Pattern Recognition 83 (2018) 196–208

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Chunk incremental learning for cost-sensitive hinge loss support vector machine

Bin Gu [a,b,c,*], Xin Quan [c], Yunhua Gu [c], Victor S. Sheng [d], Guansheng Zheng [c]

[a] Jiangsu Key Laboratory of Big Data Analysis Technology/B-DAT, Nanjing University of Information Science & Technology, Nanjing, China
[b] Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology, China
[c] School of Computer & Software, Nanjing University of Information Science & Technology, Nanjing, China
[d] Department of Computer Science, University of Central Arkansas, Conway, AR, USA

ARTICLE INFO

Article history:
Received 2 September 2017
Revised 11 March 2018
Accepted 21 May 2018
Available online 29 May 2018

Keywords:
Cost-sensitive learning
Chunk incremental learning
Hinge loss
Support vector machines

ABSTRACT

Cost-sensitive learning can be found in many real-world applications and represents an important learning paradigm in machine learning. The recently proposed cost-sensitive hinge loss support vector machine (CSHL-SVM) guarantees consistency with the cost-sensitive Bayes risk, and this technique provides better generalization accuracy compared to traditional cost-sensitive support vector machines. In practice, data typically appear in the form of sequential chunks, also called an on-line scenario. However, conventional batch learning algorithms waste a considerable amount of time under the on-line scenario due to re-training of a model from scratch. To make CSHL-SVM more practical for the on-line scenario, we propose a chunk incremental learning algorithm for CSHL-SVM, which can update a trained model without re-training from scratch when incorporating a chunk of new samples. Our method is efficient because it can update the trained model for not only one sample at a time but also multiple samples at a time. Our experimental results on a variety of datasets not only confirm the effectiveness of CSHL-SVM but also show that our method is more efficient than the batch algorithm of CSHL-SVM and the incremental learning method of CSHL-SVM only for a single sample.

1. Introduction

The objective function of most classification algorithms is to minimize the sum of a loss function on a training dataset and a regularization term to avoid overfitting. Conventional classification algorithms (e.g., standard support vector machine (SVM) [1] and guaranteed error machine (GEM) [2]) do not distinguish the loss of a false positive from that of a false negative in their objective functions. Specifically, these algorithms implicitly treat the costs of different types of mistakes equally. However, in many real-world problems, such as cancer diagnosis [3], credit card fraud detection [4], and terrorist detection [5], different types of mistakes often involve substantially different costs. We use cancer diagnosis as an example. Misdiagnosing a cancer patient as healthy is considerably more serious (or costly) than misdiagnosing a healthy patient as having cancer because the patient could lose his/her life due to a delay in treatment. Cost-sensitive learning takes unequal misclassification costs into consideration and can provide more reasonable predictions over conventional classification algorithms. Cost-sensitive learning is also considered to be a good solution for learning where the class distribution is highly imbalanced [6]. Thus, cost-sensitive learning [7] is an important learning paradigm in machine learning.

Following Vapnik's influential work on statistical learning theory [8], SVMs have received considerable interest because of their good generalization performance. Given a training set $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ and a hinge loss function $\ell(y_i, f(\boldsymbol{x}_i)) = \lfloor 1 - y_i f(\boldsymbol{x}_i) \rfloor_+$, where $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \{+1, -1\}$, $f(\cdot)$ is a discriminant function, and $\lfloor a \rfloor_+ = \max\{0, a\}$, a standard SVM [1] minimizes the sum of the hinge loss $\ell(y_i, f(\boldsymbol{x}_i))$ on $\mathcal{S}$ and an L_2-norm regularization term, with a regularization parameter C used to achieve a trade-off between these two objectives. The objective function of a standard SVM does not distinguish the loss of a false positive from that of a false negative. Therefore, a standard SVM is not a good choice (or not applicable in extreme cases) for cost-sensitive learning.

The machine learning community has proposed several SVM-based methods to effectively achieve cost-sensitive learning. For example, Karakoulas and Shawe-Taylor [9] proposed a boundary

* Corresponding author at: School of Computer & Software, Nanjing University of Information Science & Technology, Nanjing, China.
E-mail addresses: jsgubin@nuist.edu.cn (B. Gu), quanxin@nuist.edu.cn (X. Quan), yhgu_6655@163.com (Y. Gu), ssheng@uca.edu (V.S. Sheng), zgs@nuist.edu.cn (G.S. Zheng).

https://doi.org/10.1016/j.patcog.2018.05.023
0031-3203/

movement method (BM-SVM), which shifts the decision boundary by simply adjusting the threshold of the standard SVM. BM-SVM would be an optimal strategy under Bayesian decision theory if class posterior probabilities were available. However, class posterior probabilities are not available in real-world applications. To incorporate the unequal costs of different types of misclassifications into the objective function, Osuna [10] introduced two cost parameters, C_+ and C_-, into the hinge loss function to denote the costs of a false negative and those of a false positive, respectively, and thus proposed 2C-SVM. Davenport et al. [11] proposed 2ν-SVM, which is theoretically equivalent to 2C-SVM. As noted in [12], 2C-SVM degenerates into a standard SVM when a dataset is linearly separable or when the regularization parameters C_+ and C_- are large; this is also the case for 2ν-SVM. To address these limitations of 2C-SVM and 2ν-SVM, Masnadi et al. recently proposed a new cost-sensitive hinge loss support vector machine (CSHL-SVM) [12,13], which theoretically guarantees consistency with the cost-sensitive Bayes risk and has better generalization accuracy compared to the standard SVM, BM-SVM and 2C-SVM (or 2ν-SVM). Because of the aforementioned generalization ability, we primarily focus on CSHL-SVM in this paper.

In real-world machine learning applications, such as cancer diagnosis [3], credit card fraud detection [4], and terrorist detection [5], data typically appear in the form of a chunk sequentially, although in extreme cases, data appear one example at a time. This is called an on-line scenario. Batch learning algorithms normally train a model after all training samples are given at once. If some new samples are added to the original training samples, then the batch learning algorithms need to re-train the model from scratch. Thus, batch learning algorithms would not be sufficiently efficient in on-line scenarios. However, incremental learning algorithms are more capable in on-line scenarios because the advantage of incremental learning algorithms is that they allow the incorporation of additional training data without re-training from scratch [14].

As mentioned above, the recently proposed CSHL-SVM is an important cost-sensitive learning algorithm because it guarantees consistency with the cost-sensitive Bayes risk and has a better generalization accuracy than traditional cost-sensitive SVMs. However, to the best of our knowledge, no incremental learning algorithms have been proposed for CSHL-SVM, but many incremental learning algorithms have been proposed for other SVMs. These incremental algorithm can be divided into two parts, i.e., approximate algorithms and exact algorithms. We will discuss them respectively as follows.

There have been several approximate incremental algorithms provided for SVMs. For example, Bordes et al. [15] proposed an online algorithm based on the sequential minimal optimization algorithm for standard SVM. Shalev-Shwartz et al. [16] proposed an stochastic subgradient algorithm for SVMs. Karampatziakis and Langford [17] proposed an online importance weight aware updating rule. Lacoste-Julien et al. [18] proposed an block-coordinate Frank–Wolfe algorithm for solving structural SVMs. Shalev-Shwartz and Zhang [19] proposed a stochastic dual coordinate ascent method for solving linear SVM. All these algorithms provide the approximate incremental algorithms for SVMs. However, this paper will focus on the exact incremental algorithms as discussed below.

There also have been several exact incremental algorithms provided for SVMs. For example, Cauwenberghs and Poggio [20] proposed an incremental learning algorithm for the standard SVM in 2001. Later, Martin [21] extended this algorithm to support vector regression. Laskov et al. [14] proposed an accurate incremental learning algorithm for one-class SVM. Gu et al. [22] and Gu et al. [23] proposed incremental learning algorithms for ν-support vector classification and ν-support vector regression, respectively. Gu et al. [24,25] proposed incremental learning algorithms for support vector ordinal regression. All these incremental learning algorithms can only incorporate one sample at a time (called single incremental learning). Thus, if a chunk of samples is added, where the size of the chunk is greater than one, we need to run these single incremental learning algorithms multiple times. If the entire chunk of samples can be incorporated at one time, then such a technique would be more efficient than the single incremental learning algorithms. In this paper, we call the incremental learning algorithms that incorporate a chunk at a time chunk incremental learning. To the best of our knowledge, the only work on chunk incremental learning is [26], which can handle multiple samples simultaneously during incremental learning. Thus, it is highly desirable to design a chunk incremental learning algorithm for CSHL-SVM.

Although there also have some

In this paper, we propose a chunk incremental learning algorithm for CSHL-SVM (CICSHL-SVM) that can update a trained model without re-training from scratch when incorporating a chunk of new samples.[1] Our CICSHL-SVM can update the trained model for not only a chunk of new samples at a time but also for one sample at a time. More importantly, CICSHL-SVM is more efficient than the batch algorithm of CSHL-SVM and the single incremental learning algorithm because CICSHL-SVM can incorporate the entire chunk of samples at one time. Our experimental results on a variety of datasets confirm the effectiveness of CSHL-SVM and show that CICSHL-SVM is more efficient than the batch algorithm of CSHL-SVM and the single incremental learning method of CSHL-SVM.

The remainder of this paper is organized as follows. In Section 2, we present CSHL-SVM and provide its Karush–Kuhn–Tucker (KKT) conditions. In Section 3, we propose our chunk incremental learning algorithm for CSHL-SVM. Our experimental setup, results, and discussion are presented in Section 4. The final section provides some concluding remarks.

2. CSHL-SVM And its KKT conditions

In this section, we first present the formulation of CSHL-SVM, and then, we provide its KKT conditions.

2.1. CSHL-SVM

As mentioned in Section 1, the standard hinge loss function does not distinguish the loss of a false positive from that of a false negative. To address this issue, Masnadi-Shirazi and Vasconcelos [12] proposed the following cost-sensitive hinge loss function.

$$\ell_{CS}(y_i, f(x_i)) = \begin{cases} \lfloor C_+ - C_+ f(\boldsymbol{x}_i) \rfloor_+ & \text{if } y = 1 \\ \lfloor 1 + (2C_- - 1) f(\boldsymbol{x}_i) \rfloor_+ & \text{if } y = -1 \end{cases} \tag{1}$$

where $C_- \geq 1$ and $C_+ \geq 2C_- - 1$.

