International Journal of Machine Learning and Cybernetics (2019) 10:2389–2400
https://doi.org/10.1007/s13042-018-0877-7

ORIGINAL ARTICLE

An accelerator for support vector machines based on the local geometrical information and data partition

Yunsheng Song[1,2] · Jiye Liang[3] · Feng Wang[3]

Received: 8 October 2017 / Accepted: 18 September 2018 / Published online: 9 October 2018

Abstract
The support vector machines (SVM) is difficult to deal with large datasets for its low training efficiency. One of the important solutions has been developed by dividing a whole dataset into smaller subsets with data partition and combining the results of the classifiers over the divided subsets. However, traditional data partition approaches are difficult to preserve the class boundary of the dataset or control the size of divided subsets, so that their performance will be greatly influenced. To overcome this difficulty, we propose an accelerator for SVM algorithm based on the local geometrical information. In this algorithm, the feature space is divided into several regions with the approximately equal number of training instances by linear projection, and then each SVM classifier trained over the extended region only predicts the unlabeled instances within that original region. The proposed algorithm can not only hold the decision boundary of the raw data, but also saves a lot of execution time for implementing it in a parallel environment. Furthermore, the number of instances within each divided regions can be effectively controlled; it is conducive to choose the complexity of the execution in each of the processors. Experiments show that the classification performance of the proposed algorithm compares favorably with four state-of-the-art algorithms with the least training time.

Keywords Data partition · Feature space · Classification boundary · Training time · Linear projection

1 Introduction

The objective of classification task is to assign a label to an unlabeled instance. A well-known supervised classification technique is SVM. SVM uses a nonlinear mapping to transform the original training instances into a higher dimension, and then searches for the linear optimal separating hyperplane, which separates the instances from two classes within this new dimension [1, 2]. Owning to its theoretical soundness and practical performance, SVM has been highly successful in practical applications, such as detection and recognition, handwritten character and digit recognition, text detection and categorization [3–9]. Moreover, SVM with Gaussian kernel has been demonstrated to be the best classifier expect random forest by evaluating 179 classifiers with 121 UCI datasets [10].

Training an SVM classifier is equivalent to solving a convex quadratic programming (QP) problem, and it is characterized by a kernel matrix with a number of rows equal to the number of training instances. The traditional QP solver for SVM has a time complexity of $O(N^3)$, where N is the number of training instances. Thus, it becomes computationally intractable as the size of the training dataset is large. Moreover, its prediction time is proportional to the number of support vectors, it is also time-consuming significantly with an increase in training scale. Therefore, an increasing number of algorithms have been developed to speed up it. These algorithms can be categorized using a range of different taxonomies. Among them, we mention

✉ Jiye Liang
ljy@sxu.edu.cn

Yunsheng Song
sys_sd@126.com

Feng Wang
sxuwangfeng@126.com

[1] College of Information Science and Engineering, Shandong Agricultural University, Taian 271018, Shandong, China

[2] School of Computer and Information Technology, Shanxi University, Taiyuan 030006, Shanxi, China

[3] Key Laboratory of Computational Intelligence and Chinese Information Processing of Ministry of Education, School of Computer and Information Technology, Shanxi University, Taiyuan 030006, Shanxi, China

training instances reduction algorithms and decomposition algorithms.

There are a lot of reduction algorithms. The principal idea of instance reduction is that the instances with different locations in the feature space have different contributions to the training of SVM classifier [11, 12]. The training instances can be differentiated using two terms: overlapping and non-overlapping instances. Overlapping instances are those ones that can be sorted out by one class or another class at the same time. These instances are usually closed to the decision boundary. Furthermore, though the SVM classifier is trained using all the training instances, the decision plane is generated only using some critical instances which are called support vectors (SVS). Strictly speaking, SVS cannot be got before the training process of SVM, then most of the instance selection algorithms seek overlapping instances to replace SVS. However, due to the quadratic time complexity of this kind of algorithms, they still cost a lot of time for large-scale datasets.

On the other hand, there are a lot of decomposition algorithms. These algorithms mainly divide the original training dataset into some subsets, and then train an SVM classifier on each subset [13, 14]. Among these kinds of algorithms, random partition and clustering are most common ways to data partition. Random partition means that all instances are separated into the same sized subsets randomly, but it does not consider the relationship among them. Conversely, clustering usually divides all training instances into a family of subsets with different sizes according to the similarity relationship among them.

This paper has proposed a novel SVM algorithm based on the local geometrical information and data partition (L-SVM). In this algorithm, the kernel linear projection algorithm is introduced to detect the class boundary firstly. Then the feature space is divided into some regions, and the SVM classifiers are trained over the extend regions. Finally, the SVM classifier offers the predicted labels for the unlabeled instances in the same region. Analysis and experiments show that the proposed algorithm can effectively reduce the training time, but it could match with the classification performance of the original SVM. Furthermore, it has the additional advantage that we can adapt the size of the subproblems to the available resources.

The rest of this work is organized as follows. Section 2 briefly reviews previous approaches for accelerating SVM training process, and their strengths and weaknesses are discussed. Section 3 proposes the L-SVM algorithm with some main properties. Section 4 reports experimental results on benchmark classification problems compared with four existing algorithms. Finally, Section 5 concludes the classification performance, training time and potential applications of the proposed algorithm.

2 Related work

The training process of the traditional SVM is always computationally intractable when dealing with large datasets, so more and more algorithms are developed to solve it. These algorithms are mainly categorized into two types according to the different strategies: filter and decomposition.

The filter algorithms choose a smaller instance subset S by using various kinds of instance selection algorithms, where the subset S includes critical instances of the training set T . Finally, train the SVM algorithm over the subset S. Specifically, these algorithms mainly try to seek the overlapping instances in the training set T. This kind of instances could fully and succinctly constitute the SVM classifier [15, 16]. As there are fewer overlapping instances than non-overlapping instances in most of the datasets, the obtained subset size could be normally small. So these algorithms could largely reduce the training time. Well-known filter algorithms include the algorithms based on nearest neighbor and clustering [17, 18], training-set-consistent algorithm [19], the algorithm based on local geometrical and statistical information [20], and so on. A potential disadvantage of these algorithms is that they may need a long time for huge problems. Because these algorithms usually require to compute the distances among the training instances, and execute many passages over the training set to seek the critical instances. Furthermore, one kind of safe samples screening methods have been proposed, and these methods aim to greatly reduce the scale of SVM by identifying and deleting the non-support vectors [21–23]. Although this kind of methods can guarantee to achieve the same solution as solving the original problem, they first need to get an initial optimal solution with the training instances under a small value parameter, and then they compute a sequence of the optimal solutions at subsequent larger values of the parameter. Therefore, these methods cost a lot of time for getting the final optimal solution for using the all the data and continually searching different solutions under different parameter values.

