International Journal of Computational Intelligence Systems
Vol. **13(1)**, 2020, *pp.* 212–222
DOI: https://doi.org/10.2991/ijcis.d.200205.001; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Sparse Least Squares Support Vector Machine With Adaptive Kernel Parameters

Chaoyu Yang[1], Jie Yang[2,*], Jun Ma[3]

[1]*School of Economics and Management, Anhui University of Science and Technology, Huainan, 232001, China*
[2]*School of Computing and Information Technology, Faculty of Engineering and Information Sciences, University of Wollongong, Wollongong, NSW, 2522, Australia*
[3]*Operations Delivery Division, Sydney Trains, Alexandria, NSW, 2015, Australia*

ARTICLE INFO

Article History
Received 19 Nov 2019
Accepted 28 Jan 2020

Keywords
Least squares support vector machine
Sparse representation
Dictionary learning
Kernel parameter optimization

ABSTRACT

In this paper, we propose an efficient Least Squares Support Vector Machine (LS-SVM) training algorithm, which incorporates sparse representation and dictionary learning. First, we formalize the LS-SVM training as a sparse representation process. Second, kernel parameters are adjusted by optimizing their average coherence. As such, the proposed algorithm addresses the training problem via generating the sparse solution and optimizing kernel parameters simultaneously. Experimental results demonstrate that the proposed algorithm is capable of achieving competitive performance compared to state-of-the-art approaches.

1. INTRODUCTION

Least Squares Support Vector Machine (LS-SVM) algorithm has received a great deal of attention over the last two decades. By mapping the input data into a high-dimension feature space, LS-SVM algorithm is able to learn complex non-linear relationship from training samples. Due to this advanced learning capability, relevant application has spanned several disciplines, such as pattern recognition, decision making, and statistical modeling. The wide applicability also has motivated growing interest in developing efficient LS-SVM training algorithms with fast convergence and good generalization ability.

One major drawback with conventional LS-SVM based algorithms is the solution's sparsity, in which a great number of support vectors (SVs) are required in the resultant model [1–3]. The SVs are typically a portion of training samples, used to determine the decision boundary. However, too many SVs might have a negative impact on the generalization ability of the trained model, not to mention larger computational cost. In addition, computational complexity scales with the increasing number of SVs. To address the problem associated with large number SVs, a variety of sparsity-enhanced approaches have been applied, where significant SVs are determined and selected to form the decision boundary. For instance, Suykens *et al.* first proposed removing training samples that have the smallest absolute support values [2]. However, this method might eliminate training samples near the decision boundary, which has a negative influence on the training performance. An improved method was proposed in [3], in which a reduced training set is only comprised of samples near the decision boundary. Another sparsity-enhanced method was proposed in [4], in which least-important SVs are removed through regularization (*i.e.*, Nyström approximation) to accelerate the training process. In [5], SVs were eliminated using single and multi-objective evolutionary computation approaches, without affecting the classifier accuracy and even improving it.

Another problem of traditional LS-SVM training is to determine the suitable kernel parameters, that are associated with the employed kernel function. For instance, in the Gaussian Radial Basis Function (RBF), the kernel parameter is the smoothing parameter (also known as the kernel size or bandwidth). Then, before any training process, kernel parameters will need to be predetermined and fixed. However, the parameter setting is purely data-driven, as the optimal value varies case-by-case. Additionally, the resultant performance using different settings of kernel parameters can greatly differ from each other. As there is not a rule of thumb to determine the optimal kernel parameters, trial-and-error or cross-validation experiments are often required,which can be very time-consuming.

In our previous work [6], we have presented one sparse training algorithm, in which the LS-SVM training is reformulated as a sparse representation process. As such, the training is accomplished by iteratively finding important SVs that minimize the training error.

* *Corresponding author. Email: jiey@uow.edu.au*

In this paper, additionally, we introduce a dictionary-learning inspired technique to further optimize the kernel-based training. The aim is to enhance the model sparsity without compromising the training performance. Toward this end, a kernel matrix is firstly established using training samples, and later formulated as the dictionary. Then the dictionary-learning algorithm is employed to explore the optimal parameter via minimizing the average coherence of this kernel matrix (*i.e.*, dictionary). When the sparse representation and dictionary learning techniques are combined, the proposed algorithm is able to perform model training, SVs selection, and kernel parameters optimization simultaneously.

The remainder of the paper is organized as follows. Section 2 gives a brief introduction about kernel-based learning algorithms (such as Support Vector Machine [SVM] and LS-SVM). Section 3 provides the basic concept about sparse representation and dictionary learning, respectively. Section 4 presents the proposed sparsity-enhanced training approach, including applying the sparse representation technique to select important SVs and optimizing kernel parameters based on the dictionary-learning concept. Section 5 compares the proposed algorithm with conventional training methods using several benchmark problems. Section 6 presents concluding remarks.

2. KERNEL-BASED LEARNING ALGORITHMS

In this section, we first introduce the general formulation of the kernel-based learning algorithms. Then the SVMs and LS-SVMs will be discussed as two of the most popular kernel learning techniques, respectively.

Recent years have witnessed the rapid development of the kernel-based learning methods, and their wide applications in several applications, such as classification, pattern recognition, and many more. Generally speaking, the fundamental strategy of kernel-based learning is to map raw data from the original space into a Hilbert space with higher dimension, using some kernel functions. The advantage is that in the Hilbert space, more effective separation hyperplanes are expected to exist compared to the original space.

The most-used kernel functions are liner, polynomial, and RBF kernels. Among various kernels, RBF kernel is very popular and usually used as a default choice due to its desirable smoothness and numeric stability. Therefore, in this paper the RBF function is employed which is expressed as

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2^2}{2\sigma^2}\right), \tag{1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the i and j-th input samples, and σ is the smoothing parameter (also known as the kernel size).

Before training, however, we need to pre-determine the value of the kernel size or σ. In general, trial-and-error or cross-validation experiments are often required, which can be computationally demanding. To reduce the cost, some research work has been conducted, including Maximum Corr-entropy Criterion [7], Bayesian averaging rule [8], and stochastic gradient method [9], to optimize the kernel size during the training process.

2.1. Support Vector Machines

As one of the most popular kernel-based approaches, the SVM algorithm has been demonstrated to perform well in various practical applications. The SVM model is fundamentally formulated to address two-class classification problems. The decision boundary is constructed by finding a hyperplane that achieves the maximum separation between two classes. Suppose that we have a training set consisting of N samples $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is the i-th input sample and $y_i \in \{1, -1\}$ is the class label. Assume that the data is linearly separable in the original space, then the decision function can be written as

$$y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geqslant 1, \tag{2}$$

where $\boldsymbol{w}$ is the weight vector, b is a bias term, and $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ is the dot product of the vectors $\boldsymbol{w}$ and $\boldsymbol{x}$.

Note that there exist more than one hyperplanes that can separate the two classes; however, only one of them, termed *optimal separating hyperplane*, can maximize the margins between the two classes. Consequently, the SVM training is to find $\boldsymbol{w}$ and b that achieve the maximum separation:

$$\min \mathcal{J}(\boldsymbol{w}) = \frac{1}{2} \parallel \boldsymbol{w} \parallel^2 \ \ s.t.\, y_i\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\right) \geqslant 1, \forall i \in [1, \ldots, N]\,. \tag{3}$$

The parameter optimization in Eq. (3) is built on the assumption of linear separability of the training data. However, if the problem is not linearly separated in the original space, an SVM classifier with a linear decision boundary will have a poor generalization ability. To improve the classification accuracy, the data samples are usually projected from the input space to a higher-dimensional space via a mapping function $\varphi(\cdot)$. As a result, a non-linear decision boundary can be constructed for classification.

