Fast Kernel SVM Training via Support Vector Identification

Xue Mao[1], Zhouyu Fu[2], Ou Wu[1], Weiming Hu[1]
[1]CAS Center for Excellence in Brain Science and Intelligence Technology,
NLPR, Institute of Automation, Chinese Academy of Sciences, China
[2]Didi Research, Beijing, China
{xue.mao, wuou, wmhu}@nlpr.ia.ac.cn, fuzhouyu@didichuxing.com

***Abstract*—Training kernel SVM on large datasets suffers from high computational complexity and requires a large amount of memory. However, a desirable property of SVM is that its decision function is solely determined by the support vectors, a subset of training examples with non-vanishing weights. This motivates a novel efficient algorithm for training kernel SVM via support vector identification. The efficient training algorithm involves two steps. In the first step, we randomly sample the training data without replacement several times, each time a small subset of training data is sampled. Then a kernel SVM is trained on each subset, and the resulting kernel SVM models are used to identify the support vectors on the margin. In the second step, an optimization problem is solved to estimate the Lagrange multipliers corresponding to these support vectors. After obtaining the support vectors and Lagrange multipliers, we can approximate the decision function of kernel SVM. Due to the cubic complexity of standard kernel SVM training algorithm, training many kernel SVMs on small subsets of training data is much more efficient than training a single kernel SVM on the whole training data especially for large datasets. Therefore, our algorithm has better scalability than kernel SVM. Besides, training SVMs on each subset can be done independently, and hence our algorithm can be easily parallelized for further speedup. Since our algorithm only identifies the support vectors on the margin, it produces less number of support vectors as compared to that produced by standard kernel SVM. This makes our algorithm more efficient in prediction too. Experimental results show that our method outperforms state-of-the-art methods and achieves performance on par with the kernel SVM albeit with much improved efficiency.**

I. Introduction

The support vector machine (SVM) is widely used in machine learning community. Although linear SVM is extremely efficient [1], [2], [3], [4], it cannot handle nonlinear data with acceptable accuracy. On the other hand, while kernel SVM [5] often produces satisfactory classification results, its high time and space complexities limit its application to large scale datasets. Specifically, standard kernel SVM training has $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ space complexities, where N is the training set size [6]. To address this issue, methods to speed up kernel SVMs have been proposed. Some methods provide a low rank approximation to the kernel matrix, like the Nyström method [7], [8], [9]. However, on very large data sets, the resulting rank of the kernel matrix may still be too high to be handled efficiently. Another approach to scale up kernel methods is by reducing the number of support vectors [10]. As shown in the experimental section, this approach often trades off accuracy for lower complexity.

An advantage of the SVM is that the location of the decision boundary is only determined by the support vectors, and hence we only need to identify the support vectors and ignore other non-support vectors. In this paper, we propose a fast kernel SVM training algorithm via Support Vector Identification (SVI). It generally involves two major steps: identifying the support vectors on the margin and then solving the Lagrange multipliers corresponding to these support vectors. In the first step, we randomly sample the training data without replacement several times, each time only a small subset of training data is sampled. Then a kernel SVM is trained on each of these subsets, and the resulting kernel SVM models are combined to identify the support vectors on the margin. It should be noted that our algorithm only identifies the support vectors on the margin. This is because that the decision boundary is largely defined by the location of these support vectors, and it is only weakly influenced by the support vectors elsewhere. Note also that we sample the data without replacement, which means that the kernel SVMs trained on each of these subsets are different and complementary to each other. An ensemble of these diverse SVMs can help us identify the support vectors more accurately. In the second step, an optimization problem is solved to estimate the Lagrange multipliers corresponding to the support vectors. After obtaining these support vectors and Lagrange multipliers, we can approximate the decision function of kernel SVM and make predictions for new data points.

Two factors may account for the improvement in training speed brought by our method. First, we train kernel SVM on a small subset of training data for several times in the first step, which is much more efficient than training a single kernel SVM on the whole training set and leads to substantial reduction in the training cost. Second, a quadratic programming (QP) problem needs to be solved to obtain the Lagrange multipliers in the second step. The number of variables involved is the number of support vectors, while in the QP problem of standard kernel SVM training the number of variables is the number of all the training data. For large datasets, the number of the training data is usually orders of magnitude larger than that of support vectors. Hence the proposed algorithm is

much more efficient than kernel SVM during training. Besides, training SVMs on each small subsample can be done not only efficiently but also independently, and hence our algorithm can be easily parallelized for further speedup. Since only the support vectors on the margin are used for prediction, our algorithm is also more efficient than kernel SVM during prediction. We have evaluated our SVI algorithm on both synthetic and real data sets. Experimental results demonstrate that SVI outperforms other competitive methods. It is also comparable to kernel SVM in terms of classification accuracy, while it is much more efficient than kernel SVM both in training and testing.