Different from filter algorithms, decomposition algorithms train SVM classifier with all the training instances rather than some of them. There are three key steps in these algorithms. The first one is dividing the original training datasets into some subsets. The second one is training SVM classifier on each divided subset. And the last one is combining the results over all the subsets for predicting. Collobert et al. [24] randomly divided the training dataset into some subsets of approximately equal size and trained SVM classifiers separately, then iteratively refined the data partition to obtain a good approximate estimate of raw SVM classifier. However, the experiments of this algorithm are taken on only two datasets, and its performance still needed to be studied on much more datasets. Later, Graf [25] proposed a multilevel and parallel algorithm (CascadeSVM) to early identify the support vectors. Cascade SVM firstly divided the training data into some subsets. And then SVM algorithm was performed separately over each subset and the corresponding SVS were obtained. Finally, the obtained SVS were combined two-by-two and filtered again until the global optimum was reached. However, it did not take the effect of data partition to the performance of CascadeSVM into account. Do and Poulet [14] applied k-means to partition the training data into k clusters, and then constructed a SVM classifier on each cluster to classify the instances locally in a parallel way. Though this algorithm achieved a good performance on some datasets, it was difficult to theoretically show the error between the solution from the subproblems and the global solution. For this problem, Hsieh et al. [13] proposed and analyzed a novel divide-and-conquer solver for kernel SVM (DC-SVM). In the division step, the training data was partitioned into smaller subproblems by kernel clustering. And they theoretically showed the support vectors identified by the subproblem solution were likely to be support vectors of the entire kernel SVM problem. In the conquer step, the local solutions from the subproblems could converge quickly as suggested by their analysis. It was also theoretically shown that DC-SVM was likely to obtain the SVS as the same as the raw SVM problem. From above we could find that the performance of DC-SVM mainly depended on the results of the clustering. Furthermore, the size of divided subsets may be quite different, so that its training time may be quite different. The final finishing time depends on the time of training the largest subset. Though DC-SVM has achieved a great deal of reduction on training time, it suffers from significant loss of accuracy because smaller partitions do not retain the distribution of the entire dataset. To overcome this difficulty, Singh et al. [26] proposed a distribution preserving kernel support vector machine (DIP-SVM) for big data. DIP-SVM employs K-means clustering on each of the classes into the same number of subsets, then merges the instances selecting from each subset according to uniform distribution into a new subset. The experiment results show DIP-SVM achieves a minimal loss in classification accuracy among other distributed support vector machine techniques on several benchmark datasets.

Besides, there are other algorithms to reduce the test time for dealing with SVM on large problems [27–30].

3 Related concept

Let $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ denote the labeled training set, where x_i is the ith instance described by m attributes, $y_i \in \{-1, 1\}$ is, its expected real-valued output, and N is the number of labeled instances.

Given the training dataset T, SVM transforms all the training instances into a higher dimension space $\mathcal{X}$ by a mapping function $\varphi(\cdot)$, and searches the optimal separating hyperplane $\omega.\varphi(x) + b = 0$ that yields the largest margin $(2/||\omega||^2)$ between classes, where ω, b are the weight vector and bias term, and $||\omega||$ is the norm of ω. The vector ω is obtained by solving the following quadratic programming problem [31].

$$\begin{aligned} &\min\ 1/2||\omega||^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t.\ \ y_i(\omega.\varphi(x_i)) \geq 1 - \xi_i \\ &\qquad \xi_i \geq 0, i = 1, 2, \dots, N \end{aligned} \tag{1}$$

where $\xi = (\xi_1, \xi_2, \dots, \xi_N)$ are nonnegative slack variables which play a role in allowing a certain level of misclassification for a non-separable case, $C > 0$ is an error tolerance parameter, and $\varphi(x_i).\varphi(x_j)$ is the inner product between $\varphi(x_i)$ and $\varphi(x_j)$.

The optimal solution of problem (1) can be transferred into the convex quadratic problem by Lagrange multiplier algorithm [31],

$$\min_{0 \leq \alpha_i \leq C} \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N}\alpha_i + b\sum_{i=1}^{N}\alpha_i y_i \tag{2}$$

where $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_N) \in R^N$ is the nonnegative Lagrange multiplier associated with training instances, $\omega = \sum_{i=1}^{N}\alpha_i y_i \varphi(x_i)$, and $K(x_i, x_j)$ is the function to avoid the costly calculation of inner products $(\varphi(x_i), \varphi(x_j))$. Let $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)$ be the optimal solution of problem (1), we get the SVM classifier $f(x) = sign(\omega^* \cdot \varphi(x) + b^*)$, where $\omega^* = \sum_{i=1}^{N}\alpha_i^* y_i \varphi(x_i)$, $b^* = y_j - \sum_{i=1}^{N}\alpha_i^* y_i K(x_i, x_j)$, and α_j^* is one element of α^* that satisfies $0 < \alpha_j^* \leq C$. It is noted that we do not observe any improvement in test accuracy by including the bias term similar as [27, 32], so we set $b = 0$ in the following content.

4 L-SVM algorithm

Given the training dataset T, L-SVM algorithm divides the feature space $\mathcal{X}$ into n regions $\mathcal{X}_i$ $(\bigcup_{i=1}^{n}\mathcal{X}_i = \mathcal{X})$ by linear projection partition, and extends each region $\mathcal{X}_i$ to be $\mathcal{X}'_i$, where $i = 1, 2, 3, \dots, n$. In this way, it could largely keep the key instances and their neighbors still in the same regions after partition. Then L-SVM trains the SVM classifier $f_i(x)$ over each region $\mathcal{X}'_i$. Finally, $f_i(x)$ only predicts the unlabeled instances in the region $\mathcal{X}_i$ rather than others. An outline of the algorithm is shown in Algorithm 1, and Fig. 1 describes the process of this algorithm.

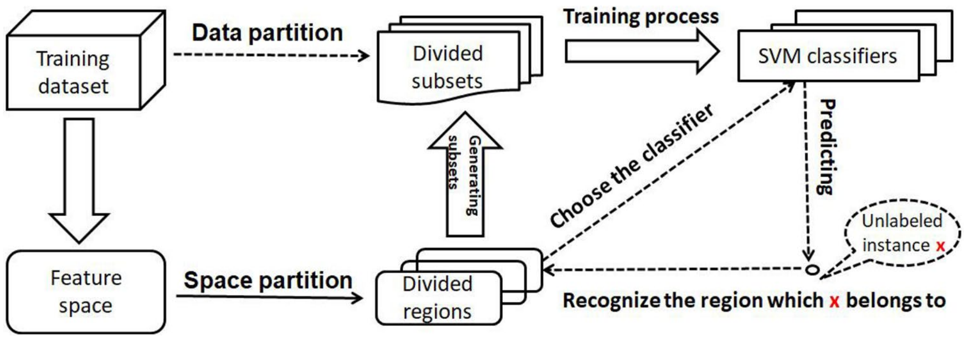

Fig. 1 The flow chart of the L-SVM algorithm

```
Algorithm 1: L-SVM
   Input  : The training set $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$, the number of
            subsets $n$, and the unlabeled instance set $U = \{x'_1, x'_2, \cdots, x'_h\}$.
   Output: The predicted label $y'_1, y'_2, \cdots, y'_h$.
1  Divide the feature space $\mathcal{X}$ into $n$ disjoint regions $\mathcal{X}_i$, and extend the each region to
   be $\mathcal{X}'_i$, $i = 1, 2, \cdots, n$;
   foreach $i = 1$ to $n$ do
2      Train the SVM classifier $f_i(x)$ over the region $\mathcal{X}'_i$;
   end
   foreach $x'_j \in U$ do
       for $i = 1$ to $n$ do
           if $x'_j \in \mathcal{X}_i$ then
3              $y'_j = f_i(x'_j)$
           end
       end
   end
4  Return $y'_1, y'_2, \cdots, y'_h$.
```