Let $\mathcal{S}^+ = \{(\boldsymbol{x}_i, y_i) \in \mathcal{S} : y_i = +1\}$ and $\mathcal{S}^- = \{(\boldsymbol{x}_i, y_i) \in \mathcal{S} : y_i = -1\}$. The discriminant function $f(\boldsymbol{x})$ is defined as $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \phi(\boldsymbol{x}) \rangle + b$, where training samples $\boldsymbol{x}_i$ are mapped into a high-dimensional reproducing kernel Hilbert space (RKHS) [27] by a transformation function ϕ, $\langle \cdot, \cdot \rangle$ denotes an inner product in RKHS, and b is the threshold of a discrimination hyperplane. Based on the cost-sensitive hinge loss function (1), CSHL-SVM considers the following primal formulation:

$$\begin{aligned} \min_{\boldsymbol{w}, b, \xi_i} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C \cdot \left(\beta \sum_{i \in S_+} \xi_i + \lambda \sum_{i \in S_-} \xi_i \right) \\ \text{s.t.} \quad & \boldsymbol{w}^T \phi(\boldsymbol{x}_i) + b \geq 1 - \xi_i, \quad \forall i \in S_+ \end{aligned} \tag{2}$$

[1] Because our CICSHL-SVM is an exact incremental algorithms which can return an exact solution of CSHL-SVM, the solution of CICSHL-SVM obtained according to the chunk of samples is also an exact solution of CSHL-SVM obtained according to all the samples including the new added chunk of samples.

$$\boldsymbol{w}^T\phi(\boldsymbol{x}_i)+b\le-\kappa+\xi_i,\quad \forall i\in S_-$$
$$\xi_i\ge 0,\quad \forall i\in S$$

where $\beta=C_+,\quad \lambda=2C_--1,\ \kappa=\frac{1}{2C_--1}$, and ξ_i is a non-negative slack variable corresponding to the sample $(\boldsymbol{x}_i, y_i)$.

For the primal formulation Eq. (2) of CSHL-SVM, the dual problem can be presented as follows:

$$\begin{aligned}\min_{\boldsymbol{\alpha}}\ & \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\alpha}-\boldsymbol{\alpha}^T\boldsymbol{\Omega}\\ \text{s.t.}\ & \boldsymbol{\alpha}^T\boldsymbol{y}=0\\ & 0\le\alpha_i\le\boldsymbol{C}_i,\quad \forall i\in S\end{aligned}\tag{3}$$

where α is the dual variable, $\boldsymbol{H}$ is a positive semidefinite matrix with $\boldsymbol{H}_{ij}=y_iy_jK(x_i,x_j)$, $K(x_i,x_j)=\langle\phi(x_i),\phi(x_j)\rangle$ is a kernel function, and vectors $\boldsymbol{C}$ and $\boldsymbol{\Omega}$ with length n are defined as follows:

$$\boldsymbol{C}_i=\begin{cases}CC_+, & \text{if}\ \ i\in S_+\\ C(2C_--1), & \text{if}\ \ i\in S_-\end{cases}\tag{4}$$

$$\boldsymbol{\Omega}_i=\begin{cases}1, & \text{if}\ \ i\in S_+\\ \frac{1}{2C_--1}, & \text{if}\ \ i\in S_-\end{cases}.\tag{5}$$

As proven in [12], CSHL-SVM Eq. (3) theoretically guarantees consistency with the cost-sensitive Bayes risk and achieves better generalization accuracy compared to traditional cost-sensitive SVMs.

2.2. KKT Conditions

According to convex optimization theory [28], the solution of the dual problem Eq. (3) can be obtained by solving the following min-max problem:

$$\min_{0\le\alpha\le\boldsymbol{C}}\max_{b'}W=\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\alpha}-\boldsymbol{\alpha}^T\boldsymbol{\Omega}+b'\boldsymbol{\alpha}^T\boldsymbol{y}\tag{6}$$

where $b'\in\mathbb{R}$ is a Lagrangian multiplier corresponding to the equality constraint in Eq. (3). Based on Eq. (3) and (6), we know that the optimal discriminant function can be formulated as follows:

$$f(\boldsymbol{x})=\sum_{i\in\mathcal{S}}\alpha_iy_iK(\boldsymbol{x},\boldsymbol{x}_i)+b'\tag{7}$$

where the Lagrangian multiplier b' is actually the threshold of the discrimination hyperplane.

According to the KKT theorem (cf. page 244 of [28]), the first-order derivative of W leads to the following KKT conditions:

$$\forall i\in S:\ \frac{\partial W}{\partial\alpha_i}=\sum_{j\in\mathcal{S}}\alpha_j\boldsymbol{H}_{ij}+y_ib'\begin{cases}y_if(\boldsymbol{x}_i)>\boldsymbol{\Omega}_i & \text{for}\quad \alpha_i=0\\ y_if(\boldsymbol{x}_i)=\boldsymbol{\Omega}_i & \text{for}\quad 0\le\alpha_i\le\boldsymbol{C}_i\\ y_if(\boldsymbol{x}_i)<\boldsymbol{\Omega}_i & \text{for}\quad \alpha_i=\boldsymbol{C}_i\end{cases}\tag{8}$$

$$\frac{\partial W}{\partial b'}=\sum_{i\in\mathcal{S}}y_i\alpha_i=0\tag{9}$$

According to the value of $y_if(\boldsymbol{x}_i)$ in Eq. (8), the training sample set $\mathcal{S}$ is partitioned into three independent sets:

$$\mathcal{M}=\{i:y_if(\boldsymbol{x}_i)=\boldsymbol{\Omega}_i,0\le\alpha_i\le\boldsymbol{C}_i\}\tag{10}$$

$$\mathcal{I}=\{i:y_if(\boldsymbol{x}_i)<\boldsymbol{\Omega}_i,\alpha_i=\boldsymbol{C}_i\}\tag{11}$$

$$\mathcal{O}=\{i:y_if(\boldsymbol{x}_i)>\boldsymbol{\Omega}_i,\alpha_i=0\}\tag{12}$$

From Eqs. (10)–(12), it is easy to see the following:

1. All samples in $\mathcal{M}$ are strictly on the margins of CSHL-SVM. They are called *margin support vectors*.
2. All samples in $\mathcal{I}$ exceed the margins of CSHL-SVM. They are called *error support vectors*.
3. All samples in $\mathcal{O}$ are ignored by the margins of CSHL-SVM. They are called *remaining vectors*.

Notations: We define some notations that will be used later. We use $\boldsymbol{v}_{\mathcal{M}}$ to denote a subvector of the vector $\boldsymbol{v}$ and whose elements are indexed by the set $\mathcal{M}$. Similarly, a matrix $\boldsymbol{H}_{\mathcal{M},\mathcal{I}}$ denotes the submatrix of $\boldsymbol{H}$, whose rows are indexed by the set $\mathcal{M}$ and columns are indexed by the set $\mathcal{I}$. If the rows and columns of a matrix are indexed by the same set, such as $\boldsymbol{H}_{\mathcal{M},\mathcal{M}}$, then we use $\boldsymbol{H}_{\mathcal{M}}$ to simplify it.

3. Chunk incremental learning of CSHL-SVM

When a chunk of samples $\mathcal{A}=\{(\boldsymbol{x}_{n+1},y_{n+1}),\ldots,(\boldsymbol{x}_{n+m},y_{n+m})\}$ is added into the original training set $\mathcal{S}$, incremental learning algorithms need to update the solution of Eq. (3) the extended training set $\mathcal{S}\cup\mathcal{A}$ based on the solution on the original training set $\mathcal{S}$. This process is in stark contrast to batch learning algorithms, which rebuild the solution for the extended training set from scratch. As mentioned in Section 1, single incremental learning algorithms only incorporate one sample at a time. Thus, if the size of $\mathcal{A}$ is greater than one, then we need to run a single incremental learning algorithm multiple times. If the entire chunk of samples can be incorporated at one time, then the process would be more efficient. In this section, we propose our chunk incremental learning algorithm: CICSHL-SVM. Specifically, we first provide a sketch of CICSHL-SVM, and then, we provide detailed descriptions for the three main steps of CICSHL-SVM in Sections 3.1, 3.2 and 3.3.

Initially, we set all the weights of the samples in $\mathcal{A}$ to zero, i.e., $\alpha_{\mathcal{A}}=\boldsymbol{0}$. This is a simple but effective setting point to make all the samples satisfy the equality constraint Eq. (9) of the KKT conditions. In this setting, if existing samples $(\boldsymbol{x}_i, y_i)$ with $y_if(\boldsymbol{x}_i)>\Omega_i$ in $\mathcal{A}$, then these samples satisfy constraint Eq. (8) of the KKT conditions. Thus, we directly move these samples from the set $\mathcal{A}$ into the set $\mathcal{O}$ because they satisfy the KKT conditions Eq. (8) and (9). Similarly, if existing samples $(\boldsymbol{x}_i, y_i)$ with $y_if(\boldsymbol{x}_i)=\Omega_i$ in $\mathcal{A}$, then we directly move these samples from the set $\mathcal{A}$ into the set $\mathcal{M}$. Thus, the key of chunk incremental learning algorithms is to find an effective method to simultaneously update all the weights of the samples in $\mathcal{A}$ without re-training from scratch to make all the samples satisfy the KKT conditions.

We use $\Delta\boldsymbol{\alpha}_{\mathcal{A}}$ to represent the changes in the weights of $\mathcal{A}$, and we set the direction $\Delta\boldsymbol{\alpha}_{\mathcal{A}}$ as $\boldsymbol{C}_{\mathcal{A}}-\boldsymbol{\alpha}_{\mathcal{A}}$. Thus, we have $\Delta\boldsymbol{\alpha}_{\mathcal{A}}=\eta(\boldsymbol{C}_{\mathcal{A}}-\boldsymbol{\alpha}_{\mathcal{A}})$, where η is a parameter with $0\le\eta\le1$ to control the adjustment quality of $\boldsymbol{\alpha}_{\mathcal{A}}$. Correspondingly, we use $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$ to represent the changes in the weights of $\mathcal{S}$. Thus, the first question is the following: What is the direction $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$ with respect to η? Note that the key to computing the direction $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$ is to keep all the samples in $\mathcal{S}$ satisfying the KKT conditions during the adjustment. After computing the direction $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$, the next question is the following: What is the maximum adjustment quality $\eta^{\max}$ of η such that the KKT conditions will not be satisfied if η exceeds $\eta^{\max}$. After finding the maximum adjustment quality $\eta^{\max}$, we need to update $\boldsymbol{\alpha}_{\mathcal{A}}$, $\boldsymbol{\alpha}_{\mathcal{S}}$, b', $\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$ and $\mathcal{A}$. This procedure is repeated until all samples in $\mathcal{S}\cup\mathcal{A}$ satisfy the KKT conditions (i.e., $\mathcal{A}=\emptyset$), which derives our CICSHL-SVM (i.e., Algorithm 1).