2.2. Least Squares Support Vector Machines

LS-SVMs have been developed to achieve more affordable and faster training. As a variant of SVM, in the LS-SVM model, the inequality constraints in Eq. (3) are replaced with equality constraints, and the unknown parameters are then obtained by solving the following problem:

$$\begin{aligned} &\min\ \mathcal{J}\,(\boldsymbol{w}, b, \boldsymbol{e}) = \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + \frac{\gamma}{2}\sum_{i=1}^{N} e_i^2, \\ &s.t. \quad y_i = \boldsymbol{w}^{\mathrm{T}}\varphi\,(\boldsymbol{x}_i) + b + e_i, \quad \forall i \in [1, 2, \ldots, N], \end{aligned}$$

where $\boldsymbol{e}$ is the error vector, and γ is a regularization parameter. This problem can be solved using the Lagrange multiplier method:

$$\begin{aligned} &\min\ \mathcal{L}\,(\boldsymbol{w}, b, \boldsymbol{e}, \boldsymbol{\alpha}) = \mathcal{J}\,(\boldsymbol{w}, b, \boldsymbol{e}) \\ &+ \sum_{i=1}^{N} \boldsymbol{\alpha}_i[y_i - \boldsymbol{w}^{\mathrm{T}}\varphi\,(\boldsymbol{x}_i) - b - e_i], \end{aligned} \tag{4}$$

where α_i ($i = 1, 2, \ldots, N$) are the Lagrange multipliers. According to the Karush–Kuhn–Tucker conditions, the minimization of Eq. (4) satisfies

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w} &= 0 \rightarrow w = \sum_{i=1}^{N} \alpha_i \varphi(x_i), \\
\frac{\partial \mathcal{L}}{\partial b} &= 0 \rightarrow \sum_{i=1}^{N} \alpha_i = 0, \\
\frac{\partial \mathcal{L}}{\partial e_i} &= 0 \rightarrow \alpha_i = \gamma e_i, \quad \forall i \in [1, 2, \ldots, N], \\
\frac{\partial \mathcal{L}}{\partial \alpha_i} &= 0 \rightarrow w^T \varphi(x_i) + b = y_i - e_i, \quad \forall i \in [1, 2, \ldots, N].
\end{aligned} \tag{5}$$

Furthermore, the above conditions lead to a linear system of equations after eliminating w and e:

$$\left[\begin{array}{c|c} Q + \gamma^{-1} I_N & \mathbf{1}_N \\ \hline \mathbf{1}_N^T & 0 \end{array}\right] \left[\begin{array}{c} \alpha \\ \hline b \end{array}\right] = \left[\begin{array}{c} y \\ \hline 0 \end{array}\right], \tag{6}$$

where Q is the kernel matrix with $Q_{ij} = \varphi(x_i)^T \varphi(x_j)$, the vector $\mathbf{1}_N$ is the N-dimensional vector whose all elements are equal to 1, and I_N is the $N \times N$ identity matrix.

With the application of Mercer's theorem, there is no need to compute explicitly the mapping $\varphi(\cdot)$ as this can be done implicitly through the use of positive-definite kernel functions $\mathcal{K}(\cdot, \cdot)$, that is, $Q_{ij} = \mathcal{K}(x_i, x_j)$. The resulting LS-SVM model for non-linear function then becomes

$$y_i = \sum_{j=1}^{N} \alpha_j \mathcal{K}(x_i, x_j) + b + e_i, \quad i = 1, 2, \ldots, N. \tag{7}$$

The traditional training algorithm for LS-SVM is to use the conjugate gradient (CG). One common problem comes from the large number of SVs required in training, which may influence the generalization capability. Therefore, several optimization methods have been suggested to improve the sparsity of the LS-SVM model (*i.e.*, minimizing the number of required SVs). Some of those early studies include [1–3] (mentioned in Section 1).

More recent studies include CLS-SVM [6], SSS [10], and SLQ [11]. For instance, in [6], the CLS-SVM algorithm is the first attempt to train the LS-SVM using the concept of the sparse representation. Additionally, the SSS algorithm in [10] conducts the training process based on a graph Laplacian matrix L from input samples. They first define a spectral similarity matrix W by measuring the spectral similarity among N samples, then the Laplacian matrix is defined by $L = W^d - W$, where W^d is a diagonal matrix with the element $W_{ii}^d = \sum_j w_{ij}$ ($\forall i \in [1, N]$). Later, a sparse model is obtained using the Laplacian matrix L. At last, SLQ [11] is a sparse l_q-norm based LS-SVM (with $0 < q < 1$). Their major contribution is to reformulate the training process using the l_q-norm based minimization.

3. SPARSE REPRESENTATION AND DICTIONARY LEARNING

Over the past decade, signal sparse representations have received extensive research interests across several communities, including image processing [12], machine learning [13], and simulation [14]. The fundamental concept underlying sparse representation indicates that a signal can be approximated by a linear combination of a few elementary signals. Alternatively, few elements are capable of representing the majority information conveyed by the target signal.

3.1. Single Measurement Vector Model

The *single measurement vector* (SMV) model is one typical model for sparse representation. The aim of the SMV model is to recover a sparse signal $x \in \mathbb{R}^N$ from a few linear measurements $y \in \mathbb{R}^M$. Formally, the SMV model is expressed as

$$\min S(x) \quad s.t. \quad y = Dx, \tag{8}$$

where $S(x)$ denotes a sparsity measure, and $D \in \mathbb{R}^{M \times N}$ is known as the *dictionary*. The SMV model can be illustrated in Figure 1.

To measure the sparsity $S(x)$, one simple strategy is to use the l_1-norm of x which is the sum of the magnitudes of the vector x, *i.e.*, $S(x) = \|x\|_1$. The SMV model of Eq. (8) can be rewritten as follows:

$$\min \|x\|_1 \quad s.t. \quad y = \mathcal{D}_x. \tag{9}$$

A variety of algorithms have been developed to solve the SMV model. For instance, the *Orthogonal Matching Pursuit* (OMP) algorithm is a popular algorithm that offers a good sparse solution [15]. During the iterative process, it will first measure the similarity between the residual error and the dictionary columns, and then select the column that minimizes the error most. In this paper, we will utilize OMP in our experiments to calculate the sparse solution.

3.2. Dictionary Learning

Solving the SMV model, however, relies heavily on the employed dictionary structure, or the degree of fitting between the data and the dictionary [16,17]. A randomly initialized dictionary $\mathcal{D}$ may not be guaranteed to perform an accurate sparse representation. An well-designed dictionary, on the other hand, will lead to an

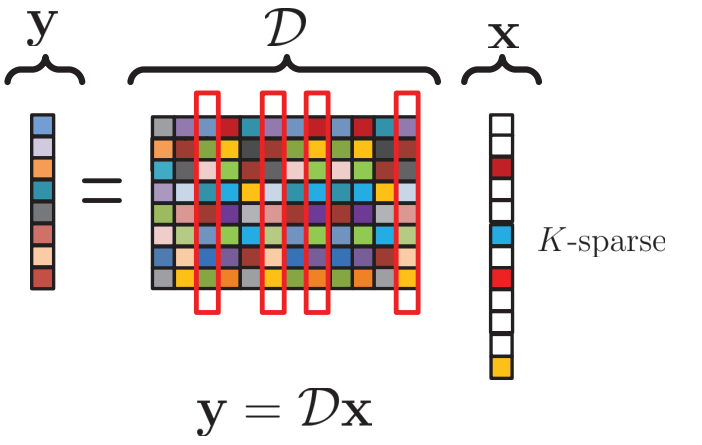


Figure 1 The SMV model aims to minimize a vector sparsity. In the sparse signal $x \in \mathrm{R}^N$, only K entries are non-zero. Thus, the signal is K-sparse. The rectangular areas from the dictionary D are associated with non-zero coefficients from x.

improved performance. Therefore, a variety of dictionary learning methods are proposed, which are grouped into three categories: *clustering-based* [18], *learning with specific structure* [16], and *probabilistic methods* [19]. In this paper, we focus on dictionary learning with specific structure, *i.e.*, optimization of the mutual coherence among columns from the dictionary [16]. As such, we firstly introduce the basic definition of the mutual coherence for a given dictionary.