Following this way, this algorithm has a great reduction in training time for dividing the original optimization problem into *n* subproblems, and this issue has been proved in the report of the experiments. Meanwhile, it is easy to implement in a parallel environment, since the execution of the SVM algorithm over each subset is performed independently. Additionally, as the size of the subsets is a parameter of this algorithm, we can choose the complexity of the execution in each of the processors.

Dividing the training set T into n subsets T_i is an important step for the L-SVM algorithm. And the number of the subsets n needs to be fixed by the user in advance. Furthermore, the SVM algorithm is applied on each subset independently, the final finishing time of this algorithm depends on the size of the largest subset. So it is necessary to get the size of subsets to be approximately equal. Besides the execution time, generation ability is another key point which we pay attention to. As we all know, SVM has a distinguishable merit that it clarifies which instances are of importance to the training. And these instances are distributed near the class boundary, and fully and succinctly define the classification task at hand [15, 31]. However, what should be noticed is that we should maintain the classification properties of these instances after dividing process: the original boundary classification instances remain the same. In this way, it could largely hold the decision boundary in the original dataset. Furthermore, it is well known that judging whether an instance near the decision boundary or not mainly depends on the consistency between it and its nearest neighbors [33]. If the instance has more heterogeneous neighbors in their class-membership, then it tends to locate near the decision boundary. Otherwise, it tends to be far from the decision boundary. So those key instances could still hold this property in their own divided subsets as the original dataset, as long as we keep them and their heterogeneous neighbors in the same divided subset after data partition. In order to meet the above two requirements, we propose a novel data partition algorithm by linear projection.

4.1 Data partition by linear projection

The locality of the instances could be largely kept by dividing the feature space into some regions. The instances in the same region are divided into as one subset, and most instances and their nearest neighbors could be still in the same subset [34]. To achieve this aim, we project all the instances onto the optimal vector and then divide the projection into equal-sized subsets. Meanwhile, we prove that the algorithm is a kind of feature space partitions using multiple parallel hyperplanes. Furthermore, this data partition algorithm can deal with the large-scale dataset in a short time for its linear time completion. In the following, we will introduce this algorithm specifically.

4.1.1 The obtain of the projected vector

Although this algorithm with a randomly projected vector also holds the locality of the most instances in the dataset, it is difficult for the instances nearby the class boundary. Because this kind of critical instances is much fewer than others in most of the dataset [35]. On the other hand, the linear projection is one of important data visualization algorithms, and it visualizes the high-dimensional data from one of all the possible angles determined by the projected vectors. Then it is useful for identifying structures in the data set, such as clusters and outliers [34, 36]. Furthermore, each base SVM classifier is trained over the divided subset, thus the "equality" of the divided subsets could affect its generation ability. If the instances of different classes in a subset separate well, then the SVM classifier over them could obtain a good classification ability. In order to prompt the divided subsets with the high "equality", the projection vector ω needs to be chosen properly.

Kernel Fisher's linear discriminant (KFLD) is one of these algorithms, it projects the instances onto the optimal vector in the mapped space $\mathcal{X}$, then it tries to separate the instances with different labels after projection. However, KFLD is difficult to deal with the large-scale data for computing the similarity among all the instances. Therefore, we use $\varphi(v)$ to replace the optimal vector for its availability and simplicity, where v is the optimal solution by Fisher's linear discriminant (FLD). We will simply introduce the process of solving the projected vector v in the following.

Let $X_1 = \{x_1^{(1)}, x_2^{(1)}, \dots, x_{n_1}^{(1)}\}$ and $X_2 = \{x_1^{(2)}, x_2^{(2)}, \dots, x_{n_2}^{(2)}\}$ be the instance subsets from class + 1 and − 1 respectably, where n_1 and n_2 are the size of set X_1 and X_2, $n_1 + n_2 = N$. Each instance x is projected onto the vector v, and it is re-expressed by a real number $v^T x$, where v^T is the transposition of the vector v. In FLD, the criteria to measure the separation of different classes is that minimizing the difference on the instances in the same classes A_0 and maximizing the difference between different classes B_0 after projection, and A_0, B_0 are computed following the idea of variance analysis. Then A_0 and B_0 can be computed as follows:

$$\begin{aligned} A_0 &= \sum_{t=1}^{2}\sum_{j=1}^{n_t}(v^T x_j^{(t)} - v^T \overline{x}^{(t)})^2 \\ &= v^T \left[\sum_{t=1}^{2}\sum_{j=1}^{n_t}(x_j^{(t)} - \overline{x}^{(t)})(x_j^{(t)} - \overline{x}^{(t)})^T \right] v \\ &= v^T A v. \end{aligned} \tag{3}$$

$$\begin{aligned} B_0 &= \sum_{t=1}^{2} n_t (v^T \overline{x}^{(t)} - v^T \overline{x})^2 \\ &= v^T \left[\sum_{t=1}^{2} n_t (\overline{x}^{(t)} - \overline{x})(\overline{x}^{(t)} - \overline{x})^T \right] v \\ &= v^T B v. \end{aligned} \tag{4}$$

where $A = \sum_{t=1}^{2}\sum_{j=1}^{n_t}(x_j^{(t)} - \overline{x}^{(t)})(x_j^{(t)} - \overline{x}^{(t)})^T$, $B = \sum_{t=1}^{2} n_t (\overline{x}^{(t)} - \overline{x})(\overline{x}^{(t)} - \overline{x})^T, \overline{x} = \sum_{t=1}^{2}\sum_{j=1}^{n_t} x_j^{(t)}/N, \overline{x}^{(t)} = \sum_{j=1}^{n_t} x_j^{(t)}/n_t$, $t = 1, 2$.

The optimal projected vector v^* is obtained by maximizing the function $\Delta(v) = v^T Bv / v^T Av$, and the constraint condition $v^T Av = 1$ is added to get the unique solution. Solving the above optimal problem, we can get the optimal projected vector v^* that is the eigenvector corresponding to the largest eigenvalue, and $\Delta(v_*) = v_*^T B v_* = \lambda_* v^T A v_* = \lambda_*$.