As mentioned above, the three main steps of CICSHL-SVM are as follows.

1. computing the direction of $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$ (see line 3 of Algorithm 1);
2. computing the maximum adjustment quality $\eta^{\max}$ (see line 4 of Algorithm 1);

Algorithm 1 Chunk Incremental Algorithm for CSHL-SVM (CICSHL-SVM).

Input: $\boldsymbol{\alpha}_{\mathcal{S}}$, b',$\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$ and $\mathcal{A}$
Output: $\boldsymbol{\alpha}_{\mathcal{S}\cup\mathcal{A}}$, b',$\mathcal{O}$, $\mathcal{M}$ and $\mathcal{I}$.
1: Initialize $\alpha_{\mathcal{A}} = \mathbf{0}$, $\eta = 0$
2: **while** $\mathcal{A} \neq \emptyset$ **do**
3: Compute the direction of $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$ according to (17).
4: Compute the maximum adjustment quality $\eta^{\max}$ according to (20).
5: Update $\begin{bmatrix} b & \boldsymbol{\alpha}_{\mathcal{M}}^T \end{bmatrix} \leftarrow \begin{bmatrix} b & \boldsymbol{\alpha}_{\mathcal{M}}^T \end{bmatrix} + \eta\boldsymbol{\varphi}$
6: Update $\boldsymbol{\alpha}_{\mathcal{A}} \leftarrow \boldsymbol{\alpha}_{\mathcal{A}} + \eta(\boldsymbol{C}_{\mathcal{A}} - \boldsymbol{\alpha}_{\mathcal{A}})$
7: Update the sets $\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$ and $\mathcal{A}$ depending on which boundary is reached
8: **end while**

3. and updating $\boldsymbol{\alpha}_{\mathcal{A}}$, $\boldsymbol{\alpha}_{\mathcal{S}}$, b', $\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$ and $\mathcal{A}$ (see lines 5–7 of Algorithm 1).

In the following subsections, we provide detailed descriptions for the three steps.

3.1. Computing the direction of $\Delta\alpha_{\mathcal{S}}$

As mentioned above, the key aspect during the adjustment of $\boldsymbol{\alpha}_{\mathcal{A}}$ is to keep all the samples in $\mathcal{S}$ satisfying the KKT conditions. According to the KKT conditions Eq. (8) and (9), it is easy to find that $\boldsymbol{\alpha}_{\mathcal{O}}$ and $\boldsymbol{\alpha}_{\mathcal{I}}$ are fixed on the bounds $\mathbf{0}$ and $\boldsymbol{C}_{\mathcal{I}}$, respectively. Only the weights $\boldsymbol{\alpha}_{\mathcal{M}}$ and the Lagrange multiplier b' can be adjusted when changing $\boldsymbol{\alpha}_{\mathcal{A}}$. Thus, from Eq. (9) and (10) in the KKT conditions, we have the following linear system:

$$\sum_{j\in\mathcal{A}} \boldsymbol{H}_{ij}\Delta\alpha_j + \sum_{j\in\mathcal{M}} \boldsymbol{H}_{ij}\Delta\alpha_j + y_i\Delta b' = 0, \quad i \in \mathcal{M} \tag{13}$$

$$\sum_{j\in\mathcal{A}} y_j\Delta\alpha_j + \sum_{j\in\mathcal{M}} y_j\Delta\alpha_j = 0 \tag{14}$$

We can represent the above linear system in the following matrix form:

$$\begin{bmatrix} 0 & \boldsymbol{y}_{\mathcal{M}}^T \\ \boldsymbol{y}_{\mathcal{M}} & \boldsymbol{H}_{\mathcal{M}} \end{bmatrix} \begin{bmatrix} \Delta b' \\ \Delta\boldsymbol{\alpha}_{\mathcal{M}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{y}_{\mathcal{A}}^T \\ \boldsymbol{H}_{\mathcal{M},\mathcal{A}} \end{bmatrix} \Delta\boldsymbol{\alpha}_{\mathcal{A}} = 0 \tag{15}$$

As previously mentioned, we set the direction of $\Delta\boldsymbol{\alpha}_{\mathcal{A}}$ as $\boldsymbol{C}_{\mathcal{A}} - \boldsymbol{\alpha}_{\mathcal{A}}$. Thus, the linear system Eq. (15) can be simplified as follows:

$$\underbrace{\begin{bmatrix} 0 & \boldsymbol{y}_{\mathcal{M}}^T \\ \boldsymbol{y}_{\mathcal{M}} & \boldsymbol{H}_{\mathcal{M}} \end{bmatrix}}_{\tilde{H}} \begin{bmatrix} \Delta b' \\ \Delta\boldsymbol{\alpha}_{\mathcal{M}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{y}_{\mathcal{A}}^T \\ \boldsymbol{H}_{\mathcal{M},\mathcal{A}} \end{bmatrix} (\boldsymbol{C}_{\mathcal{A}} - \boldsymbol{\alpha}_{\mathcal{A}})\eta = 0 \tag{16}$$

Let $\boldsymbol{\varphi}$ denote $\begin{bmatrix} \frac{\Delta b'}{\eta} \\ \frac{\Delta\boldsymbol{\alpha}_{\mathcal{M}}}{\eta} \end{bmatrix}$ (i.e., the directions of $\Delta b'$ and $\Delta\boldsymbol{\alpha}_{\mathcal{M}}$ w.r.t. η); then, the linear system Eq. (16) can be rewritten as follows:

$$\underbrace{\begin{bmatrix} 0 & \boldsymbol{y}_{\mathcal{M}}^T \\ \boldsymbol{y}_{\mathcal{M}} & \boldsymbol{H}_{\mathcal{M}} \end{bmatrix}}_{\tilde{\boldsymbol{H}}} \boldsymbol{\varphi} = -\begin{bmatrix} \boldsymbol{y}_{\mathcal{A}}^T \\ \boldsymbol{H}_{\mathcal{M},\mathcal{A}} \end{bmatrix} (\boldsymbol{C}_{\mathcal{A}} - \boldsymbol{\alpha}_{\mathcal{A}}) \tag{17}$$

If $\boldsymbol{\varphi}$ is obtained, then we actually know the direction of $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$ because the directions $\Delta\boldsymbol{\alpha}_{\mathcal{O}}$ and $\Delta\boldsymbol{\alpha}_{\mathcal{I}}$ are both $\mathbf{0}$. Thus, the key to computing the direction of $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$ is to compute $\boldsymbol{\varphi}$ by solving the linear system (17).

The traditional approach to solving the linear system (17) is to directly compute the matrix inverse of $\tilde{\boldsymbol{H}}$ [14,20–22]. As mentioned in [29,30], the training samples may be linearly dependent in the RKHS. Thus, the key matrix $\tilde{\boldsymbol{H}}$ will encounter singularities. Here, we present a robust approach for computing $\boldsymbol{\varphi}$ without directly computing the inverse of $\tilde{\boldsymbol{H}}$. According to the QR decomposition with column pivoting [31], the matrix $\tilde{\boldsymbol{H}}$ can be decomposed as $\tilde{\boldsymbol{H}}P = \boldsymbol{Q}\boldsymbol{R}$, where $\boldsymbol{Q}$ is an orthogonal matrix, $\boldsymbol{R}$ is an upper triangular matrix with diagonal elements in decreasing order, and $\boldsymbol{P}$ is a permutation matrix that, when right-multiplied against $\tilde{\boldsymbol{H}}$, reorders the columns of $\tilde{\boldsymbol{H}}$. Thus, we can have the following linear system according to Eq. (17), and we solve it with $\boldsymbol{z}$.

$$\boldsymbol{Q}\underbrace{\boldsymbol{R}P\boldsymbol{\varphi}}_{\boldsymbol{z}} = -\begin{bmatrix} \boldsymbol{y}_{\mathcal{A}}^T \\ \boldsymbol{Q}_{\mathcal{M},\mathcal{A}} \end{bmatrix} (\boldsymbol{C}_{\mathcal{A}} - \boldsymbol{\alpha}_{\mathcal{A}}) \tag{18}$$

It is easy to solve $\boldsymbol{z}$ because $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^T$. After obtaining $\boldsymbol{z}$, we can use $\boldsymbol{R}\boldsymbol{g} = \boldsymbol{z}$ to obtain $\boldsymbol{g}$, which is also easily solved by backward substitution, therein exploiting the upper triangular structure of $\boldsymbol{R}$. Finally, $\boldsymbol{\varphi} = \boldsymbol{P}\boldsymbol{g}$, which is a permutation on $\boldsymbol{g}$.

After obtaining the direction of $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$, according to Eq. (13), the linear relationship between $\Delta f(\boldsymbol{x}_{\mathcal{S}\cup\mathcal{A}})$ and η can also be obtained as follows:

$$\Delta f(\boldsymbol{x}_{\mathcal{S}\cup\mathcal{A}}) = \left(\begin{bmatrix} \boldsymbol{y}_{\mathcal{S}\cup\mathcal{A}} & \boldsymbol{Q}_{\mathcal{S}\cup\mathcal{A},\mathcal{M}} \end{bmatrix}\boldsymbol{\varphi} + \boldsymbol{Q}_{\mathcal{S}\cup\mathcal{A},\mathcal{A}}(\boldsymbol{C}_{\mathcal{A}} - \boldsymbol{\alpha}_{\mathcal{A}})\right) \cdot \eta \stackrel{\text{def}}{=} \boldsymbol{\psi}\eta \tag{19}$$

3.2. Computing the maximum adjustment quality $\eta^{\max}$

Based on the direction of $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$ and the linear relationship between $\Delta f(\boldsymbol{x}_{\mathcal{S}\cup\mathcal{A}})$ and η, we can compute the maximum adjustment quality $\eta^{\max}$ such that the KKT conditions will not be satisfied if η exceeds $\eta^{\max}$. There are four cases that need to be considered for computing the maximum adjustment quality $\eta^{\max}$. In the following, we present a detailed description for each case.