Definition 1. The mutual coherence $\mu(\mathcal{D})$ of a dictionary $\mathcal{D}$ with N columns is defined as the maximum normalized correlation between the columns of $\mathcal{D}$:

$$\mu(\mathcal{D}) = \max_{1\le i,j\le N, i\neq j} \frac{|\boldsymbol{d}_i^T \boldsymbol{d}_j|}{\|\boldsymbol{d}_i\| \left\|\boldsymbol{d}_j\right\|}, \tag{10}$$

where $\boldsymbol{d}_i$ and $\boldsymbol{d}_j$ is the i and j-th column vector of $\mathcal{D}$, respectively.

In general, a dictionary consisting of more similar columns has a higher $\mu(\mathcal{D})$ value, whereas a dictionary with smaller $\mu(\mathcal{D})$ has a higher probability to contain columns with different structures. Obviously, a dictionary with a variety of columns is more preferred (*i.e.*, small value of $\mu(\mathcal{D})$). Therefore, optimizing the dictionary structure is equivalent to minimize the mutual coherence of $\mathcal{D}$. Empirical results from existing literature have further demonstrated that the performance of the sparse representation is improved by minimizing the dictionary coherence [16–18]. In other words, with the same sparse representation algorithm (such as OMP), the optimized dictionary leads to a much better solution.

4. PROPOSED SPARSE LS-SVM

In this section, we present a sparsity-enhanced algorithm, termed *Adaptive-Kernel based Sparse Learning (AKSL)*, to address two main issues associated with the LS-SVM training, *i.e.*, solution sparsity and kernel parameter optimization. The main difference between AKSL and existing LS-SVM technique is that the proposed method is capable of training the model using less SVs and optimizing the kernel parameter simultaneously.

Overall, the proposed AKSL algorithm consists of two stages: sparse representation and dictionary learning. See Algorithm 1 for the general diagram of the proposed AKSL algorithm. In the sparse representation stage, the goal is to conduct kernel-based training via forming a sparse solution. Additionally, the stage of dictionary learning is characterized by adjusting the kernel parameter while fixing the sparse solution. Toward this end, in Section 4.1, we formulate the LS-SVM training as a sparse representation process. In Section 4.2, we introduce a heuristic approach (based on the dictionary learning) to optimize kernel parameters.

4.1. Sparsity-Enhanced Training

Suppose that we have a training set consisting of N samples $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$, where $\boldsymbol{x}_i$ is the i-th input sample and $y_i \in \{1, -1\}$ is the class label. Our aim is to build a mapping function (denoted as $f(\cdot)$) between the input–outputs samples to form a correct relationship. This mapping function could be a classifier (in classification problems) or a regression function (in approximation problems).

Algorithm 1: The proposed sparsity-enhanced algorithm (AKSL) for the LS-SVM training.

Input : N pairs of training samples $\boldsymbol{x}_i, y_i$ $(i \in [1, N])$, and the number of maximal iterations T;
Output: An optimized kernel size σ^* and the sparse solution $\boldsymbol{\alpha}^*$;

Initialization*: randomly set the kernel size* σ_1;
for $t = 1$ **to** T **do**
  Sparse representation stage: *Calculate the kernel matrix* Q_t *(based on* σ_t*) and then compute the sparse solution* $\boldsymbol{\alpha}_t$.
  Dictionary learning stage: *Update/calculate* σ_{t+1} *via minimizing the column coherence among* Q_t *while fixing* $\boldsymbol{\alpha}_t$.
  Evaluate the kernel model using the validation dataset;
  if Evaluate the kernel model using the validation dataset*;* **then**
    break;
  end
end
Return $\sigma^* = \sigma_t$, *and* $\boldsymbol{\alpha}^* = \boldsymbol{\alpha}_t$.

To simply the process, we skip the bias term b by setting $b = 0$ in Eq. (7), so the mapping function becomes the following form: $f(\boldsymbol{x}_i) = \sum_{j=1}^N \boldsymbol{\alpha}_j \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) + e_i$ $(i = 1, 2, \ldots, N)$. Then the training process aims to find the optimal vector $\boldsymbol{\alpha}$ that satisfies:

$$\boldsymbol{y} = f([\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]) + \boldsymbol{\epsilon} = \mathcal{D}\boldsymbol{\alpha} + \boldsymbol{\epsilon}, \quad i = 1, 2, \ldots, N, \tag{11}$$

where $\boldsymbol{y} = [y_1, \cdots, y_N]$, $\mathcal{D}$ is the kernel matrix with $\mathcal{D}_{ij} = \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $\boldsymbol{\epsilon}$ bounds the amount of additional error. Traditionally, this coefficient vector $\boldsymbol{\alpha}$ can be calculated as $\boldsymbol{\alpha} = (\mathcal{D}^{-1} + \gamma \mathcal{I})\boldsymbol{y}$, where γ is the regularization factor that controls the smoothness of the solution, and $\mathcal{I}$ is the $N \times N$ identity matrix.

Nevertheless, the computation of $\boldsymbol{\alpha}$ usually requires significant memory and computational burden due to the large dimension of the kernel matrix and its inverse operation, in particular with the increasing number of samples (or SVs). To find a compact or sparse model, less number of SVs is preferred, rather than using all of them.

Note that setting a particular element of $\boldsymbol{\alpha}$ to zero is equivalent to omitting the corresponding training sample or SVs. Therefore, the goal of finding a sparse model, within a given tolerance of accuracy, can be equated to minimizing the number of non-zero elements

from the parameter vector $\boldsymbol{\alpha}$. For further discussion, we simply cast the training problem as follows:

$$\min S(\boldsymbol{\alpha}) \quad \text{subject to} \quad \|\boldsymbol{y} - \mathcal{D}\boldsymbol{\alpha}\|_2^2 < \varepsilon, \tag{12}$$

where $S(\boldsymbol{\alpha})$ denotes a sparsity measure, which can be calculated using the l_1 norm of the $\boldsymbol{\alpha}$ vector. Now the sparsity-enhanced LS-SVM training is converted to solving a SMV model, and the training process is then equivalent to finding a sparse representation for the parameter vector $\boldsymbol{\alpha}$. Accordingly, only SVs associated with non-zero parameters are selected; the remaining SVs do not affect the training model (as their weights equal to zero).

Algorithms that applied to solve the SMV model can be accordingly employed for Eq. (12). Herein, we use the OMP algorithm [15]. The reason is two-fold: (i) the OMP algorithm has been proven to be effective for solving the SMV model in numerous case studies; and (ii) more importantly, via OMP we can control the sparsity or number of non-zero elements in the solution $\boldsymbol{\alpha}$. More precisely, this greedy algorithm starts from an empty set. At each iteration, at most one particular element of $\boldsymbol{\alpha}$ will be selected according to the similarity between columns of $\mathcal{D}$ and the residual error. If OMP halts at the K-th iteration, there will be maximally K non-zero elements formed in $\boldsymbol{\alpha}$ while others are set to zero value, *i.e.*, $S(\boldsymbol{\alpha}) \leqslant K$.

4.2. Kernel Parameter Optimization Using Dictionary Learning

During the sparse representation stage, the proposed algorithm generates a compact model by selecting most significant elements from $\boldsymbol{\alpha}$. Again, this selection is equivalent to finding the sparse representation for the vector $\boldsymbol{\alpha}$. Note that now the kernel matrix $\mathcal{D}$ (see Eq. (12)) serves as the dictionary from the SMV model in Eq. (8). To improve the performance of the sparse representation, a dictionary-learning inspired algorithm is accordingly introduced in this paper. Our aim is to optimize $\mathcal{D}$ in a way that a better performance for the sparse representation of $\boldsymbol{\alpha}$ is achievable. Note that the kernel matrix $\mathcal{D}$ is determined by the kernel parameter. More precisely, as for the employed RBF, this simply depends on the kernel size (or σ). As a result, the process of optimizing the $\mathcal{D}$ matrix is further formulated as searching for an optimal value of σ.