4.1.2 Data partition

After getting the projected vector $\varphi(v^*)$, we project all the training instances onto it, and then dividing their projection into approximately equal sized subsets. Concretely, under the given threshold s of divided subset size, we first estimate the number of divided subsets $n = [N/s]$, where $[N/s]$ is the smallest integer larger than N/s. Then we compute the (k/n)-quantile b_k of the projection value $p_i = \varphi(v^*)^T x_i = K(v^*, x_i)$ within T, $i = 1, 2, \dots, n$, $k = 1, 2, \dots, n-1$. Finally, according to the relationship between projection value and quantile, the set T is divided into n disjoint subsets T_j as following:

$$T_j = \begin{cases} \{(x_i, y_i) \in T : p_i < b_1\}, & j = 1; \\ \{(x_i, y_i) \in T : b_{j-1} \le p_i < b_j\}, & j = 2, \dots, n-1; \\ \{(x_i, y_i) \in T : p_i \ge b_{n-1}\}, & j = n. \end{cases} \tag{5}$$

In this way, it produces n subsets of equal size, and each subset T_j has about N/n instances with the property of quantile. Furthermore, these divided subsets are the results of dividing the feature space into regions of equal size and using the instances within each region as subsets. In the following, we give the related proof.

To meet the need of proof, we construct $n-1$ hyperplane $L_i : \varphi(v^*)^T x - b_i = 0$ with the same vector direction vector $\varphi(v^*)$. Then these hyperplanes divide the feature space $\mathcal{X}$ into n regions $\mathcal{X}_j$, they are

$$\mathcal{X}_j = \begin{cases} \{x_i \in \mathcal{X} : p_i < b_1\}, & j = 1; \\ \{x_i \in \mathcal{X} : b_{j-1} \le p_i < b_j\}, & j = 2, \dots, n-1; \\ \{x_i \in \mathcal{X} : p_i \ge b_{n-1}\}, & j = n. \end{cases} \tag{6}$$

Furthermore, if we use the instances within each region $\mathcal{X}_j$ as a subset, then the divided subset T_j is obtained, where $j = 1, 2, \dots, n$. So this data partition algorithm is also one kind of feature space partition using these parallel hyperplanes.

4.1.3 The extension of regions

After partitioning the training dataset with the above algorithm, we execute the SVM algorithm on each divided subset T_j of T and obtain n SVM classifiers $f_j(x)$, where $j = 1, 2, \dots, n$. For the divided subset T_j that could preserve the local decision boundary of the training set in the region $\mathcal{X}_j$, the SVM classifier $f_j(x)$ may be a good approximation of the original SVM classifier for the instances $x \in \mathcal{X}_j$. However, the proposed data partition method is difficult to guarantee those critical instances nearby the boundaries (CINB) of divided region and their nearest neighbors in the same divided subsets. So there exist some regions that cannot preserve the decision boundary of the training set within these regions well. To make up this defect, we extend the region $\mathcal{X}_j$ to make CINB far from the boundary of this region. As these divided regions are generated by dividing the feature space with multiple parallel hyperplanes, then we extend these regions by removing the location of these hyperplanes in the feature space.

Therefore, we need to calculate the $((j-\tau)/n)$-quantile u_j and the $((j+\tau)/n)$-quantile d_j besides (j/n)-quantile, where $j = 1, 2, \dots, n-1$. Then the regions can be extended as follows:

$$\mathcal{X}'_j = \begin{cases} \{x_i \in \mathcal{X} : p_i < d_1\}, & j = 1; \\ \{x_i \in \mathcal{X} : u_{j-1} \le p_i < d_j\}, & j = 2, \dots, n-1; \\ \{x_i \in \mathcal{X} : p_i \ge u_{n-1}\}, & j = n. \end{cases} \tag{7}$$

Meanwhile, the divided subset T_j is also enlarged, and the enlarged subsets T'_j is noted as

$$T'_j = \begin{cases} \{(x_i, y_i) \in T : p_i < d_1\}, & j = 1; \\ \{(x_i, y_i) \in T : u_{i-1} \le p_i < d_i\}, & j = 2, \dots, n-1; \\ \{(x_i, y_i) \in T : p_i \ge u_{n-1}\}, & j = n. \end{cases} \tag{8}$$

In the above the operation of extending divided subsets, the parameter τ controls the size of the extend subsets and location of the extended regions in the feature space, and then it affects the classification performance and executing time of the SVM classifier over the subset. Compared with the divided region $\mathcal{X}_j$, each extended region $\mathcal{X}'_j$ increases some instances from its adjacent regions, and these instances are mainly located on the other sides of the boundaries of the region $\mathcal{X}_j$, where $j = 1, 2, \dots, n$. Therefore, these increased instances could be the nearest neighbors of CINB, and they can help the extended region hold the local the decision boundary of the original training set within this region. Finally, the classification performance of the SVM classifiers over the extended subsets can be improved. On the other hand, the size of the largest subset T'_j is about 2τ times larger than the set T_j and then the execution time of training SVM classifier over the extended subset T'_j increases, where $j = 2, 3, \dots, n-1$. Through extensive experiments, we find that $\tau = 0.05$ is appropriate for the trade-off between the improved classification performance and the increased execution time. An outline of the algorithm is shown in Algorithm 2.

Algorithm 2: Data partition based on linear projection (PDP)

```
Input  : Dataset $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$, the subset size $s$ and the
         parameter $\tau$.
Output: The subset $T'_j$, $T = \bigcup T'_j$.
1 Compute the number of subsets $n = [N/s]$;
2 Get the projection vector $\omega^*$ using FLD;
3 Compute the projection $\omega^{*T} x_i$ for each instance $x_i$, where $i = 1, 2, \cdots, N$;
4 Divide the set $T$ into $n$ subsets $T_j$ by the formula (5), $j = 1, 2, 3, \cdots, n$;
5 Enlarge each divided subsets $T_j$ to be $T'_j$ following the formula (8), $j = 1, 2, 3, \cdots, n$;
6 Return $T'_j$.
```

4.2 Parameter selection

The subset size s is a critical parameter for the L-SVM algorithm, and it is proportional to the execution time. The larger value of subset size, the more execution time of the L-SVM algorithm. Furthermore, the divided subset size is also closely related to keeping keep a certain locality in the partition, where keeping the locality is conducive to identify the classification boundary. If the divided subsets have less number of instances after data partition, then it is difficult to achieve this aim. A first natural choice would be the use of a cross-validation procedure. However, automatic determination of the subset size s in a computationally efficient manner is much more difficult due to the inability to reuse computations performed for different values of s. Fortunately, it is a reasonable and effective choice for the subset size $s = N^{0.7}$ in many situations for large-scale data [34, 37].