1. A certain α_i in $\mathcal{M}$ reaches a bound (an upper or lower bound). Thus, we can compute the maximal possible $\eta^{\mathcal{M}}$ before a certain sample in $\mathcal{M}$ moves to $\mathcal{O}$ or $\mathcal{L}$ according to the constraints $0 \le \alpha_i + \boldsymbol{\varphi}_j\eta \le \boldsymbol{C}_i$, $\forall i \in \mathcal{M}$ in the KKT conditions.
2. A certain $y_i f(\boldsymbol{x}_i)$ in $\mathcal{O}$ reaches $\boldsymbol{\Omega}_i$. Thus, we can compute the maximal possible $\eta^{\mathcal{O}}$ before a certain sample in $\mathcal{O}$ moves to $\mathcal{M}$ according to the constraints $y_i\left(f(\boldsymbol{x}_i) + \boldsymbol{\psi}_i\eta\right) > \boldsymbol{\Omega}_i$, $\forall i \in \mathcal{O}$ in the KKT conditions.
3. A certain $y_i f(\boldsymbol{x}_i)$ in $\mathcal{I}$ reaches $\boldsymbol{\Omega}_i$. Thus, we can compute the maximal possible $\eta^{\mathcal{I}}$ before a certain sample in $\mathcal{I}$ moves to $\mathcal{M}$ according to the constraints $y_i\left(f(\boldsymbol{x}_i) + \boldsymbol{\psi}_i\eta\right) < \boldsymbol{\Omega}_i$, $\forall i \in \mathcal{I}$ in the KKT conditions.
4. A certain α_i or $y_i f(\boldsymbol{x}_i)$ in $\mathcal{A}$ reaches a bound ($\boldsymbol{C}_i$ or $\boldsymbol{\Omega}_i$). Thus, we can compute the maximal possible $\eta^{\mathcal{A}}$ before a certain sample in $\mathcal{A}$ moves to $\mathcal{M}$ or $\mathcal{I}$ according to the constraints $y_i\left(f(\boldsymbol{x}_i) + \boldsymbol{\psi}_i\eta\right) < \boldsymbol{\Omega}_i$, $\forall i \in \mathcal{A}$ and $\eta \le 1$ in the KKT conditions.

Thus, the smallest of the four values $\left(\eta^{\mathcal{M}}, \eta^{\mathcal{O}}, \eta^{\mathcal{I}}, \eta^{\mathcal{A}}\right)$ will be the maximal adjustment quantity of $\eta^{\max}$. Specifically,

$$\eta^{max} = \min\left\{\eta^{\mathcal{M}}, \eta^{\mathcal{O}}, \eta^{\mathcal{I}}, \eta^{\mathcal{A}}\right\} \tag{20}$$

3.3. Update $\alpha_{\mathcal{A}}$, $\alpha_{\mathcal{S}}$, b′, $\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$ and $\mathcal{A}$

Once we obtain the maximum adjustment quality $\eta^{\max}$, we should update $\boldsymbol{\alpha}_{\mathcal{M}}$, $\boldsymbol{\alpha}_{\mathcal{A}}$ and b' for the next iteration. Specifically, $\boldsymbol{\alpha}_{\mathcal{M}}$, $\boldsymbol{\alpha}_{\mathcal{A}}$ and b' are updated as follows:

$$\boldsymbol{\alpha}_{\mathcal{A}} \leftarrow \boldsymbol{\alpha}_{\mathcal{A}} + (\boldsymbol{C}_{\mathcal{A}} - \boldsymbol{\alpha}_{\mathcal{A}})\eta^{max} \tag{21}$$

$$\boldsymbol{\alpha}_{\mathcal{M}} \leftarrow \boldsymbol{\alpha}_{\mathcal{M}} + \boldsymbol{\varphi}_{\mathcal{M}}\eta^{max} \tag{22}$$

$$b' \leftarrow b' + \boldsymbol{\varphi}_{b'}\eta^{max} \tag{23}$$

Note that all the samples in $\mathcal{S}$ still satisfy the KKT conditions under the updated $\boldsymbol{\alpha}_{\mathcal{A}}$, $\boldsymbol{\alpha}_{\mathcal{S}}$, and b'. Let $\eta^{[k]}$ denote the value of η for

Table 1
The computational complexities of CICSHL-SVM, where s is the number of iterations in CICSHL-SVM.

Step	Description	Complexity
1	Line 3 of Algorithm 1	$O\big((\lvert\mathcal{M}\rvert+1)^2\big)$
2	Line 4 of Algorithm 1	$O(n+m)$
3	Lines 5–7 of Algorithm 1	$O(\lvert\mathcal{M}\rvert+\lvert\mathcal{A}\rvert)$
Overall	CICSHL-SVM	$O(((\lvert\mathcal{M}\rvert+1)^2+n+m)s)$

the last round. For all $\eta\in[\eta^{[k]},\eta+\eta^{\max}]$, $\boldsymbol{\alpha}_{\mathcal{M}}$, $\boldsymbol{\alpha}_{\mathcal{A}}$ and b' can be updated as Eqs. (21)–(23).

After the maximum adjustment quality $\eta^{\max}$ is calculated, we should update the sets $\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$ and $\mathcal{A}$ for the next iteration. Because there must be a sample yielding the minimum in Eq. (20) once the maximum adjustment quality $\eta^{\max}$ is obtained, we use an index t to represent this sample. Thus, the sets $\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$ and $\mathcal{A}$ can be updated as follows.

1. If $t\in\mathcal{M}$ and $\boldsymbol{\varphi}_t>0$, t should be moved from $\mathcal{M}$ to $\mathcal{L}$.
2. If $t\in\mathcal{M}$ and $\boldsymbol{\varphi}_t<0$, t should be moved from $\mathcal{M}$ to $\mathcal{O}$.
3. If $t\in\mathcal{O}$, t should be moved from $\mathcal{O}$ to $\mathcal{M}$.
4. If $t\in\mathcal{I}$, t should be moved from $\mathcal{I}$ to $\mathcal{M}$.
5. If $t\in\mathcal{A}$ and $y_t f(\boldsymbol{x}_t)=\boldsymbol{\Omega}_t$, t should be moved from $\mathcal{A}$ to $\mathcal{M}$.
6. If $t\in\mathcal{A}$ and $y_t f(\boldsymbol{x}_t)<\boldsymbol{\Omega}_t$, t should be moved from $\mathcal{A}$ to $\mathcal{I}$.

3.4. Computational complexity analysis of CICSHL-SVM

As mentioned above, CICSHL-SVM is an iterative algorithm. Assume that the number of iterations in CICSHL-SVM is s. In Section 5.1, our experimental results empirically show that CICSHL-SVM does achieve fast convergence for each iteration of the three above-mentioned steps. To analyze the computational complexity of CICSHL-SVM, we first provide the computational complexity analysis of the three steps as follows.

1. The first (also the main) step is to compute the direction of $\Delta\boldsymbol{\alpha}_{\mathcal{S}}$, which requires solving the system of Eq. (17) using QR decomposition. Normally, QR decomposition involves $O\big((\lvert\mathcal{M}\rvert+1)^3\big)$ computations [32]. However, because the change in $\widetilde{H}$ is slight (either adding or deleting a row and a column from $\widetilde{H}$), the QR decomposition can be updated without recomputing the QR decomposition from scratch [32]. The computational complexity of the updating is $O\big((\lvert\mathcal{M}\rvert+1)^2\big)$. Thus, the computational complexity for the first step can be reduced to $O\big((\lvert\mathcal{M}\rvert+1)^2\big)$.
2. The second step is to compute the maximum adjustment quality $\eta^{\max}$. According to Section 3.2, we have that the computational complexity for the second step is $O(n+m)$.
3. The third step is to update $\boldsymbol{\alpha}_{\mathcal{A}}$, $\boldsymbol{\alpha}_{\mathcal{S}}$, b', $\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$ and $\mathcal{A}$. The corresponding computational complexity is $O(\lvert\mathcal{M}\rvert+\lvert\mathcal{A}\rvert)$.

Thus, the total computational complexity of CICSHL-SVM is $O\big(\big((\lvert\mathcal{M}\rvert+1)^2+n+m\big)\cdot s\big)$. We also provide the analysis of computational complexity of CICSHL-SVM in Table 1.

Because the formulation of CSHL-SVM is very similar to the one of ν-SVC, the convergence of CICSHL-SVM can be proved similarly to the work of [33]. We do not provide the convergence of CSHL-SVM in this paper. If anyone is interested in the convergence of CICSHL-SVM, we recommend the reader to refer to the reference [33].

4. Experimental setup

In this section, we present the experimental setup in terms of the following three aspects: (1) the design of experiments, (2) the details about the implementation of the experiments, and (3) the datasets used in the experiments.

4.1. Design of experiments

As previously mentioned, CICSHL-SVM is more efficient than the batch algorithm of CSHL-SVM (BCSHL-SVM) and the single incremental learning of CSHL-SVM (SICSHL-SVM). In the experiments, we will first investigate the effectiveness of CICSHL-SVM and then compare the running times of CICSHL-SVM with those of SICSHL-SVM and BCSHL-SVM. In addition, we compare CSHL-SVM with standard SVM, BM-SVM and 2C-SVM to confirm that CSHL-SVM has better generalization accuracy for cost-sensitive learning.

To show the effectiveness of CICSHL-SVM, we mainly investigate the average number of iterations and the size of margin vectors as analyzed in Section 3.4. The number of iterations can empirically show the finite convergence of CICSHL-SVM. The size of the margin vectors will help us analyze the running time of CICSHL-SVM under different kernels. Specifically, if the size of the chunk set $\mathcal{A}$ is one, then CICSHL-SVM degenerates into SICSHL-SVM. Thus, we also investigate the average number of iterations of SICSHL-SVM and the size of the margin vectors during the running of SICSHL-SVM over 10 trials.

To verify that CICSHL-SVM is more efficient than SICSHL-SVM and BCSHL-SVM, we investigate the average running times of the three algorithms by gradually increasing the size of the training set over 10 trials on benchmark datasets. In addition, we compare the average running times of CICSHL-SVM and GEM with a warm start. Note that SICSHL-SVM is a special case of CICSHL-SVM with a chunk set size of one.

To confirm that CSHL-SVM has better generalization accuracy for cost-sensitive learning, we use total cost[2] and area under the curve (AUC)[3] [34,35] as two performance evaluation metrics. We compare the total costs and AUCs of CSHL-SVM, standard SVM, BM-SVM, 2C-SVM and GEM on the test sets based on the parameters selected by a 10-fold cross-validation over 10 trials.

4.2. Implementation

We implement our proposed CICSHL-SVM in MATLAB.[4] As mentioned above, SICSHL-SVM is a special case of CICSHL-SVM with a chunk set size of one. Thus, we implement SICSHL-SVM based on CICSHL-SVM. We use the quadprog function from MATLAB and the SeDuMi solver[5] to implement BCSHL-SVM and GEM, respectively. All the experiments were performed on the MATLAB 2015a platform, running on Mac OS X 10.10.2 with a 1.4 GHz Intel Core i5 CPU and 8 GB of memory.