Toward this end, we introduce a two-step methodology to systematically conduct the σ-based optimization. First of all, we consider to optimize the kernel size (σ) by minimizing the training error, that is computed as follows (the sparse vector $\boldsymbol{\alpha}$ is fixed):

$$\varepsilon = \|\boldsymbol{y} - \mathcal{D}\boldsymbol{\alpha}\|_2^2 \tag{13}$$

where $\mathcal{D}_{ij} = \mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{2\sigma^2}\right)$ $(\forall i, j \in [1, N])$, given N training samples.

Second, we further like to minimize the mutual coherence of the dictionary (*i.e.*, $\mu(\mathcal{D})$ see Eq. (10)) to promote the accuracy. A simple way to consider $\mu(\mathcal{D})$ is to compute the largest magnitude of the off-diagonal entries of its Gram matrix $\mathcal{G}$; alternatively, the mutual coherence can be calculated as follows:

$$\mu(\mathcal{D}) = \max_{i \neq j} \mathcal{G}_{i,j}, \quad \text{and} \quad \mathcal{G} = \tilde{\mathcal{D}}^T \tilde{\mathcal{D}}, \tag{14}$$

where $\mathcal{G}_{i,j}$ represents the element from the i-th row and the j-th column of $\mathcal{G}$, and $\tilde{\mathcal{D}}$ represents the normalized dictionary of $\mathcal{D}$. Again, the $\mu(\mathcal{D})$ measurement evaluates the worst similarity between any two dictionary columns. Obviously, two closely-related columns may confuse any sparse representation technique; therefore, a smaller $\mu(\mathcal{D})$ is preferred.

However, $\mu(\mathcal{D})$ could fail to measure the "average" behavior of the dictionary as it only focuses on the worst-case standpoint [16]. In this context, an average measurement of coherence is more appealing as it is more likely to lead to a overall-better dictionary with a wider range of columns. As such, we introduce a new measurement of the *average mutual coherence*, that is defined as follows:

Definition 2. For a dictionary $\mathcal{D}$ with N columns, its average mutual coherence $\mu_{avg}(\mathcal{D})$ considers the aggregated similarity among all columns of $\mathcal{D}$:

$$\mu_{avg}(\mathcal{D}) = \frac{\sum_{i,j} \mathcal{G}_{i,j} - \boldsymbol{tr}(\mathcal{G})}{N(N-1)}. \tag{15}$$

Note that the proposed $\mu_{avg}(\mathcal{D})$ coherence is conceptually related to the $\mu(\mathcal{D})$ measurement. The major difference is that $\mu_{avg}(\mathcal{D})$ considers the overall behavior of the dictionary (average similarity), instead of the worst case only (*i.e.*, $\mu(\mathcal{D})$).

To minimize $\mu_{avg}(\mathcal{D})$, let $\widetilde{\mathcal{D}_{:,i}}$ represent the i-th column of $\tilde{\mathcal{D}}$ ($\tilde{\mathcal{D}}$ is the normalized matrix of $\mathcal{D}$), and $\langle \cdot, \cdot \rangle$ denote the dot product of two vectors. Since $\mathcal{G} = \tilde{\mathcal{D}}^T \tilde{\mathcal{D}}$, we know $\mathcal{G}_{i,j} = \langle \widetilde{\mathcal{D}_{:,i}}, \widetilde{\mathcal{D}_{:,j}} \rangle$. Alternatively, we have:

$$\mathcal{G}_{i,j} = \sum_l \frac{\mathcal{D}_{l,i}}{\sqrt{\sum_k \mathcal{D}_{k,i}^2}} \cdot \frac{\mathcal{D}_{l,j}}{\sqrt{\sum_k \mathcal{D}_{k,j}^2}}, \quad \forall k \in [1, 2, \ldots, N]. \tag{16}$$

On the other hand, to compute $\boldsymbol{tr}(\mathcal{G})$, it is straightforward to have:

$$\mathcal{G}_{i,i} = \langle \widetilde{\mathcal{D}_{:,i}}, \widetilde{\mathcal{D}_{:,i}} \rangle = \frac{\langle \mathcal{D}_{:,i}, \mathcal{D}_{:,i} \rangle}{\|\mathcal{D}_{:,i}\|_2^2} = 1, \tag{17}$$

Immediately, we have $\boldsymbol{tr}(\mathcal{G}) = N$ in Eq. (15).

Overall, the σ-oriented optimization is considered by combining Eqs. (13) and (15). In other words, σ will be optimized toward minimizing the training error and average mutual coherence simultaneously. As a result, the proposed optimization problem becomes:

$$\sigma^* = \arg\min_{\sigma} \frac{\Sigma_{i,j} \mathcal{G}_{i,j} - N}{N(N-1)} \qquad s.t. \qquad \|\boldsymbol{y} - \mathcal{D} - \boldsymbol{\alpha}\|_2^2 \leqslant \varepsilon. \tag{18}$$

The Lagrangian function of Eq. (18) can be written as:

$$\xi(\sigma) = \frac{\Sigma_{i,j} \mathcal{G}_{i,j} - N}{N(N-1)} + \lambda \|\boldsymbol{y} - \mathcal{D}\boldsymbol{\alpha}\|_2^2, \tag{19}$$

where λ is the penalty term to balance the dictionary behavior and the training error. In this paper, we employ Stochastic gradient descent (SGD) to solve the proposed optimization problem, so that the updating rule for σ_{t+1} (at the $(t+1)$-th iteration) can be derived as follows:

$$\sigma_{t+1} = \sigma_t + \gamma \frac{\partial(\xi(\sigma_t))}{\partial \sigma_t}, \tag{20}$$

where γ is the learning step. On the other hand, we have:

$$\begin{aligned}\frac{\partial(\xi(\sigma_t))}{\partial\sigma_t} &= \frac{\partial}{\partial\sigma_t}\left[\frac{\sum_{i,j}\mathcal{G}_{i,j}-N}{N(N-1)}+\lambda\left\|\boldsymbol{y}-\mathcal{D}\boldsymbol{\alpha}\right\|_2^2\right]\\ &= \frac{\partial}{\partial\sigma_t}\left[\frac{\sum_{i,j}\mathcal{G}_{i,j}-N}{N(N-1)}\right]+\frac{\partial}{\partial\sigma_t}\left[\lambda\sum_i\left(y_i-\sum_j\mathcal{K}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)\alpha_j\right)^2\right]\\ &= \frac{1}{N(N-1)}\sum_{i,j(i\neq j)}\frac{\partial\mathcal{G}_{i,j}}{\partial\sigma_t}\\ &\quad -\lambda\sum_i\left[2\left(y_i-\sum_j\mathcal{K}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)\alpha_j\right)\left(\sum_j\alpha_j\frac{\partial\mathcal{K}_{i,j}}{\partial\sigma_t}\right)\right].\end{aligned} \tag{21}$$