4.3 Complexity of our algorithm

The aim of this work is to obtain an algorithmology that is able to scale up to large and even huge problems. Thus, an analysis of the complexity of the algorithm is essential. We divide the training dataset of N instances into n disjoint subsets size of $[N/s]$. Let K be the number of operations needed by the SVM algorithm to perform its task in an instance subset of size N/n. For this original dataset, we must perform SVM algorithm process once for each subset, that is, n times, spending a time proportional to nK.

Besides, the data partition of the dataset should be considered apart from training SVM classifiers. This is because that many different algorithms could be devised to perform data partition. PDP algorithm is implemented with a complexity $O(Nm^2 + m^3)$ using the matrix decomposition approach to get the eigenvector, and it divides the training dataset into some subsets of equal size with the time complexity $O(Nlog(N))$, where m is the number of features. So PDP is a linear algorithm with the number of instances and could cope with the larger-scale dataset. Furthermore, PDP algorithm can be improved with a much more efficient solution to inverse matrix when facing the high-dimensional problem [38].

Table 1 Summary of the used data sets

Dataset	Size	Features	Classes
Cifar	60,000	3072	2
Cod-rna	59,535	8	2
Covtype	581,012	54	2
Ijcnn1	141,691	22	2
MiniBooNE	130,065	50	2
Skin-nonskin	245,057	3	2
Susy	5,000,000	18	2
Webspam	350,000	254	2

5 Experimental analysis

5.1 Experimental setup

In order to make a comprehensive comparison between our algorithm and other state-of-the-art algorithms, we have selected a set of 8 benchmark datasets from the Libsvm Repository [39] and UCI Machine Learning Repository [40]. These binary-class datasets have larger than 10,000 instances, and their feature values are normalized in the interval [0,1] to equalize the influence of attributes with different range domains. These datasets are representative of problems from medium to large size. A summary of these datasets is shown in Table 1.

Four representative acceleration algorithms for SVM are selected in this study: LIBSVM [39], early DC-SVM (EDC-SVM) and DC-SVM [13], and DIP-SVM [26]. The selection of these algorithms is based on their representativeness and popularity. LIBSVM algorithm is a classical SVM algorithm with the modified sequential minimal optimization, and it can effectively deal with the large data. EDC-SVM algorithm and DC-SVM algorithm represent the acceleration algorithms with data partition; they obtain the less execution time than many state-of-the-art algorithms, such as CascadeSVM [25], FastFood [29], SpSVM [41], and LLSVM [42].

The estimation performance of algorithms on a set of benchmark problems is the most usual way for evaluating acceleration algorithms for SVM. Accuracy (Acc) [43], Cohen′s Kappa (Kappa) [44] and executing time (*ET*) in s are adopted to evaluate the performance of their simplicity and successful application, where the first two measures evaluate generalization ability and the last one measures their execution time. Acc is the number of successful hits relative to the total number of classifications, and Kappa is a compensate of accuracy and takes random successes into consideration as a standard, in the same way as the AUC measure [44]. The values of Acc and Kappa are both in the interval [0,1], and the classification ability of classifiers is increasing by the enlarging values. For estimating the values of these indexes, we used a k-fold cross-validation method. In this method, the available data is divided into k approximately equal subsets. Then, the algorithm is learned k times, in turn, using each one of the k subsets as a test set, and the remaining $k-1$ subsets as a training set. The final result is the average test result of the k subsets. A fairly standard value for k is $k = 10$.

For evaluating the difference between our algorithm and each of the other algorithms, the Wilcoxon signed rank test [45] is used for its limited commensurability and without the hypothesis normal distributions or homogeneity of variance [46]. The Wilcoxon signed rank test is used to perform a paired, two-sided signed rank test of the null hypothesis that there is no significant difference between our algorithm and each of the other algorithms, against the alternative that there is a significant difference. Under the given significance level α, the p value of the test is computed to judge whether the difference between them exists or not. If the p value is smaller than α, it indicates a rejection of the null hypothesis at the significance level α, against that it indicates a failure to reject the null hypothesis at the significance level α.

We use LIBSVM as the default solver for EDC-SVM, DC-SVM, DIP-SVM, and L-SVM. It is noted that the number of subsets (*NS*) affects the performance of these algorithms, especially for the training time. So we should compare their performance under the same number of subsets. According to the suggestion in [13], DC-SVM divides the data into 4^i subsets at i-th level operation and $NS = 64$ for EDC-SVM, where $i = 1, 2, 3, 4, 5$. As this parameter setting does not consider the size of the original data, so that the efficiency of two algorithms are affected. Meanwhile, it seems that $NS = N^{0.3}$ is an effective choice with the conclusion in [37], where N is the size of the original dataset. Therefore, we compare their performance under the fixed parameter ($NS = 64$) and adaptive parameter $NS = N^{0.3}$. On the other hand, it has been shown in [10] that for most datasets the optimal kernel function for SVM classification is Gaussian kernel function. So we use radial basis function (RBF) $\exp(-\gamma|x_i - x_j|^2)$. Furthermore, We choose the default parameter in LIBSVM. In the following, a significance level of $\alpha = 0.05$ is used. All the experiments are carried out in MATLAB R2013a on Windows 7 running on a PC with system configuration Intel(R) Xeon(R) e5-16200 CPU (3.60 GHz) with 16.00 GB of RAM.

5.2 Experimental results under the fixed *NS*

In this section, we compare the above algorithms performance from the view of the classification performance and training time under the fixed *NS*.

Table 2 Acc and Kappa of five algorithms on 8 datasets

Dataset	L-SVM		EDC-SVM		DC-SVM		DIP-SVM		LIBSVM	
	Acc	Kappa	Acc	Kappa	Acc	Kappa	Acc	Kappa	Acc	Kappa
Cifar	0.961	0.909	0.919	0.912	0.969	0.914	0.968	0.914	0.966	0.929
Cod-rna	0.960	0.906	0.914	0.796	0.923	0.818	0.915	0.886	0.922	0.816
Covtype	0.857	0.698	0.839	0.678	0.842	0.685	0.843	0.698	0.806	0.682
Ijcnn1	0.956	0.732	0.949	0.613	0.970	0.803	0.958	0.673	0.969	0.794
MiniBooNE	0.899	0.788	0.827	0.632	0.836	0.656	0.864	0.742	0.855	0.692
Skin-nonskin	0.998	0.985	0.994	0.981	0.993	0.979	0.994	0.984	0.994	0.981
Susy	0.812	0.587	0.796	0.582	0.800	0.595	0.793	0.584	0.796	0.583
Webspam	0.956	0.916	0.968	0.934	0.965	0.926	0.968	0.924	0.967	0.930
Average	0.925	0.815	0.901	0.766	0.912	0.797	0.913	0.801	0.909	0.801
Median	0.956	0.847	0.917	0.737	0.944	0.811	0.937	0.814	0.944	0.805

5.2.1 Classification performance

Table 2 lists the classification performance measured by Acc and Kappa of these algorithms on different datasets.