In the experiments for investigating the effectiveness of CICSHL-SVM and comparing the running times of CICSHL-SVM, SICSHL-SVM and BCSHL-SVM, we use three commonly used kernel functions, i.e., a linear kernel $K(x_1,x_2)=x_1\cdot x_2$, a polynomial kernel $K(x_1,x_2)=(2x_1\cdot x_2+1)^d$ with $d=2$, and a Gaussian kernel $K(x_1,x_2)=\exp(-\|x_1-x_2\|^2/2\sigma^2)$ with $\sigma=1.414$. For convenience, we fix the parameters as $C=1$, $C_+=1$ and $C_-=4$. For the on-line scenario, we randomly select 50 new samples to add to the training set. Thus, SICSHL-SVM needs to run 50 times to incorporate the 50 new samples. However, BCSHL-SVM needs to re-train the model on the expanded training set from scratch.

[2] Given a testing set $\mathcal{T}=\{(\bar{x}_1,\bar{y}_1),\ldots,(\bar{x}_l,\bar{y}_l)\}$, the total cost on the testing set is defined as $\mathcal{E}=\frac{1}{l}\sum_{i=1}^{l}C(\hat{y}_i,\bar{y}_i)$, where $\hat{y}_i$ is the predicted label for the sample $(\bar{x}_i,\bar{y}_i)$ and $C(-,+)$ and $C(+,-)$ are the misclassification costs of false negatives and false positives, respectively.

[3] AUC represents an ROC (receiver operating characteristic) curve as a single scalar value by estimating the area under the curve, varying between 0 and 1. The AUC is an important evaluation metric for cost-sensitive learning because it compares the classifiers' performance across the entire range of error costs.

[4] Our MATLAB code for CICSHL-SVM is available at https://sites.google.com/site/jsgubin/our-software-codes.

[5] The MATLAB toolbox SeDuMi site is http://sedumi.ie.lehigh.edu.

Table 2
Benchmark datasets used in the experiments, where YWDT and DWDT are the abbreviations of Yahoo! Web Directory Topics and DMOZ Web Directory Topics, respectively.

Dataset	Samples	Features	Ratio
abalone19	4174	8	130
covtype	581,012	54	1
ijcnn1	49,990	22	9.3
page-blocks0	5472	10	8.8
segment0	2308	19	6
shuttle-c0-vs-c4	1829	9	13.9
svmguide3	1243	22	3.2
winequality-red-4	1599	11	29
yeast4	1484	8	28
YWDT	1106	10,629	3.4
DWDT	1329	10,629	3.9

In the experiments for confirming that CSHL-SVM has better generalization accuracy, we use a grid search to select the kernel parameter and the regularization parameters for standard SVM, BM-SVM, 2*C*-SVM and CSHL-SVM. The Gaussian kernel $K(x_1, x_2) = \exp(-\|x_1 - x_2\|^2/2\sigma^2)$ is used with $\sigma \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$. The specific settings for the regularization parameters in standard SVM, BM-SVM, 2*C*-SVM and CSHL-SVM are as follows.

1. The C, C_-, and C_+ in standard SVM, BM-SVM or 2*C*-SVM are linearly selected from the region $\{(\log_2 C, \log_2 C_-, \log_2 C_+) | -9 \le \log_2 C \le 10, -9 \le \log_2 C_- \le 10, -9 \le \log_2 C_+ \le 10, \}$.
2. The C, C_- and C_+ in CSHL-SVM are linearly selected from the region $\{(\log_2 C, \log_2 C_-, \log_2 C_+) | -9 \le \log_2 C \le 10, 0 \le \log_2 C_- \le 10, 0 \le \log_2 C_+ \le 10, C_+ \ge 2C_- - 1\}$.

4.3. Datasets

Table 2 summarizes the eleven benchmark datasets used in our experiments, where the dataset covtype is from the UCI machine learning repository [36], the ijcnn1 and svmguide3 datasets are from the LIBSVM site[6] [37], the abalone19, page-blocks0, segment0, winequality-red-4 and yeast4 datasets are from the KEEL dataset repository[7] [38], and the Yahoo! Web Directory Topics (YWDT) and DMOZ Web Directory Topics (DWDT) are from http://www.mldata.org/repository/data/. Note that the YWDT and DWDT datasets are originally the ordinal regression datasets [39]. In our experiments, we treat the first class as the negative class, and the other classes as the positive class. The column "Ratio" in Table 2 presents the ratio between the size of the positive samples to the size of the negative samples in a dataset. The ratios for the abalone19, shuttle-c0-vs-c4, winequality-red-4 and yeast4 datasets are greater than 10. These datasets are imbalanced datasets. Cost-sensitive learning is also considered to be a good solution for imbalanced learning, as mentioned in Section 1.

In the experiments for confirming that CSHL-SVM has better generalization accuracy, we randomly partition each dataset into 65% training and 35% test sets. The training set is used to determine the optimal parameters with a 10-fold cross-validation procedure. In addition, we map the values of each feature into the range [0,1] to normalize all nine benchmark datasets.

[6] The URL for the datasets from the LIBSVM site is http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

[7] The URL for the datasets from the KEEL site is http://sci2s.ugr.es/keel/imbalanced.php.

5. Experimental results and discussion

In this section, we present the experimental results and discussion, first for verifying the effectiveness of CICSHL-SVM, then for verifying the efficiency of CICSHL-SVM, and finally for confirming the effectiveness of CSHL-SVM.

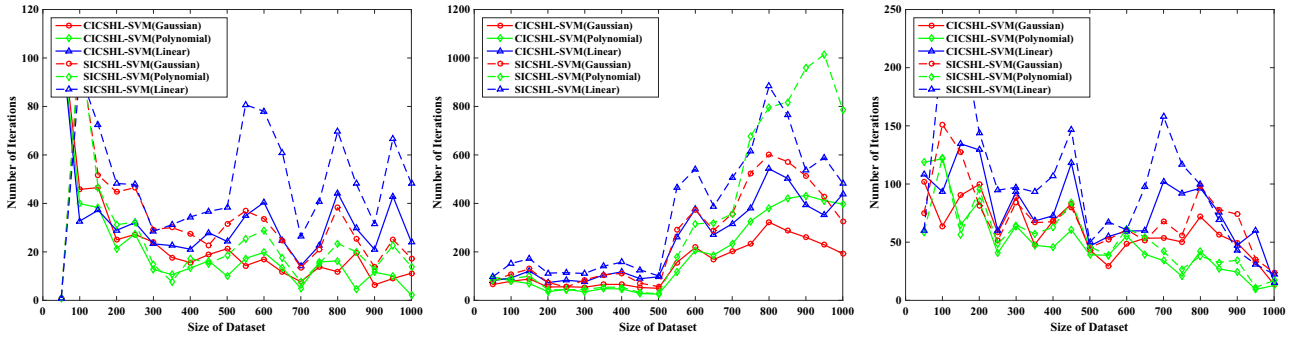
5.1. The effectiveness of CICSHL-SVM

As analyzed in Section 3.4, the computational complexity of CICSHL-SVM is $O\left(\left((|\mathcal{M}|+1)^2 + n + m\right) \cdot s\right)$. Thus, we primarily investigate the main factors in the computational complexity (i.e., the average number of iterations, denoted by s, and the size of the margin vectors, denoted by $|\mathcal{M}|$). Fig. 1 presents the average numbers of iterations of CICSHL-SVM and SICSHL-SVM on the abalone19, covtype, ijcnn1, page-blocks0, segment0, shuttle-c0-vs-c4, svmguide3, winequality-red-4 and yeast4 datasets with different kernels. As shown in Fig. 1, both CICSHL-SVM and SICSHL-SVM with the linear, polynomial and Gaussian kernels converge to the optimal solution with a high convergence speed. This result also demonstrates the efficiency of CICSHL-SVM, which will be discussed in Section 5.2. More importantly, the number of iterations does not evidently increase with the size of the training set. In contrast, the number of iterations for both CICSHL-SVM and SICSHL-SVM decreases with the size of the training set on the ijcnn1, shuttle-c0-vs-c4 and yeast4 datasets. This result is because the samples in a chunk set $\mathcal{A}$ have a higher probability of being correctly classified before the adjustments of CICSHL-SVM and SICSHL-SVM with the increment of the size of the training set. In addition, as shown in Fig. 1, the number of iterations of SICSHL-SVM is greater than that of CICSHL-SVM on almost all the benchmark datasets. This phenomenon will help us to explain why CICSHL-SVM is more efficient than SICSHL-SVM, as discussed in Section 5.2.

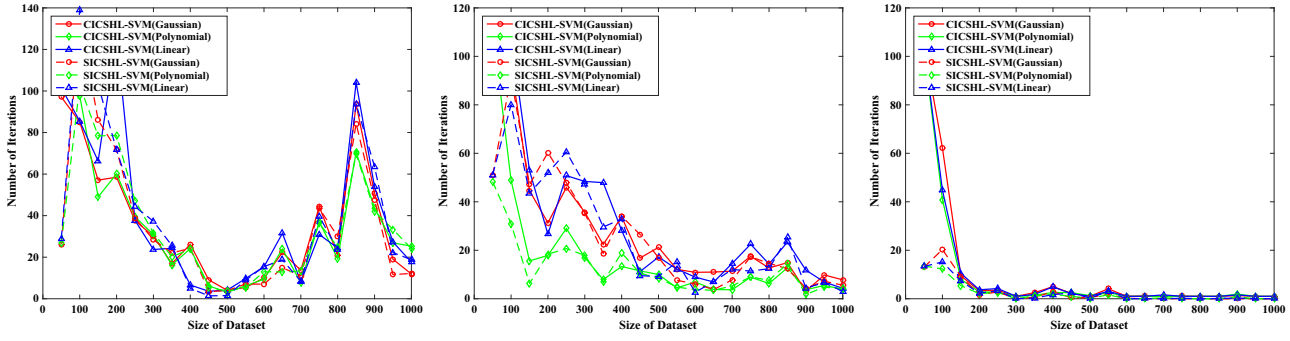
Fig. 2 presents the average numbers of margin vectors in $\mathcal{M}$ for CICSHL-SVM and SICSHL-SVM with different kernels on the abalone19, covtype, ijcnn1, page-blocks0, segment0, shuttle-c0-vs-c4, svmguide3, winequality-red-4 and yeast4 datasets. As shown in Fig. 2, the number of margin vectors of CICSHL-SVM and SICSHL-SVM with the linear, polynomial and Gaussian kernels increases approximately linearly with the size of the training set, except on the shuttle-c0-vs-c4 dataset. For the shuttle-c0-vs-c4 dataset, it is not surprising that the number of margin vectors becomes relatively constant because the iterations of CICSHL-SVM and SICSHL-SVM are near zero, as shown in Fig. 1 f. In addition, as shown in Fig. 1, SICSHL-SVM has more margin vectors than does CICSHL-SVM on almost all the benchmark datasets. As mentioned in Section 3, the computational complexity of Algorithm 1 primarily depends on solving the linear system Eq. (17), which has a size of $|\mathcal{M}| + 1$. This means that the computational complexity of each iteration of SICSHL-SVM is higher than that of CICSHL-SVM.

