–208, MIT Press.

[17] D. DeCoste and K. Wagstaff, "Alpha seeding for support vector machines," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2000, pp. 345–359.

[18] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," 2001, Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[19] L. J. Cao and F. E. H. Tay, "Support vector machine with adaptive parameters in financial time series forecasting," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1506–1518, 2003.

[20] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000.

[21] G. Dornhege, J. d. R. Millán, T. Hinterberger, D. McFarland, and K.-R. Müller, Eds., *Toward Brain Computer Interfacing*, MIT Press, Cambridge, MA, USA, 2007.

[22] M. Sugiyama, M. Krauledat, and K.-R. Müller, "Covariate shift adaptation by importance weighted cross validation," *Journal of Machine Learning Research*, vol. 8, pp. 985–1005, May 2007.

[23] W. Burde and B. Blankertz, "Is the locus of control of reinforcement a predictor of brain-computer interface performance?," in *Proceedings of the 3rd International Brain-Computer Interface Workshop and Training Course 2006*, Graz, Austria, 2006, Verlag der Technischen Universität.

[24] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard, "Most likely heteroscedastic Gaussian process regression," in *Proceedings of the 24th Annual International Conference on Machine Learning (ICML 2007)*, Z. Ghahramani, Ed. 2007, pp. 393–400, Omnipress.

[25] A. Asuncion and D. J. Newman, "UCI machine learning repository," `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2007.