Table 2 indicts that L-SVM obtains a higher classification accuracy than other algorithms, especially for data sets including Cod-rna, Covtype, MiniBooNE and Susy. And it is not worse than other algorithms on the rest of datasets. Moreover, the mean of Acc of these algorithms are 0.925, 0.901, 0.912, 0.913 and 0.909, as well as their median of Acc are 0.956, 0.917, 0.944, 0.937 and 0.944 respectively in the last row of Table 1. So L-SVM obtains the similar classification accuracy as LIBSVM, DIP-SVM and DC-SVM, but better than EDC-SVM.

For another classification performance measurement Kappa, L-SVM is better than other algorithms on Cod-rna and MinniNooNE datasets, and it is not worse than others on the rest of datasets from the Table 2. Meanwhile, the mean of Kappa of these algorithms on different datasets are 0.815, 0.766, 0.797, 0.801 and 0.801, and their medians of Kappa are 0.847, 0.737, 0.811, 0.814 and 0.805. It obviously shows that L-SVM obtains the similar Kappa as LIBSVM, EDC-SVM and DIP-SVM, but better than EDC-SVM.

Finally, in order to provide an accurate evaluation of the probability of obtaining the observed outcomes by chance, the Wilcoxon signed rank test is used to compare the performance between L-SVM and each one of these four algorithms. The p values of the Wilcoxon signed rank test on Acc are 0.039, 0.313, 0.195 and 0.250, and p values for Kappa are 0.078, 0.641, 0.297 and 0.617.

In conclusion, L-SVM exists a significant difference in classification performance with EDC-SVM, and no significant differences with other algorithms. In the L-SVM algorithm, it largely keeps the training instances and their nearest neighbors with different labels in the same divided regions, and then each divided region can hold the original decision boundary over this region. Therefore, it could hold the original classification boundary, and obtains the similar predictive ability as LIBSVM, EDC-SVM and DIP-SVM.

Table 3 *RS* of four algorithms on 8 datasets

Dataset	L-SVM	EDC-SVM	DC-SVM	DIP-SVM
Cifar	33.50	6.44	1.73	16.96
Cod-rna	176.21	5.72	2.29	13.61
Covtype	87.81	33.29	10.01	64.02
Ijcnn1	123.23	10.28	2.27	27.05
MiniBooNE	214.70	21.68	1.78	38.03
Skin-nonskin	125.61	4.35	1.72	12.42
Susy	2.61	1.34	1.10	1.38
Webspam	84.99	24.48	1.26	64.41
Mean	106.08	13.45	2.77	29.73
Median	105.52	10.28	1.78	22.01

5.2.2 The training time

Besides the classification performance, the executing time is another measurement to evaluate the performances of these algorithms. As is well-known, algorithms with less executing time are more suitable for dealing with the large-scale problems in application. Table 3 lists the relative speed (*RS*) of these four algorithms. The relative speed is defined as the ratio of executing time (in s) between each of four three decomposition algorithms and LIBSVM on each dataset.

In Table 3, it is obviously that RS of L-SVM is much larger than that of other four algorithms on the most of the datasets. So L-SVM has the least execution time among four algorithms on these datasets. Besides, the p values obtained by Wilcoxon signed rank test between L-SVM and each one of other algorithms are 0.0078, 0.0078, 0.0078, less than the given significant level 0.05. So L-SVM and other algorithms exist the significant differences in training time. The reason for this issue is that they have different training process. L-SVM trains SVM classifiers over the divided subsets of

Table 4 *R* and *ET* of two ways on 8 datasets

Dataset	PDP		KKCDP	
	R	*ET*	*R*	*ET*
Cifar	0.02	37.70	0.98	8.50
Cod-rna	0.02	0.09	0.40	6.92
Covtype	0.02	0.63	0.09	13.8
Ijcnn1	0.02	0.07	0.10	3.3
MiniBooNE	0.02	0.09	0.09	2.11
Skin-nonskin	0.02	0.04	0.24	5.24
Susy	0.02	2.64	0.05	115.61
Webspam	0.02	0.72	0.14	10.98

the entire dataset just once, while both DC-SVM and DIP-SVM need multilevel training process, and they use the divided subsets of the entire dataset or support vectors set at each level.

Though EDC-SVM has the similar training process, the sizes of divided subsets are quite different. As we all know, the executing time of the EDC-SVM and L-SVM with data partition depends on the size of the largest subset. So EDC-SVM has the longer training time than L-SVM. In order to corroborate this statement, we compute the ratio (R) between the size of the largest divided subset and the size of the original dataset, and use its value on different datasets to corroborate the above statement. Furthermore, we also compare the execution time of PDP and kernel K-means clustering partition (KKCDP). Table 4 lists *ET* (in s) and R of PDP and KKCDP on different datasets.

It shows that the R of PDP has a constant value, and it is smaller than KKCDP on all the datasets in Table 4. Especially, the value of R obtained by KKCDP is tens of times smaller than PDP on Cifa, Skin-noskin and Cod-rna datasets, and the value of *RS* of L-SVM is also tens of times shorter than EDC-SVM on these datasets. Meanwhile, the p value of the test on R between PDP and KKCPD is 0.0078, less than 0.05, so it exists a significant difference on R under the given significant level 0.05. So this result corroborates that the execution time of L-SVM is much shorter than EDC-SVM on all the datasets.

Furthermore, we also compare the execution time of PDP and KKCDP. It indices that the execution time of PDP is less than KKCDP on all the datasets except Cifa in Table 4. And the p value of test on *ET* between PDP and KKCPD is 0.0078, less than 0.05, so it exists significant difference on *ET* under the given significant level 0.05. The time complexity of PDP and KKCDP are $O(Nm^3)$ and $O(Nnm + n^2m)$ respectively, where m is the number features in the dataset and n is the number of sampled instances. Although both PDP and KKCDP have the linear time complexity with the number of training instances, KKCDP needs many iterations to get the final result. So the execution time of KKCDP is much longer than PDP on the low-dimensional dataset. Because PDP costs a lot of time to get the optimal eigenvector for the high-dimensional problem, then its execution time is longer than KKCDP on dataset Cifa. Compared with the time of training the SVM classifiers, the execution time of PDP is a small fraction of the total execution time of the L-SVM algorithm. Therefore, the L-SVM algorithm can effectively deal with the high-dimensional problem, and this issue is verified by the result on data Cifa in Table 3.

5.3 Experimental results under the adaptive *NS*

Different from the above section, we compare our algorithm with other four algorithms under adaptive *NS* in this section.

5.3.1 Classification performance

Table 5 lists Acc and Kappa of these four algorithms on different datasets.