For simplicity, let $\Delta_1 = \mathcal{D}_{l,i}$, $\Delta_2 = \sqrt{\sum_k \mathcal{D}_{k,i}^2}$, $\Delta_3 = \mathcal{D}_{l,j}$, and $\Delta_4 = \sqrt{\sum_k \mathcal{D}_{k,j}^2}$. Consequently, Eq. (16) can be written as $\mathcal{G}_{i,j} = \sum_l \frac{\Delta_1}{\Delta_2}\cdot\frac{\Delta_3}{\Delta_4}$, and its derivative is computed as:

$$\begin{aligned}\frac{\partial\mathcal{G}_{i,j}}{\partial\sigma} &= \sum_l\frac{\frac{\partial\Delta_1\Delta_3}{\partial\sigma}\Delta_2\Delta_4-\frac{\partial\Delta_2\Delta_4}{\partial\sigma}\Delta_1\Delta_3}{(\Delta_2\Delta_4)^2}\\ &= \sum_l\frac{\left(\frac{\partial\Delta_1}{\partial\sigma}\Delta_3+\frac{\partial\Delta_3}{\partial\sigma}\Delta_1\right)\Delta_2\Delta_4-\left(\frac{\partial\Delta_2}{\partial\sigma}\Delta_4+\frac{\partial\Delta_4}{\partial\sigma}\Delta_2\right)\Delta_1\Delta_3}{(\Delta_2\Delta_4)^2},\end{aligned} \tag{22}$$

where we have

$$\frac{\partial\Delta_2}{\partial\sigma} = \frac{1}{\Delta_2}\sum_k\mathcal{D}_{k,i}\frac{\partial\mathcal{D}_{k,i}}{\partial\sigma},$$

and

$$\frac{\partial\Delta_4}{\partial\sigma} = \frac{1}{\Delta_4}\sum_k\mathcal{D}_{k,j}\frac{\partial\mathcal{D}_{k,j}}{\partial\sigma}.$$

On the other hand, from the definition of RBF kernel function in Eq. (1), it is easy to know

$$\frac{\partial\mathcal{K}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)}{\partial\sigma} = \frac{\partial\mathcal{D}_{i,j}}{\partial\sigma} = \mathcal{K}\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)\left\|\boldsymbol{x}_i-\boldsymbol{x}_j\right\|_2^2\sigma^{-3}. \tag{23}$$

Note that $\frac{\partial(\xi(\sigma_t))}{\partial\sigma_t}$ can be computed based on Eqs. (22) and (23). Additionally, we notice some common elements that can be reused for this computation. That is, Eq. (22) is barely a combination of $\frac{\partial\mathcal{D}_{i,j}}{\partial\sigma}$ and the l_2 norm of the column vector (*i.e.*, $\sqrt{\sum_k \mathcal{D}_{k,i}^2}$); Besides, we also have $\frac{\partial\mathcal{G}_{i,j}}{\partial\sigma} = \frac{\partial\mathcal{G}_{j,i}}{\partial\sigma}$ and $\frac{\partial\mathcal{D}_{i,j}}{\partial\sigma} = \frac{\partial\mathcal{D}_{j,i}}{\partial\sigma}$. Hence, to efficiently compute $\frac{\partial(\xi(\sigma))}{\partial\sigma}$, two tables are maintained to store $\frac{\partial\mathcal{D}_{i,j}}{\partial\sigma}$ ($i\neq j$, and the storing complexity is $\mathcal{O}\left(\frac{N\times(N-1)}{2}\right)$) and the l_2 norm of all columns from $\mathcal{D}$ (the storing complexity is $\mathcal{O}(N)$). As such, once these two tables are built, to compute $\frac{\partial(\xi(\sigma))}{\partial\sigma}$ we can then quickly check and combine necessary elements from these two tables. Meanwhile, these two tables only need calculating once for each iteration, so that the total computation will be very efficient.

5. EXPERIMENTAL RESULTS

This section presents the experimental results and comparisons of the proposed algorithm with conventional training techniques. The employed classification data sets and the evaluation of the training algorithms are presented in Section 5.1. The performance of the proposed algorithm is then evaluated in Section 5.2. The comparison results with conventional training algorithms are presented in Section 5.3.

5.1. Experimental Methods

Several classification problems are chosen from UCI repository [20] for the experimental evaluation (see Table 1). These problems are selected so as to ensure a good coverage of sample sizes, output dimensions (multiple and binary), variable types (discrete and continuous), and attribute numbers.

To measure the performance of the proposed algorithm, we applied the 4-fold cross validation and randomly partitioned one entire data into three independent sets: a training set, a validation set, and a testing set. The size of the training, validation and testing sets in all cases is 50%, 25%, and 25%, respectively. Additionally, training samples will firstly be standardized to zero mean and unit variance.

The solution sparsity or the model structure is measured as follows:

$$k = \frac{K}{N}\times 100\%, \tag{24}$$

where K is the number of selected SVs, and N is the number of training samples. Thus, a larger value for k means that more SVs are selected during training.

Additionally, the proposed training algorithm is terminated if the error on the validation set increases for a few successive iterations (empirically, the value is set as three). This is a common strategy for terminating training in other existing work [21]. The penalty term of λ and the learning step γ is set to 0.01 and 10^{-3}, respectively. The sparse-representation solver is employed from OMP, which is implemented using the scikit-learn package.[1] The required parameter for running OMP is the number of non-zero elements, which is the same as the solution sparsity. Therefore, we are using k (or K) from Eq. (24) to run OMP. Finally, the accuracy performance of the training algorithms is evaluated using the Area Under the Receiver Operating Characteristic Curve (AUC, %).

Table 1 Employed classification datasets from UCI.

Data Set	#.Features	#.Size	Classes
Leaf	16	340	Binary
Online (News Popularity)	60	39797	Binary
Adult	14	32562	Binary
Badges	26	294	Binary
Breastcancer	10	699	Binary
Ionosphere	34	351	Binary
Contraceptive	9	1473	Multiple
Dermatology	34	366	Multiple
Glass	10	214	Multiple
Wine	13	178	Multiple

[1] Available from https://scikit-learn.org.

5.2. Performance Analysis of AKSL

In this section, we first investigate the effect of the solution sparsity on the generalization ability. Second, we test the performance of the proposed method with the optimized kernel size.

5.2.1. Support vectors

The number of SVs is critical to the performance of the AKSL algorithm. For instance, too many SVs might have a negative impact on the generalization ability, not to mention larger computational cost. Therefore, in this section we analyze how the number of SVs impacts the performance of AKSL. The remaining model structure is set to $k = 20\%, 30\%, 50\%, 70\%$, and 100%, respectively. Note that selecting all SVs (*i.e.*, $k = 100\%$) is equivalent to utilize all training inputs (same as the traditional LS-SVM method).

Table 2 shows the performance of the proposed algorithm with respect to the remaining model structure. As can be observed, the proposed method achieves a better classification accuracy on the training sets with increasing number of SVs. The performance on the training sets is 96.43%, 98.43%, 99.39%, 99.83%, and 99.96% for $k = 20\%, 30\%, 50\%, 70\%$, and 100%, respectively. On the other hand, the generalization ability of the trained model is not guaranteed to be improved with more SVs. For instance, the average classification rate on the test set is 76.20%, 77.37%, 79.06%, 78.52%, and 78.31% for $k = 20\%, 30\%, 50\%, 70\%$, and 100%, respectively. One reason is that the model with more SVs could overfit the training data, and thus its accuracy is reduced on the test sets. Overall, the results indicate that selecting fewer SVs leads to better generalization performance.

5.2.2. Kernel size optimization

The adaptive kernel-size technique is considered in the proposed algorithm, which is used to obtain a suitable value for the kernel size (*i.e.*, σ). (Note that without optimizing the kernel size, the proposed algorithm downgrades to the CLS-SVM algorithm [6]. We will compare their performance in Section 5.3.) Toward this end, in our paper σ is optimized to minimize the training error and the average mutual coherence of the employed dictionary. Consequently, in this section, we analyze the performance of the proposed algorithm based on the optimized σ. Meanwhile, we set the percentage of selected SV to $k = 50\%$, as it leads to the best testing accuracy in our previous experiments.