The rest of the paper is outlined as follows: Section II will introduce some related work about fast solvers for the kernel SVM. The proposed SVI model will be described in detail in Section III followed by some experimental results in Section IV. The conclusions are given in Section V.

II. Related Work

There are quite a lot of existing studies focusing on speeding up the kernel SVM from different perspectives.

Some approaches employ the Nyström method [7], [8], [9], [11] to provide a low rank approximation to the kernel matrix by sampling a small subset of columns. However, on very large data sets, the resulting rank of the kernel matrix may still be too high to be handled efficiently. Besides, as a generic technique for matrix factorization, the Nyström method is not specifically designed for the classification task and does not take into account label information in the factorization process. Hence discriminative information may be lost with the Nyström approximation, which further leads to sub-optimal solution to the classifier model.

There is previous work that focuses on obtaining an approximate kernel SVM solution by reducing the number of support vectors [12], [13], [14]. SpSVM [10] is one of the representative methods. It greedily finds a set of kernel basis functions (support vectors) to approximate the SVM primal cost function. As shown in the experimental section, these methods often trade off accuracy for lower complexity.

Locally linear classifiers have been also proposed, which learn multiple linear classifiers to approximate the decision boundary of kernel SVM. The first idea which has been widely explored is using a divide-and-conquer strategy [15] involving two steps: partitioning the data into clusters and training a linear classifier for each cluster. CSVM [16] falls into this category. CSVM adopts K-means to partition the data into clusters and then trains a linear SVM for each cluster. Meanwhile, CSVM requires the weight vector of each linear SVM to align with a global weight vector, which can be treated as a type of regularization. The second way is to employ a lazy learning strategy: given a testing sample, a classifier is trained in a subregion of the input space near the sample and then used to classify the sample. SVM-KNN [17] and the adaptive SVM nearest neighbor classifier [18] belong to this category. Since the learning process is postponed until the testing phase, these methods are inefficient during testing. There also exist other locally linear classifiers based on local coding schemes [19], [20], [21], such as LLSVM [22]. LLSVM approximates a nonlinear classifier by a set of linear classifiers associated with each anchor point in a dictionary. It is time-consuming due to nearest neighbor search and the local coordinate coding.

Another work in this field is LDKL [23]. It generalizes localized multiple kernel learning so as to learn a tree-based primal feature embedding that encodes non-linearities, and then combines this feature embedding with the SVM classifier. LDKL is a fully supervised technique where the trees are built by utilizing the label information.

In the experimental section, we will compare our algorithm with some of the above-mentioned methods.

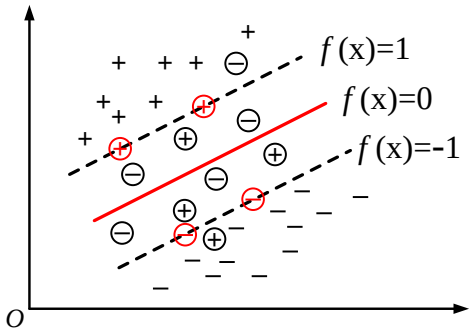


Fig. 1. Illustration of the support vectors and the margin. Data points with circles around them are support vectors. The four red circled points are the support vectors on the margin.