Compared with other algorithms, L-SVM has the largest value of Acc on Cod-rna and MiniBooNE datasets, and it

Table 5 Acc and Kappa of four algorithms on 8 datasets

Dataset	L-SVM		EDC-SVM		DC-SVM		DIP-SVM	
	Acc	Kappa	Acc	Kappa	Acc	Kappa	Acc	Kappa
Cifar	0.962	0.911	0.948	0.881	0.965	0.917	0.958	0.904
Cod-rna	0.962	0.911	0.917	0.818	0.933	0.828	0.953	0.888
Covtype	0.840	0.680	0.829	0.678	0.845	0.687	0.845	0.687
Ijcnn1	0.960	0.752	0.949	0.610	0.956	0.773	0.961	0.773
MiniBooNE	0.900	0.789	0.831	0.642	0.841	0.686	0.891	0.756
Skin-nonskin	0.991	0.975	0.993	0.978	0.994	0.979	0.994	0.979
Susy	0.796	0.582	0.792	0.576	0.801	0.565	0.806	0.568
Webspam	0.968	0.934	0.953	0.931	0.961	0.936	0.960	0.937
Average	0.922	0.817	0.902	0.764	0.912	0.796	0.921	0.812
Median	0.961	0.850	0.933	0.748	0.945	0.801	0.956	0.831

Table 6 *RS* of four algorithms on 8 datasets

Dataset	L-SVM	EDC-SVM	DC-SVM	DIP-SVM
Cifar	8.71	2.25	1.56	5.64
Cod-rna	274.41	69.67	29.92	125.63
Covtype	89.09	24.91	10.64	48.24
Ijcnn1	70.41	9.59	5.20	45.69
MiniBooNE	110.85	15.63	12.23	55.98
Skin-nonskin	78.16	3.92	6.28	48.61
Susy	2.64	1.70	1.33	1.96
Webspam	77.38	17.16	7.27	35.69
Mean	88.96	18.11	9.30	45.93
Median	77.77	12.61	6.77	46.97

Table 7 *R* and *ET* of two ways on 8 datasets

Dataset	PDP		KKCDP	
	R	*ET*	*R*	*ET*
Cifar	0.04	37.89	0.98	8.79
Cod-rna	0.02	0.10	0.42	7.04
Covtype	0.02	0.63	0.07	14.23
Ijcnn1	0.03	0.07	0.10	3.41
MiniBooNE	0.03	0.09	0.09	2.14
Skin-nonskin	0.03	0.04	0.22	5.32
Susy	0.01	2.67	0.03	120.29
Webspam	0.02	0.74	0.16	11.03

has no less value of Acc than others on the rest datasets in Table 5. Meanwhile, the mean of Acc of these algorithms are 0.922, 0.902, 0.912, 0.922 and 0.909, as well as their median of Acc are 0.961, 0.933, 0.945,0.956 and 0.944, respectively in the last row of Table 5. On average, L-SVM obtains the similar classification accuracy as DC-SVM, DIP-SVM and LIBSVM, DC-SVM but better than EDC-SVM. In term of Kappa, L-SVM has the largest value on Cod-rna and MiniBooNE datasets, and it has the similar value on the rest datasets in Table 5. Furthermore, we also find that L-SVM has the larger value of Kappa than EDC-SVM from the view of mean and median, and it has the similar Kappa value with DC-SVM, DIP-SVM and LIBSVM. Finally, the p values of the Wilcoxon signed rank test on Acc between L-SVM and each one of four algorithms are 0.016, 0.445, 0.711 and 0.469; similarly, the p values on Kappa are 0.031, 0.844 , 0.484 and 1. According to the judging rule, L-SVM has a significant difference on Acc and Kappa with EDC-SVM, while it has no significant difference with DC-SVM, LIBSVM and DIP-SVM. So L-SVM obtains the better classification than EDC-SVM, and it could match DC-SVM, DIP-SVM and LIBSVM.

5.3.2 The training time

Besides the classification performance, the training time is another important measure to evaluate the performance of algorithms. Similarly as Table 3, Table 6 shows the *RS* on different datasets for L-SVM, EDC-SVM, DC-SVM and DIP-SVM.

It shows that L-SVM has the largest *RS* on all the datasets than EDC-SVM, DC-SVM and DIP-SVM. In fact, the *RS* of L-SVM is larger tens times than DC-SVM and EDC-SVM on dataset Skin-noskin, and it has several times faster than them on the rest of datasets. Therefore, L-SVM has the least training time among four algorithms on these datasets. Besides, the p values obtained by Wilcoxon signed rank test between L-SVM and each one of other algorithms are all 0.0078, less than the given significant level 0.05. So L-SVM and other algorithms exist a significant difference on training time.

As the execution time of these algorithm depends on the size of the largest divided subset, we compute the ratio of size *R* between the largest divided subset and original data to corroborate the above conclusion. Furthermore, We also compare the execution time of two kinds of data partition. Table 7 lists *ET* and *R* of PDP and KKCDP over different datasets.

It obviously shows that PDP has a much smaller value of *R* than KKCDP on all the datasets from Table 7. Specifically, the value of *R* obtained by KKCDP is tens of times larger than PDP on Cifa and Cod-rna datasets, and several times larger than PDP on the rest of datasets. So the size of largest divided subsets produced by PDP is smaller than KKCDP, and the time spent by L-SVM is also less than EDC-SVM and DC-SVM. Meanwhile, the execution time of PDP is also much less than KKCDP on all the datasets except Cifa. As the time complexity of PDP is the third power of the dimension of data, then its execution time becomes longer for the high-dimension dataset Cifa. However, the execution time of PDP algorithm is a small fraction of the total execution time of the L-SVM algorithm, it can effectively deal with the high-dimensional problem.

6 Conclusion

In this paper, we propose a novel algorithm for accelerating SVM. It is applicable to the similar algorithms that training learners use the geometric structure information of dataset without any modification. This algorithm divides the training dataset into some subsets of approximately equal size with the linear projection, and it could largely hold the class boundary as far as possible. In the predicted process, the divided regions to which the test instances belong are firstly recognized, then they are classified by the SVM classifiers over the recognized regions. Experiments show that the proposed algorithm is able to match the classification performance of four state-of-the-art SVM acceleration algorithms while it has the least training time among them.

The divide-and-conquer approach is one of the most commonly used ways to deal with the large-scale problem. Hence, this paper provides an efficient algorithm for dividing the difficult problem into some feasible subproblems and combining their solutions. Additionally, the proposed algorithm provides a promising way for the large-scale classification problem, such as face recognition, text detection and categorization, sentiment classification and so on. It is noted that the number of divided subsets should be adaptively determinate the optimal values for different datasets. Besides, our algorithm does not fully consider the high-dimension problems. Then these problems are two parts of our future work.

Acknowledgements This work was supported by the National Natural Science Foundation of China (No. 61432011, No. U1435212, and No. 61876103), the Project of Key Research and Development Plan of Shanxi Province (201603D111014), and the 1331 Engineering Project of Shanxi Province, China.