5.2. The efficiency of CICSHL-SVM

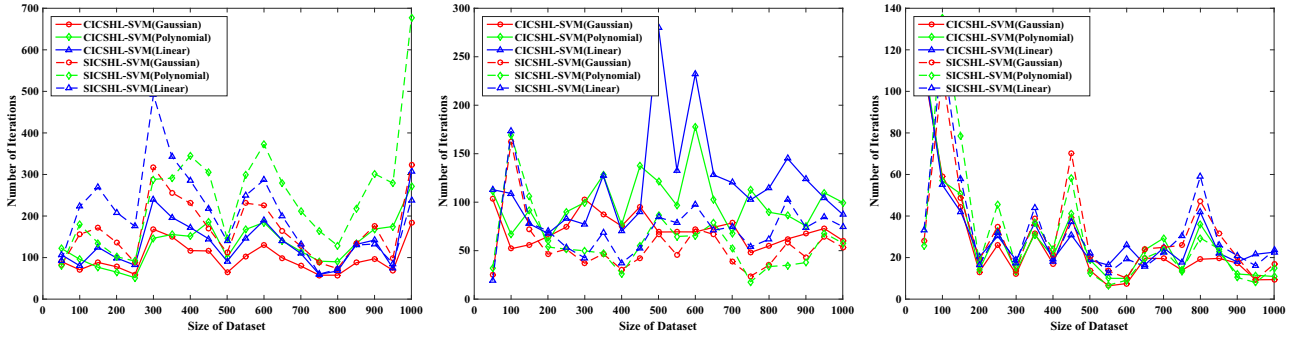
Fig. 3 presents the average running times (in seconds) of CICSHL-SVM, SICSHL-SVM, BCSHL-SVM and GEM on the abalone19, covtype, ijcnn1, page-blocks0, segment0, shuttle-c0-vs-c4, svmguide3, winequality-red-4 and yeast4 datasets with different kernels. As shown in Fig. 3, both CICSHL-SVM and SICSHL-SVM are significantly faster than BCSHL-SVM on all nine benchmark datasets when the linear, polynomial and Gaussian kernels are used. This result is because BCSSVM would re-train the model from scratch irrespective of how many new samples are added. However, CICSHL-SVM and SICSHL-SVM can update the model for a chunk of new samples without re-training from scratch. Specifically, as shown in Fig. 3, although CICSHL-SVM, SICSHL-SVM and



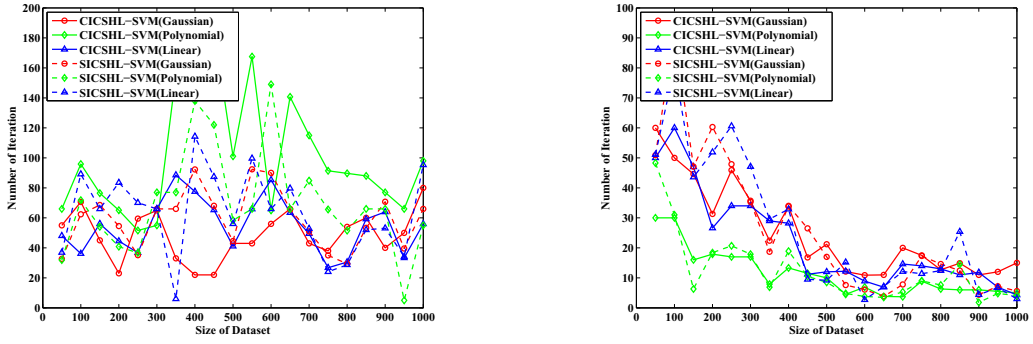
(a) abalone19 (b) covtype (c) ijcnn1



(d) page-blocks0 (e) segment0 (f) shuttle-c0-vs-c4



(g) svmguide3 (h) winequality-red-4 (i) yeast4



(j) YWDT (k) DWDT

Fig. 1. Average numbers of iterations of CICSHL-SVM and SICSHL-SVM on different benchmark datasets.

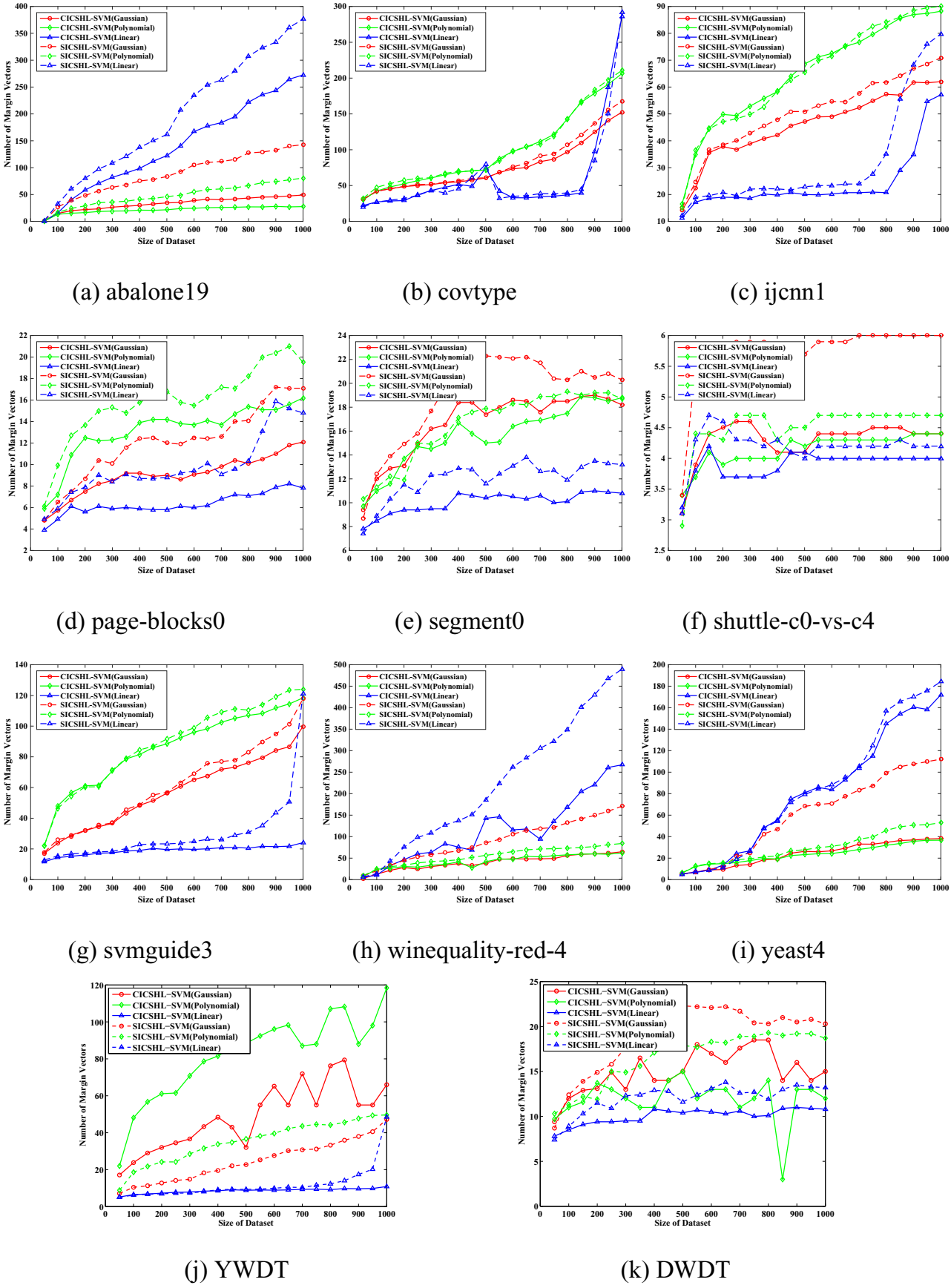


Fig. 2. Average numbers of margin vectors of CICSHL-SVM on different benchmark datasets.

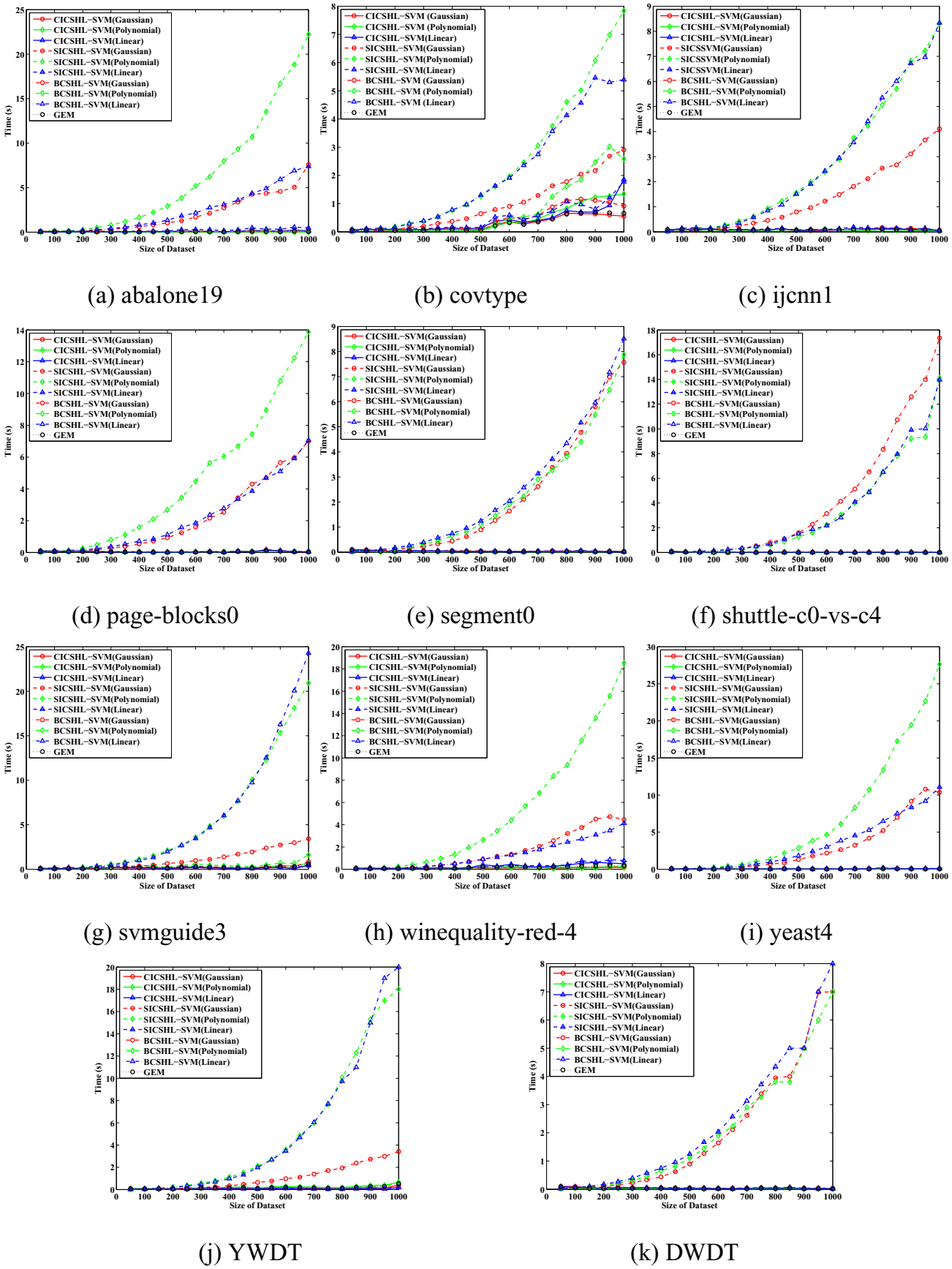


Fig. 3. Average running times (in seconds) of CICSHL-SVM, SICSHL-SVM, BCSHL-SVM and GEM on different benchmark datasets.