The evolutionary curve related to the average mutual coherence ($\mu_{avg}(\boldsymbol{D})$) is shown in Figure 2. As can be seen and as expected, the results of $\mu_{avg}(\boldsymbol{D})$ decreases with more iterations. The proposed

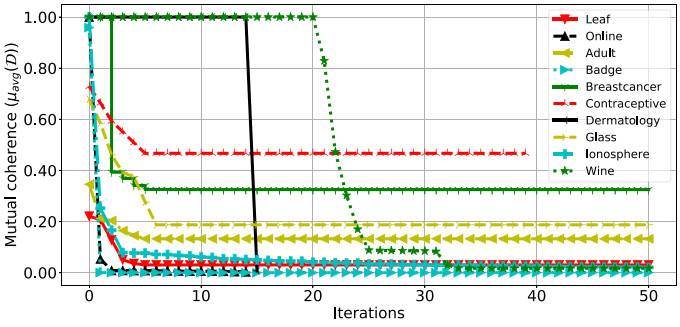


Figure 2 Related average mutual coherence ($\mu_{avg}(\boldsymbol{D})$) as the function of kernel size for employed datasets.

Table 2 Summary of the classification accuracy (%) from the proposed AKSL method. Various numbers of selected support vectors are considered.

DataSets	k	20%	30%	50%	70%	100%
Leaf	Training	96.81%	98.95%	99.89%	100.00%	100.00%
	Test	68.21%	68.57%	71.96%	68.95%	70.36%
Online	Training	92.09%	97.52%	100.00%	100.00%	100.00%
	Test	70.02%	69.46%	67.81%	69.72%	67.96%
Adult	Training	97.67%	98.77%	99.58%	99.83%	100.00%
	Test	81.80%	82.25%	84.94%	82.73%	86.68%
Badge	Training	93.69%	100.00%	100.00%	100.00%	100.00%
	Test	69.87%	73.16%	73.29%	80.92%	78.03%
Breastcancer	Training	100.00%	100.00%	100.00%	100.00%	100.00%
	Test	98.87%	98.85%	98.81%	98.35%	99.28%
Contraceptive	Training	88.06%	91.98%	95.63%	98.44%	99.57%
	Test	80.51%	83.18%	89.82%	79.37%	74.10%
Dermatology	Training	99.95%	99.99%	100.00%	100.00%	100.00%
	Test	96.12%	96.56%	95.16%	93.08%	95.97%
Glass	Training	96.02%	97.29%	98.82%	99.99%	100.00%
	Test	78.33%	79.31%	71.43%	72.22%	77.09%
Ionosphere	Training	99.96%	99.84%	100.00%	100.00%	100.00%
	Test	96.56%	92.40%	97.33%	90.32%	97.39%
Wine	Training	100.00%	100.00%	100.00%	100.00%	100.00%
	Test	91.67%	100.00%	100.00%	100.00%	96.25%
Average	Training	96.43%	98.43%	99.39%	99.83%	99.96%
	Test	76.20%	77.37%	79.06%	78.52%	78.31%

Note. AKSL = adaptive-Kernel based Sparse Learning.

optimization indeed leads to the minimization of the average coherence of the dictionary.

The next experiment explores the effect of the adaptive σ on the training performance. The idea behind such an experiment is to find out, to what extent, how the proposed algorithm benefits from minimizing $\mu_{avg}(\boldsymbol{D})$. The obtained results, in Figure 3, show a consistent improvement in terms of the training performance. Again, as discussed in Section 3, the performance of sparse representation depends heavily on the employed dictionary. When the dictionary is optimized (*i.e.*, a smaller value of $\mu_{avg}(\boldsymbol{D})$), a sparser and more accurate solution is achieved. In the kernel-training context, a better-designed kernel matrix (dictionary) requires less SVs for training, thereby achieving a higher-sparsity solution and better training outcome. The experimental results confirm that the proposed algorithm benefits from the dictionary learning technique via achieving more-accurate classification performance.

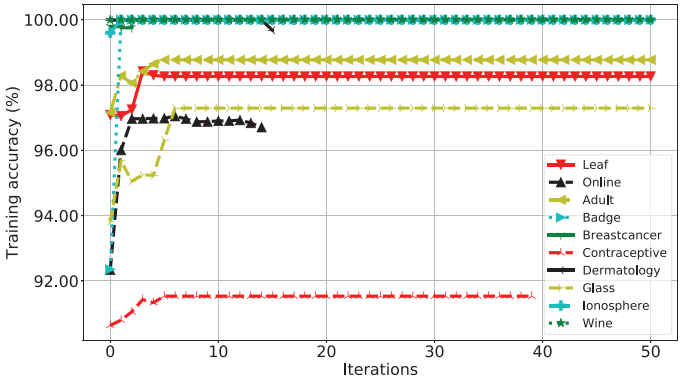


Figure 3 Evolutionary curve of the training accuracy (%) from the proposed algorithm for employed datasets.

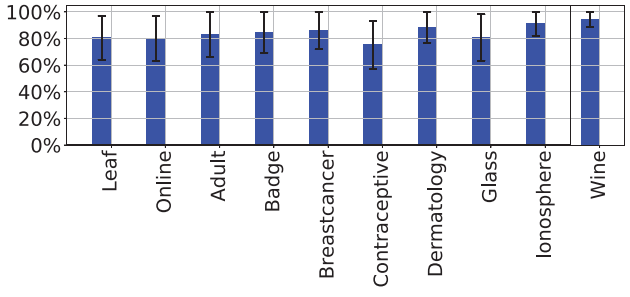
5.3. Comparison with Other Training Algorithms

In this section, the proposed AKSL algorithm is compared with five conventional sparse training algorithms, namely CLS-SVM [6], SSS [10], SLQ [11], CCS [22], and VSS [9]. We have briefly introduced the algorithm of CLS-SVM, SSS, and SLQ in Section 2. For CCS, it has been proposed to enhance the traditional LS-SVM by minimizing the number of SVs. The major contribution is that CCS adopts a Singular Value Decomposition (SVD) to form a measurement matrix $\boldsymbol{\Psi}$, so their optimization problem becomes: $\min S(\boldsymbol{\alpha})$ s.t $\left\|\boldsymbol{\Psi}_y - \boldsymbol{\Psi}\boldsymbol{D}\boldsymbol{\alpha}\right\|_2 < \varepsilon$ (compared to Eq. (12)).

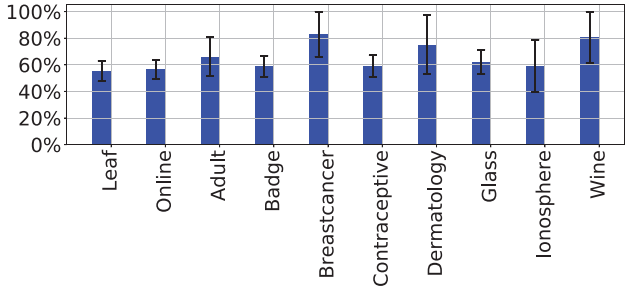
Note that all algorithms of CLS-SVM, SSS, SLQ, and CCS are aiming for reducing the number of SVs and maximizing the solution sparsity. To make a fair comparison, for these sparsity-based algorithms, we apply the same least number of SVs (*i.e.*, set $k = 50\%$). However, existing algorithms need the cross-validation experiment to decide the kernel size. In the following experiment, their σ value is determined using a grid-search method between the range of $[10^{-5}, 10^{3}]$.

On the other hand, the VSS method presented in [9] is to solve a least-mean-square based problem with adaptive kernel-sizes. The major difference of VSS, compared to others, is that VSS does not consider the solution sparsity, but only optimizes the kernel size.