References

1. Bosner B, Guyon I, Vapnik V (1992) A training algorithm for optimal margin classifier. In: Proceedings of the 5th annual ACM workshop on computational learning theory, pp 144–152
2. Doran G, Ray S (2014) A theoretical and empirical analysis of support vector machine methods for multiple-instance classification. Mach Learn 97(1–2):79–102
3. Chen W, Shao Y, Hong N (2014) Laplacian smooth twin support vector machine for semi-supervised classification. Int J Mach Learn Cybern 5(3):459–468
4. Li C, Huang Y, Wu H, Shao Y, Yang Z (2016) Multiple recursive projection twin support vector machine for multi-class classification. Int J Mach Learn Cybern 7(5):729–740
5. Abe S (2016) Fusing sequential minimal optimization and newtons method for support vector training. Int J Mach Learn Cybern 7(3):345–364
6. Yang Z, Wu H, Li C, Shao Y (2016) Least squares recursive projection twin support vector machine for multi-class classification. Int J Mach Learn Cybern 7(3):411–426
7. Peng X, Kong L, Chen D (2017) A structural information-based twin-hypersphere support vector machine classifier. Int J Mach Learn Cybern 8(1):295–308
8. Ding S, Zhu Z, Zhang X (2017a) An overview on semi-supervised support vector machine. Neural Comput Appl 28(5):969–978
9. Ding S, Zhang X, An Y, Xue Y (2017b) Weighted linear loss multiple birth support vector machine based on information granulation for multi-class classification. Pattern Recognit 67:32–46
10. Fernández-Delgado M, Cernadas E, Barro S, Amorim D (2014) Do we need hundreds of classifiers to solve real world classification problems? J Mach Learn Res 15(1):3133–3181
11. Cachin C (1994) Pedagogical pattern selection strategies. Neural Netw 7(1):175–181
12. Foody GM (1999) The significance of border training patterns in classification by a feedforward neural network using back propagation learning. Int J Remote Sens 20(18):3549–3562
13. Hsieh CJ, Si S, Dhillon IS (2014) A divide-and-conquer solver for kernel support vector machines. In: Proceedings of the 31th international conference on machine learning, pp 566–574
14. Do TN, Poulet F (2015) Random local SVMS for classifying large datasets. In: Proceedings of the second international conference on future data and security engineering, pp 3–15
15. Poggio T, Cauwenberghs G (2001) Incremental and decremental support vector machine learning. In: Advances in neural information processing systems, pp 409–415
16. Pontil M, Verri A (1998) Properties of support vector machines. Neural Comput 10(4):955–974
17. Koggalage R, Halgamuge S (2004) Reducing the number of training samples for fast support vector machine classification. Neural Inf Process Lett Rev 2(3):57–65
18. Lyhyaoui A, Martinez M, Mora I, Vaquez M, Sancho JL, Figueiras-Vidal AR (1999) Sample selection via clustering to construct support vector-like classifiers. IEEE Trans Neural Netw 10(6):1474–1481
19. Angiulli F, Astorino A (2010) Scaling up support vector machines using nearest neighbor condensation. IEEE Trans Neural Netw 21(2):351–357
20. Li Y, Maguire L (2011) Selecting critical patterns based on local geometrical and statistical information. IEEE Trans Pattern Anal Mach Intell 33(6):1189–1201
21. Wang J, Wonka P, Ye J (2013) Scaling SVM and least absolute deviations via exact data reduction. Comput Sci 2013:523–531
22. Pan X, Yang Z, Xu Y, Wang L (2018a) Safe screening rules for accelerating twin support vector machine classification. IEEE Trans Neural Netw Learn Syst 29(5):1876–1887
23. Pan X, Pang X, Wang H, Xu Y (2018b) A safe screening based framework for support vector regression. Neurocomputing 287:163–172
24. Collobert R, Bengio S, Bengio Y (2002) A parallel mixture of SVMS for very large scale problems. Neural Comput 14(5):1105–1114
25. Graf HP, Cosatto E, Bottou L, Dourdanovic I, Vapnik V (2004) Parallel support vector machines: The cascade SVM. In: Advances in neural information processing systems, pp 521–528
26. Singh D, Roy D, Mohan CK (2017) Dip-SVM: distribution preserving kernel support vector machine for big data. IEEE Trans Big Data 3(1):79–90
27. Keerthi SS, Chapelle O, DeCoste D (2006) Building support vector machines with reduced classifier complexity. J Mach Learn Res 7(Jul):1493–1515
28. Zhang K, Lan L, Wang Z, Moerchen F (2012) Scaling up kernel SVM on limited resources: A low-rank linearization approach. In: Artificial intelligence and statistics, pp 1425–1434
29. Le Q, Sarlós T, Smola A (2013) Fastfood-approximating kernel expansions in loglinear time. In: Proceedings of the 30th international conference on machine learning, pp 16–21
30. Jose C, Goyal P, Aggrwal P, Varma M (2013) Local deep kernel learning for efficient non-linear SVM prediction. In: Proceedings of the 30th international conference on machine learning, pp 486–494
31. Vapnik V (2013) The nature of statistical learning theory. Springer, New York
32. Joachims T (2006) Training linear SVMs in linear time. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 217–226
33. Shin H, Cho S (2007) Neighborhood property-based pattern selection for support vector machines. Neural Comput 19(3):816–855
34. García-Osorio C, de Haro-García A, García-Pedrajas N (2010) Democratic instance selection: a linear complexity instance selection algorithm based on classifier ensemble concepts. Artif Intell 174(5):410–441

35. Garcia S, Derrac J, Cano J, Herrera F (2012) Prototype selection for nearest neighbor classification: taxonomy and empirical study. IEEE Trans Pattern Anal Mach Intell 34(3):417–435
36. Asimov D (1985) The grand tour: a tool for viewing multidimensional data. SIAM J Sci Stat Comput 6(1):128–143
37. Kleiner A, Talwalkar A, Sarkar P, Jordan MI (2014) A scalable bootstrap for massive data. J R Stat Soc Ser B (Stat Methodol) 76(4):795–816
38. Zhang X (2004) Matrix analysis and application. Tsinghua University Press, Beijing
39. Chang CC, Lin CJ (2011) Libsvm: a library for support vector machines. ACM Trans Intell Syst Technol 2(3):27
40. Bache K, Lichman M (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml/datasets.html
41. Kugler M, Kuroyanagi S, Nugroho AS, Iwata A (2006) Combnet-iii: a support vector machine based large scale classifier with probabilistic framework. IEICE Trans Inf Syst 89(9):2533–2541
42. Wang Z, Djuric N, Crammer K, Vucetic S (2011) Trading representability for scalability: adaptive multi-hyperplane machine for nonlinear classification. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining, pp 24–32
43. Han J, Pei J, Kamber M (2011) Data mining: concepts and techniques. Margan Kaufmann, San Francisco
44. Ben-David A (2007) A lot of randomness is hiding in accuracy. Eng Appl Artif Intell 20(7):875–885
45. Wilcoxon F (1992) Individual comparisons by ranking methods. Springer, New York
46. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7(Jan):1–30