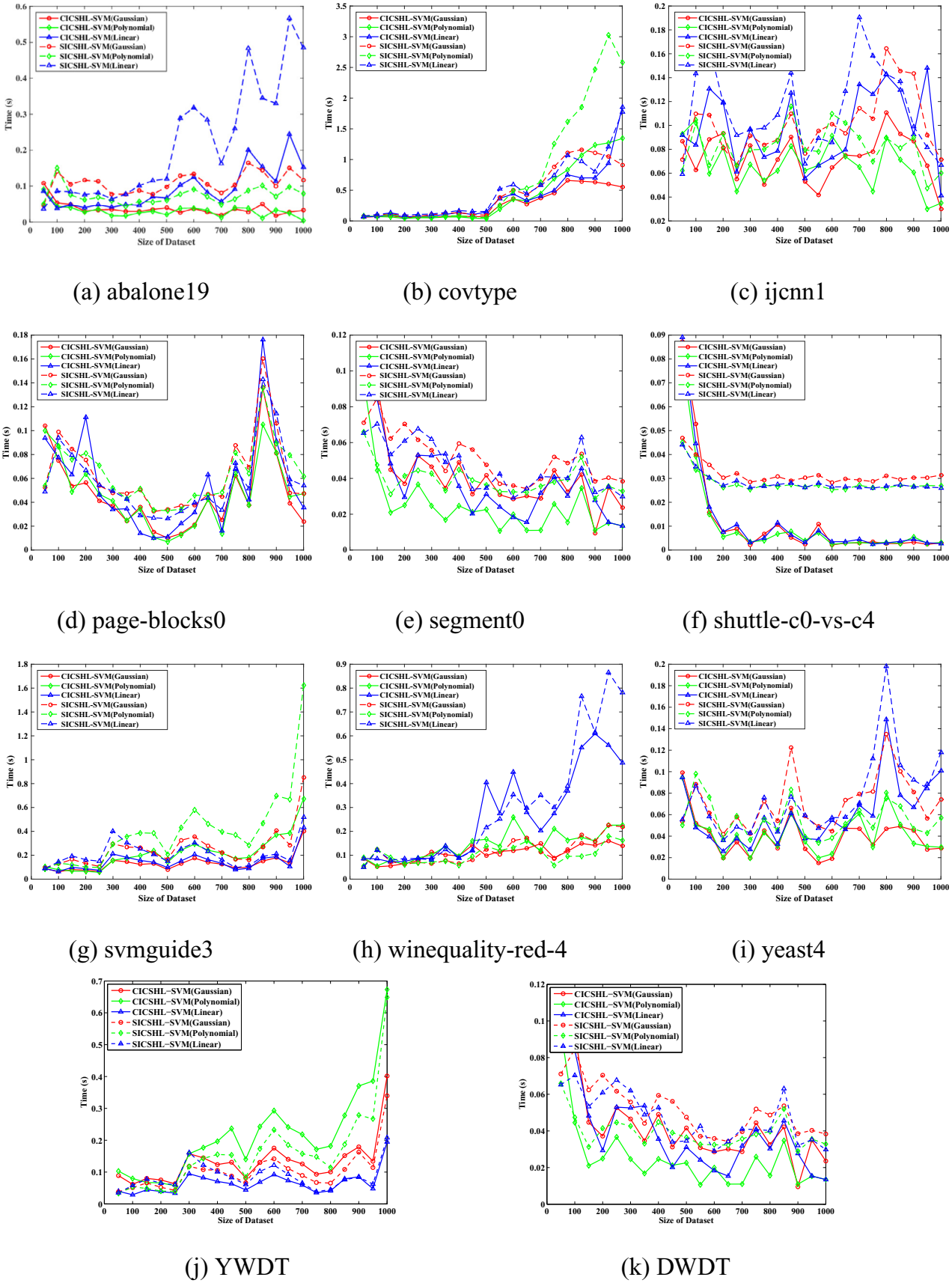
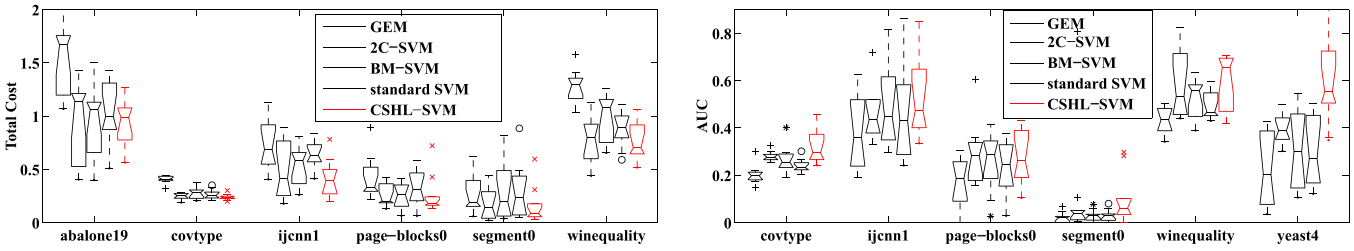


Fig. 4. Average running times (in seconds) of CICSHL-SVM and SICSHL-SVM on different benchmark datasets.



(a) MAEs on Bank dataset. (b) MAEs on Bank dataset.

Fig. 5. Costs and AUCs (over 10 trials) of GEM, 2C-SVM, BM-SVM, standard SVM, and CSHL-SVM. The results clearly show that CSHL-SVM has better generalization accuracy. The grouped boxes represent the results of GEM, 2C-SVM, BM-SVM, standard SVM, and CSHL-SVM from left to right on different datasets. The notched boxes have lines at the lower, median, and upper quartile values. The whiskers are lines extended from each end of the box to the most extreme data value within a $1.5 \times$ IQR (interquartile range) of the box. Outliers are data with values beyond the ends of the whiskers, which are displayed using plus signs. (a) Total cost. (b) AUC.

BCSHL-SVM scale approximately linearly with the size of the training set, the linear scale factor of BCSHL-SVM is considerably higher than those of CICSHL-SVM and SICSHL-SVM. Thus, the difference in running times between CICSHL-SVM (SICSHL-SVM) and BCSHL-SVM becomes increasingly larger as the training set size increases. In addition, although GEM with a warm start is as efficient as CICSHL-SVM and SICSHL-SVM, it has poor generalization accuracy, as shown in Section 5.3.

Again, we present the average running times (in seconds) of CICSHL-SVM and SICSHL-SVM in Fig. 4 to clearly show the difference in the running times between CICSHL-SVM and SICSHL-SVM. As shown, CICSHL-SVM is faster than SICSHL-SVM with the linear, polynomial and Gaussian kernels on all the benchmark datasets. This result is because CICSHL-SVM requires fewer iterations and fewer margin vectors than does SICSHL-SVM, as discussed in Section 5.1. In addition, we find that different kernels lead to different efficiencies for CICSHL-SVM and SICSHL-SVM on different datasets. For example, CICSHL-SVM (SICSHL-SVM) with the linear kernel costs more time than that with the Gaussian kernel on the abalone19, ijcnn1 and winequality-red-4 datasets. However, CICSHL-SVM (SICSHL-SVM) with the linear kernel costs less time than that with the Gaussian kernel on the shuttle-c0-vs-c4 dataset. This result is because CICSHL-SVM (SICSHL-SVM) with the Gaussian kernel requires fewer iterations but more margin vectors than does that with the linear kernel, as discussed in Section 5.1. Thus, CICSHL-SVM (SICSHL-SVM) with the Gaussian kernel is occasionally more efficient than that with the linear kernel; however, they are sometimes completely opposite.

5.3. The effectiveness of CSHL-SVM

Fig. 5 presents the total costs and AUCs of CSHL-SVM, standard SVM, BM-SVM, 2C-SVM and GEM on the abalone19, covtype, ijcnn1, page-blocks0, segment0, winequality and yeast4 datasets with the Gaussian kernel. For the total cost, $C(+,-)$ is set to 1, and $C(-,+)$ is set to 5. As indicated by the results, CSHL-SVM can achieve the smallest total cost and the highest AUC on these benchmark datasets. Although 2C-SVM is better than standard SVM, it can be observed that there are obvious gaps compared with CSHL-SVM. The results confirm that CSHL-SVM has better generalization accuracy than standard SVM, BM-SVM, 2C-SVM and GEM; therefore, CSHL-SVM is more suitable for cost-sensitive learning.

6. Conclusion

In this paper, we proposed a chunk incremental learning algorithm for CSHL-SVM (i.e., CICSHL-SVM) that can update a trained model without re-training from scratch when incorporating a chunk of new samples. Specifically, if the size of the chunk set is one, then our CICSHL-SVM degenerates into a single incremental algorithm. Our method is efficient because it can update the trained model not only for one sample at a time but also for multiple samples at a time. The experimental results on a variety of datasets not only confirm the effectiveness of CSHL-SVM but also show that our method is more efficient than the batch algorithm of CSHL-SVM and the single incremental algorithm.

This paper only focuses on binary classification problems. In the future, we plan to extend the chunk incremental algorithm and the cost-sensitive strategy to ordinal regression [40], multi-class classification [41] and semi-supervised learning [42] problems. In addition, we hope to design a general chunk incremental learning algorithm, which would benefit the machine learning community in unifying chunk incremental implementations with different learning formulations. In the future, we plan to apply the method to image classification [43–45,48] and computer-aided diagnosis [46,47] in on-line scenarios.