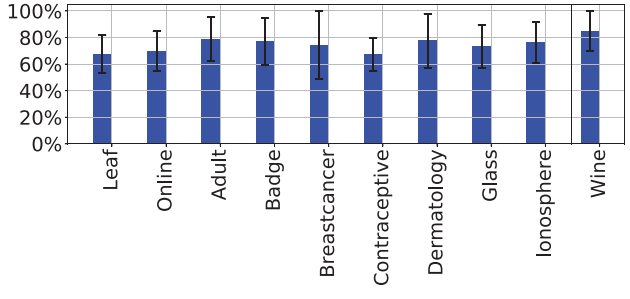
Figures 4 and 5 presents the average training and test accuracy obtained from different methods, respectively, while the accuracy standard deviation is also provided. Compared to conventional training algorithms, the proposed AKSL algorithm achieves a significant improvement in terms of classification accuracy. For instance, using the same number of SVs (k), on average, AKSL achieves the best result of 79.06% on the test sets, which is much better than the accuracy of CLS-SVM (59.70%), CCS (61.62%), and SSS (59.47%). Although the SLQ/VSS method has performed



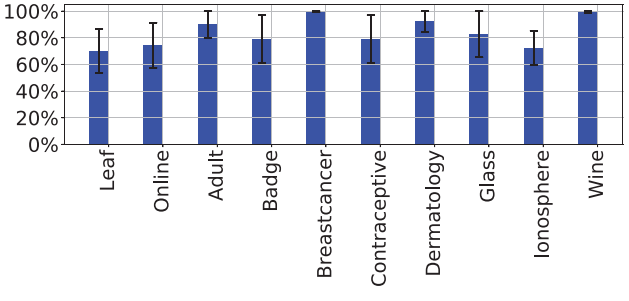
(a) CLS-SVM



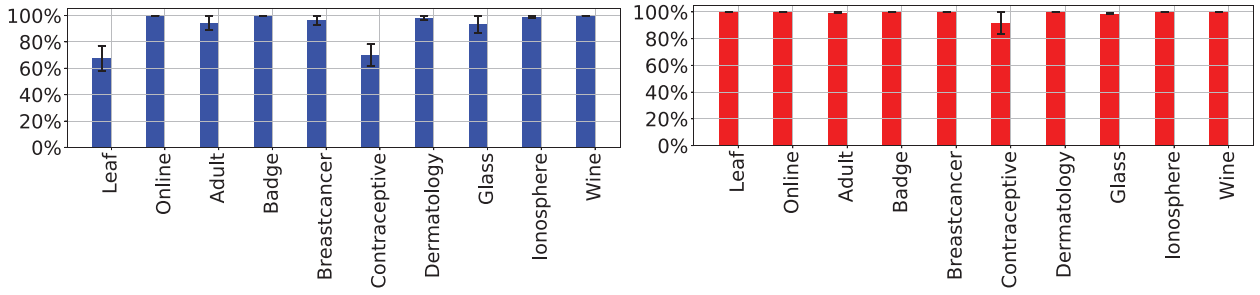
(b) CCS



(c) SSS

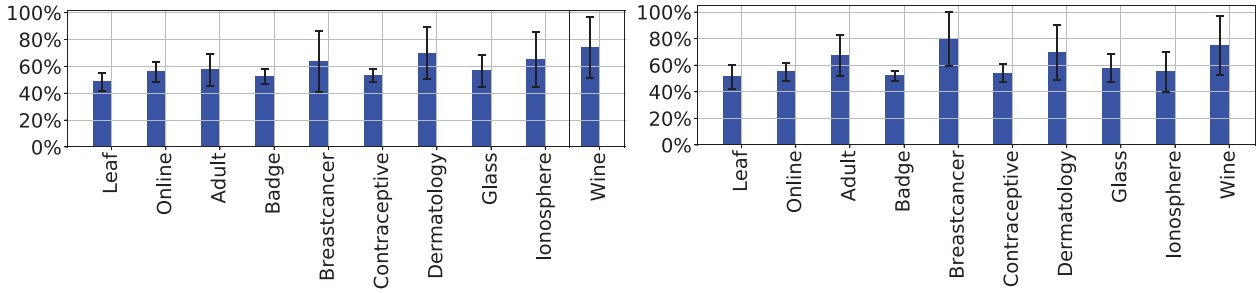


(d) SLQ

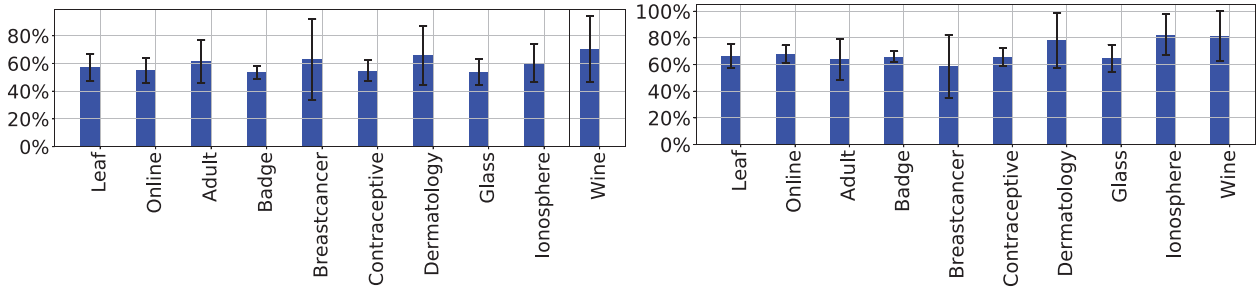


(e) VSS (f) AKSL (proposed)

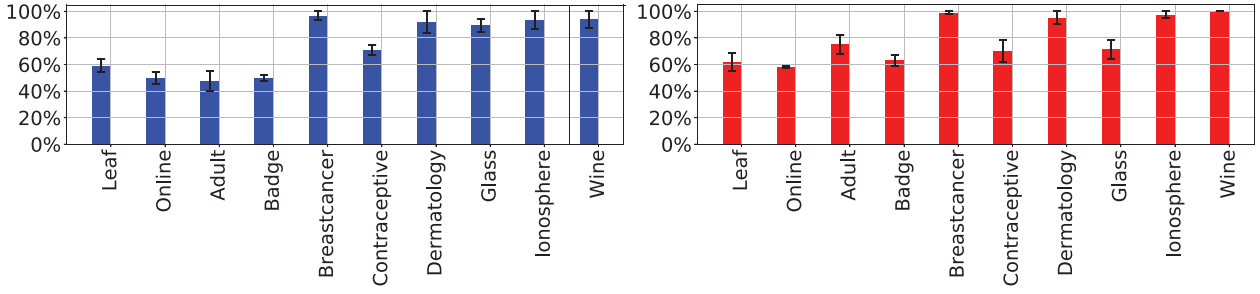
Figure 4 Average training accuracy obtained from different algorithms for the classification tasks, while black lines represent the confidence intervals at the 95% level.



(a) CLS-SVM (b) CCS



(c) SSS (d) SLQ



(e) VSS (f) AKSL (proposed)

Figure 5 Average test accuracy obtained from different algorithms for the classification tasks; again, the error bars are associated with the confidence intervals at the 95% level.

slightly better in some cases, again the average accuracy of the proposed algorithm is superior than that of SLQ (70.55%), and VSS (74.27%) methods. More precisely, our proposed method scores the better generalization (from testing) in eight and night out of all ten datasets, respectively, compared to SLQ and VSS. Overall, it is empirically confirmed that the proposed method obtains a significant improvement compared to existing training methods, in terms of classification accuracy.

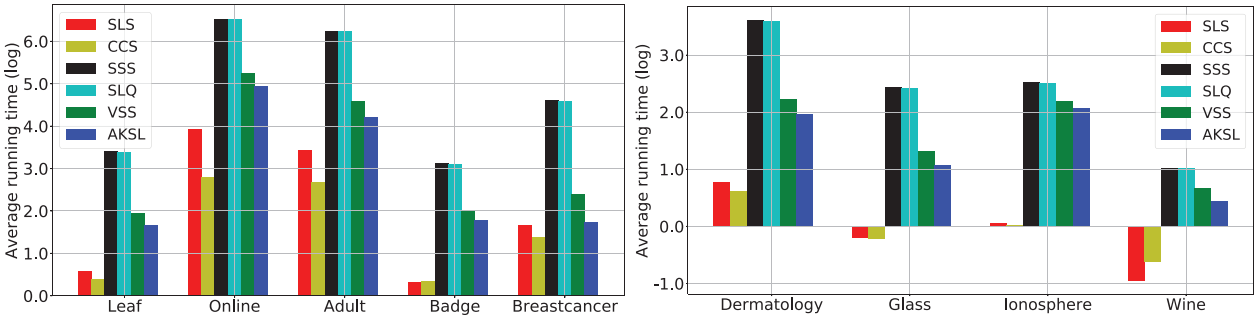


Figure 6 | Average training time (log) obtained from different algorithms for the classification tasks.

We then investigated the computational complexity of all training algorithms. Figure 6 presents the average training time. The proposed algorithm spends 44.68 seconds on average to solve those benchmark problems, which is much better than the training time of SSS (270.70s), SLQ (266.72s), and VSS (62.58s) methods. However, the required time for CLS-SVM (7.59s) and CCS (17.89s) is shorter than the proposed algorithm. The reason could be both CLS-SVM and CCS algorithm do not consider to optimize the kernel size, but only concerns about the sparse solution. Nevertheless, the performance from AKSL is much better than that of CLS-SVM and CCS in terms of the training/testing accuracy, which compensates for its long computational time.

In conclusion, it can be empirically confirmed that the proposed AKSL algorithm outperforms existing state-of-the-art approaches, in terms of solution sparsity (less SVs) and generalization ability. Again, starting from the empty solution, AKSL iteratively selects significant SVs that minimize the training error. This method can be regarded as a forward selection instead of backward elimination, therefore a fast convergence is expected. Furthermore, the proposed approach employs the dictionary-learning technique to optimize the kernel size, instead of traditional cross-validation methods. As a result, the AKSL algorithm achieves an affordable training time and satisfactory classification accuracy.

6. CONCLUSION

In this study, we have proposed an improved training method for LS-SVM, based on sparse signal representation and dictionary learning. The kernel matrix formed using training inputs is regarded as a dictionary in the sparse representation, consequently casting the training process to simply finding a sparse solution for the kernel classifier. The dictionary learning technique is also introduced to further optimize the kernel size during the training process. More precisely, the kernel size is optimized in the way that the average coherence of the kernel matrix is minimized. The main difference between AKSL and existing LS-SVM techniques is that the proposed method is capable of selecting important SVs, training the classification model, and optimizing the kernel size simultaneously. Several benchmark classification problems are chosen for the experimental evaluation. Results demonstrate that the proposed training method leads to competitive performance, in terms of the sparse model and satisfactory classification accuracy.

CONFLICT OF INTERESTS

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

AUTHORS' CONTRIBUTIONS

Chaoyu Yang is responsible for the Conceptualization, Methodology, Software, Validation, Investigation, Visualization, and Writing of the original draft. Jie Yang and Jun Ma are responsible for the Software, Writing–Review and Editing, and Supervision of the original manuscript.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (Grant No. 61873004), and the Humanities and Social Sciences Foundation of Anhui Department of Education, China (Grant No. SK2017A0098).

REFERENCES

[1] A. Kuh, P.D. Wilde, Comments on "Pruning error minimization in least squares support vector machines." IEEE Trans. Neural Netw. 18 (2007), 606–609.
[2] J.A.K. Suykens, L. Lukas, J. Vandewalle, Sparse approximation using least squares support vector machines, IEEE Int. Symp. Circuits Syst. 2 (2000), 757–760.
[3] L. Yuan, C. Lin, W. Zhang, Improved sparse least squares support vector machine classifiers, Neurocomputing. 69 (2006), 1655–1658.
[4] R. Mall, J.A.K. Suykens, Very sparse LSSVM reductions for large-scale data, IEEE Trans. Neural Netw. Learn. Syst. 26 (2015), 1086–1097.
[5] D.A. Silva, J.P. Silva, A.R.R. Neto, Novel approaches using evolutionary computation for sparse least squares support vector machines, Neurocomputing. 168 (2015), 908–916.
[6] J. Yang, J. Ma, A sparsity-based training algorithm for least squares SVM, in 2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Orlando, FL, USA, 2014, pp. 345–350.

[7] Y. Tan, Y.C. Fang, Y. Li, W. Dai, Adaptive kernel size selection for correntropy based metric, in: J. Park, J. Kim (Eds.), Computer Vision – ACCV 2012 Workshops, Springer, Berlin, Heidelberg, 2013, pp. 50–60.
[8] C.L. Bcklin, C. Andersson, M.G. Gustafsson, Self-tuning density estimation based on Bayesian averaging of adaptive kernel density estimations yields state-of-the-art performance, Pattern Recognit. 78 (2018), 133–143.
[9] S.G. Vega, X.J. Zeng, J. Keane, Learning from data streams using kernel least-mean-square with multiple kernel-sizes and adaptive step-size, Neurocomputing. 339 (2019), 105–115.
[10] L.X. Yang, M. Wang, S.Y. Yang, R. Zhang, P.T. Zhang, Sparse spatio-spectral LapSVM with semisupervised kernel propagation for hyperspectral image classification, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 10 (2017), 2046–2054.
[11] Y.H. Shao, C.N. Li, M.Z. Liu, Z. Wang, N.Y. Deng, Sparse Lq-norm least squares support vector machine with feature selection, Pattern Recognit. 78 (2018), 167–181.
[12] M. Yin, Z.Z. Wu, D.M. Shi, J.B. Gao, S.L. Xie, Locally adaptive sparse representation on Riemannian manifolds for robust classification, Neurocomputing. 310 (2018), 69–76.
[13] J. Wang, S. Daming, D.S. Cheng, Y.Q. Zhang, J.B. Gao, LRSR: low-rank-sparse representation for subspace clustering, Neurocomputing. 214 (2016), 1026–1037.
[14] J. Yang, J. Ma, Compressive sensing-enhanced feature selection and its application in travel mode choice prediction, Appl. Soft Comput. 75 (2019), 537–547.
[15] H.F. Zhu, G.H. Yang, W. Chen, Efficient implementations of orthogonal matching pursuit based on inverse cholesky factorization, in 2013 IEEE 78th Vehicular Technology Conference (VTC Fall), Las Vegas, NV, USA, 2013, pp. 1–5.
[16] M. Elad, Optimized projections for compressed sensing, IEEE Trans. Signal Process. 55 (2007), 5695–5702.
[17] C.Y. Lu, H. Li, Z.C. Lin, Optimized projections for compressed sensing via direct mutual coherence minimization, Signal Process. 151 (2018), 45–55.
[18] M.L. Xie, Z.X. Ji, G.Q. Zhang, T. Wang, Q.S. Sun, Mutually exclusive-KSVD: learning a discriminative dictionary for hyperspectral image classification, Neurocomputing. 315 (2018), 177–189.
[19] I. Tosic, P. Frossard, Dictionary learning: what is the right representation for my signal?, IEEE Signal Process. Mag. 28 (2011), 27–38.
[20] D. Dua, C. Graff, UCI Machine Learning Repository, School of Information and Computer Sciences, University of California, Irvine, 2017.
[21] J. Yang, J. Ma, Feed-forward neural network training using sparse representation, Expert Syst. Appl. 116 (2019), 255–264.
[22] L.X. Yang, S.Y. Yang, S.J. Li, R. Zhang, F. Liu, L.C. Jiao, Coupled compressed sensing inspired sparse spatial-spectral LSSVM for hyperspectral image classificatio, Knowl. Based Syst. 79 (2015), 80–89.