Theoretically, the chunk decremental learning for CSHL-SVM can also be designed in a similar manner. Moreover, we will provide feasibility analysis for CICSHL-SVM, similar to the work of [33].

Acknowledgments

This work was supported by the Project Funded by the Priority Academic Program Development (PAPD) of Jiangsu Higher Education Institutions, the Natural Science Foundation (No. BK20161534), Six talent peaks project (No. XYDXX-042) and the 333 Project (No. BRA2017455) in Jiangsu Province, the U.S. National Science Foundation (IIS-1115417), and the National Natural Science Foundation of China (No. 61573191).

References

[1] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
[2] M.C. Campi, Classification with guaranteed probability of error, Mach. Learn. 80 (1) (2010) 63–84.
[3] Y.-J. Park, S.-H. Chun, B.-C. Kim, Cost-sensitive case-based reasoning using a genetic algorithm: application to medical diagnosis, Artif. Intell. Med. 51 (2) (2011) 133–145.
[4] N. Mahmoudi, E. Duman, Detecting credit card fraud by modified fisher discriminant analysis, Expert Syst. Appl. 42 (5) (2015) 2510–2516.
[5] P.J. Koyande, D.H. Deshmukh, S.D. Naravadkar, T-Alert: terrorist alert system using data mining techniques, Int. J. Eng. Sci. 5886 (2016).
[6] C. Elkan, The foundations of cost-sensitive learning, in: International Joint Conference on Artificial Intelligence, vol. 17, Lawrence Erlbaum Associates Ltd, 2001, pp. 973–978.
[7] J. Wang, P. Zhao, S.C. Hoi, Cost-sensitive online classification, IEEE Trans. Knowl. Data Eng. 26 (10) (2014) 2425–2438.
[8] V. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.

[9] G.I. Karakoulas, J. Shawe-Taylor, Optimizing classifers for imbalanced training sets, in: Advances in Neural Information Processing Systems, 1999, pp. 253–259.
[10] E.E. Osuna, Support vector machines: Training and applications, Massachusetts Institute of Technology, 1998 Ph.D. thesis.
[11] M.A. Davenport, R.G. Baraniuk, C.D. Scott, Tuning support vector machines for minimax and Neyman–Pearson classification, IEEE Trans. Pattern Anal. Mach. Intell. 32 (10) (2010) 1888–1898.
[12] H. Masnadi-Shirazi, N. Vasconcelos, Risk minimization, probability elicitation, and cost-sensitive svms., in: Proceedings of the ICML, Citeseer, 2010, pp. 759–766.
[13] H. Masnadi-Shirazi, N. Vasconcelos, A. Iranmehr, Cost-sensitive support vector machines, (2012). arXiv:1212.0975
[14] P. Laskov, C. Gehl, S. Krüger, K.-R. Müller, Incremental support vector learning: analysis, implementation and applications, J. Mach. Learn. Res. 7 (Sep) (2006) 1909–1936.
[15] A. Bordes, S. Ertekin, J. Weston, L. Bottou, Fast kernel classifiers with online and active learning, J. Mach. Learn. Res. 6 (Sep) (2005) 1579–1619.
[16] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, Pegasos: primal estimated sub-gradient solver for SVM, Math. Program. 127 (1) (2011) 3–30.
[17] N. Karampatziakis, J. Langford, Online importance weight aware updates, in: Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2011, pp. 392–399.
[18] S. Lacoste-Julien, M. Jaggi, M. Schmidt, P. Pletscher, Block-coordinate Frank–Wolfe optimization for structural SVMs, in: Proceedings of the ICML International Conference on Machine Learning, 2013, pp. 53–61.
[19] S. Shalev-Shwartz, T. Zhang, Stochastic dual coordinate ascent methods for regularized loss minimization, J. Mach. Learn. Res. 14 (Feb) (2013) 567–599.
[20] G. Cauwenberghs, T. Poggio, Incremental and decremental support vector machine learning, in: Advances in Neural information Processing Systems, 2001, pp. 409–415.
[21] M. Martin, On-line support vector machine regression, in: European Conference on Machine Learning, Springer, 2002, pp. 282–294.
[22] B. Gu, J.-D. Wang, Y.-C. Yu, G.-S. Zheng, Y.-F. Huang, T. Xu, Accurate on-line ν-support vector learning, Neural Netw. 27 (2012) 51–59.
[23] B. Gu, V.S. Sheng, Z. Wang, D. Ho, S. Osman, S. Li, Incremental learning for ν-support vector regression, Neural Netw. 67 (2015) 140–150.
[24] B. Gu, V.S. Sheng, K.Y. Tay, W. Romano, S. Li, Incremental support vector learning for ordinal regression, IEEE Trans. Neural Netw. Learn. Syst. 26 (7) (2015) 1403–1416.
[25] B. Gu, J. Wang, H. Chen, On-line off-line ranking support vector machine and analysis, in: Proceedings of the IEEE International Joint Conference on Neural Networks, IJCNN 2008. (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 1364–1369.
[26] M. Karasuyama, I. Takeuchi, Multiple incremental decremental learning of support vector machines, IEEE Trans. Neural Netw. 21 (7) (2010) 1048–1059.
[27] B. Schölkopf, A.J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, 2002.
[28] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.
[29] C.-J. Ong, S. Shao, J. Yang, An improved algorithm for the solution of the regularization path of support vector machine, IEEE Trans. Neural Netw. 21 (3) (2010) 451–462.
[30] J. Dai, C. Chang, F. Mai, D. Zhao, W. Xu, On the SVMpath singularity, IEEE Trans. Neural Netw. Learn. Syst. 24 (11) (2013) 1736–1748.
[31] D. Poole, Linear Algebra: A Modern Introduction, Cengage Learning, 2014.
[32] G.H. Golub, C.F. Van Loan, Matrix Computations, vol. 3, JHU Press, 2012.
[33] B. Gu, V.S. Sheng, Feasibility and finite convergence analysis for accurate on–line ν-support vector machine, IEEE Trans. Neural Netw. Learn. Syst. 24 (8) (2013) 1304–1315.
[34] C.X. Ling, J. Huang, H. Zhang, et al., Auc: a statistically consistent and more discriminating measure than accuracy, in: Proceedings of the IJCAI, 3, 2003, pp. 519–524.
[35] W. Gao, R. Jin, S. Zhu, Z.-H. Zhou, One-pass AUC optimization, in: Proceedings of the International Conference on Machine Learning, 2013, pp. 906–914.
[36] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. http://archive.ics.uci.edu/ml.
[37] C.-C. Chang, C.-J. Lin, Libsvm: a library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (3) (2011) 27.
[38] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Mult. Valued Log. Soft Comput. 17 (2011) 255–287.
[39] B. Gu, A regularization path algorithm for support vector ordinal regression, Neural Netw. 98 (2018) 114–121.
[40] A. Riccardi, F. Fernández-Navarro, S. Carloni, Cost-sensitive adaboost algorithm for ordinal regression based on extreme learning machine, IEEE Trans. Cybern. 44 (10) (2014) 1898–1909.
[41] S.P. Parambath, N. Usunier, Y. Grandvalet, Optimizing f-measures by cost-sensitive classification, in: Advances in Neural Information Processing Systems, 2014, pp. 2123–2131.
[42] Y.-H. Zhou, Z.-H. Zhou, Large margin distribution learning with cost interval and unlabeled data, IEEE Trans. Knowl. Data Eng. 28 (7) (2016) 1749–1763.
[43] C. Deng, X. Liu, C. Li, D. Tao, Active multi-kernel domain adaptation for hyperspectral image classification, Pattern Recognit. 77 (2018) 306–315, doi:10.1016/j.patcog.2017.10.007.
[44] J. Hu, Y.-P. Tan, Nonlinear dictionary learning with application to image classification, Pattern Recognit. 75 (2018) 282–291, doi:10.1016/j.patcog.2017.02.009.
[45] X. Zhou, X. Gao, J. Wang, H. Yu, Z. Wang, Z. Chi, Eye tracking data guided feature selection for image classification, Pattern Recognit. 63 (2017) 56–70, doi:10.1016/j.patcog.2016.09.007.
[46] L. Tsochatzidis, K. Zagoris, N. Arikidis, A. Karahaliou, L. Costaridou, I. Pratikakis, Computer-aided diagnosis of mammographic masses based on a supervised content-based image retrieval approach, Pattern Recognit. 71 (2017) 106–117, doi:10.1016/j.patcog.2017.05.023.
[47] P. Cao, X. Liu, J. Yang, D. Zhao, W. Li, M. Huang, O. Zaiane, A multi-kernel based framework for heterogeneous feature selection and over-sampling for computer-aided detection of pulmonary nodules, Pattern Recognit. 64 (2017) 327–346, doi:10.1016/j.patcog.2016.11.007.
[48] C. Deng, Z. Chen, X. Liu, et al., Triplet-Based Deep Hashing Network for Cross–Modal Retrieval, IEEE Trans. Image Process. 27 (8) (2018) 3893–3903.

Bin Gu (M08) received the B.S. and Ph.D. degrees in computer science from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2005 and 2011, respectively. He joined the School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, in 2010, as a Lecturer, where he is currently an Associate Professor. He is also currently a Post-Doctoral Fellow with the University of Western Ontario, London, ON, Canada. His current research interests include machine learning, data mining, and medical image analysis.

Xin Quan received the B.S. and M.S. degrees in computer science and technology from the Nanjing University of Information Science and Technology, Nanjing, China, in 2013 and 2016, respectively.

Yunhua Gu received the B.S. degrees in computer science and technology from the Shanghai Jiao Tong University, Shanghai, China, in 1987 and received the M.S. degrees in computer science and technology from the Southeast University, Nanjing, China, in 2000, respectively. She joined the School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, in 1987, as a Lecturer, where he is currently a Professor.

Victor S. Sheng (M11) received the Ph.D. degree in computer science from the University of Western Ontario, London, ON, Canada, in 2007.He was an Associate Research Scientist and NSERC Post-Doctoral Follow in Information Systems with Stern Business School, New York University, New York, NY, USA. He is an Assistant Professor with the Department of Computer Science, University of Central Arkansas, Conway, AR, USA, and the founding Director of the Data Analytics Laboratory. His current research interests include data mining, machine learning, and related applications.Prof. Sheng is a member of the IEEE Computer Society. He is a PC Member for a number of international conferences and a reviewer for several international journals. He was a recipient of the Best Paper Runner-Up Award from the 2008 International Conference on Knowledge Discovery and Data Mining, and the Best Paper Award from the 2011 Industrial Conference on Data Mining.