III. SVI Model

Kernel SVM can be prohibitively expensive when dealing with large datasets, which limits its application to many real problems. However, the decision function of kernel SVM is determined only by the support vectors. Specifically, the decision function can be formulated as $f(\mathbf{x}) = \sum_{i=1}^{s} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b$, where $\{(\mathbf{x}_i, y_i)\}_{i=1}^{s}$ are support vectors and $\{\alpha_i\}_{i=1}^{s}$ are the positive Lagrange multipliers corresponding to these support vectors. b is the bias parameter. Fig. 1 gives an illustration of the support vectors and the margin. Data points with circles around them are support vectors, which can be divided into three categories: the support vectors on the margin, inside the margin and outside the margin. The support vectors on the margin (illustrated by the four red circled points in Fig. 1) are the points satisfying $y_i f(\mathbf{x}_i) = 1$, where y_i is the true label of $\mathbf{x}_i$ and $f(\mathbf{x}_i)$ is its decision value. The decision boundary is largely defined by the location of these support vectors. So we just identify these support vectors. Then we solve an optimization problem to obtain the corresponding Lagrange multipliers and the bias parameter. After that, the decision function $f(\mathbf{x})$ is obtained. In the following two subsections, we will introduce how to identify the support vectors and estimate the Lagrange multipliers respectively.

A. Identifying the Support Vectors on the Margin

Training kernel SVM on the entire training data is expensive. This is because standard SVM training has $\mathcal{O}(N^3)$ time complexity, where N is the training set size. We randomly sample the training data without replacement for M times. In each time, only a small subset of training data is sampled, resulting in M non-overlapped subsets, each containing n samples. Then a kernel SVM is trained on each of these subsets. Thus the time complexity of training these M SVMs is $M\mathcal{O}(n^3)$, which is much less than the training complexity $\mathcal{O}(N^3)$ on the entire training data, particularly when the whole training set size N is very large. Then the obtained M SVM models are used to predict the decision values of all the training data. Namely, for each training sample $\mathbf{x}_i$, we use the trained M SVM models to predict its decision value. Then by averaging these M decision values, we get the averaged decision value for each training sample. If the product of the averaged decision value and the true label y_i is approximately equal to 1, the sample $\mathbf{x}_i$ is identified as the support vector on the margin. This can be formulated as:

$$y_i\big(\frac{1}{M}\sum_{m=1}^{M} f_m(\mathbf{x}_i)\big) \approx 1 \tag{1}$$

The reason why we only identify the support vectors on the margin is that the decision hyperplane is largely defined by the location of these support vectors, and it is only weakly influenced by the support vectors elsewhere. It should be also noted that we sample the training data without replacement, resulting in M non-overlapped subsets. Thus the obtained M SVM models are different and complementary to each other. In Equation (1), it can be regarded as ensembling M different classifiers to predict the decision value of the sample $\mathbf{x}_i$. Ensemble diversity, that is, the difference among the individual classifiers, is crucial to ensemble performance [24]. By combining M different SVM models to predict the decision value, we can identify the support vectors on the margin more accurately.

B. Estimating the Lagrange Multipliers

After identifying the support vectors $\{(\mathbf{x}_i, y_i)\}_{i=1}^{s}$ on the margin, we solve the following optimization problem to estimate the Lagrange multipliers:

$$\min_f \frac{1}{2}\|f\|_{\mathcal{H}}^2 + C\sum_{i=1}^{s}(y_i - f(\mathbf{x}_i))^2 \tag{2}$$

$$f(\mathbf{x}_i) = \sum_{j=1}^{s}\beta_j k(\mathbf{x}_j, \mathbf{x}_i) + b \tag{3}$$

where the first term in Equation (2) is the regularization term and the second term minimizes the approximation error to enforce that $f(\mathbf{x}_i) = y_i$. Note here $\beta_j = y_j\alpha_j$, and we will drop the constraints on β to simplify the problem. $\|f\|_{\mathcal{H}}^2$ denotes the function norm in the reproducing kernel Hilbert space (RKHS). Given the above form of f in Equation (3), we have the following relation:

$$\|f\|_{\mathcal{H}}^2 = \sum_{i,j=1}^{s}\beta_i\beta_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{4}$$

Pluging Equation (4) into Equation (2), we can reformulate the optimization problem in the following vectorized form:

$$\min_{\beta,b} Q(\beta, b) = \frac{1}{2}\beta^T\mathbf{K}\beta + C\|\mathbf{y} - \mathbf{K}\beta - \mathbf{1}_s b\|^2 \tag{5}$$

where $\mathbf{K} = [k(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^{s}$ is an $s \times s$ Gram matrix of kernel values calculated between support vectors and $\mathbf{1}_s$ is an s-dimensional vector of ones. This is a quadratic programming (QP) problem. The number of variables that need to be optimized is the number of support vectors (except for the bias b), while in the QP problem of standard kernel SVM training the number of variables is the number of all the training data. For large datasets, the number of the training data is usually much larger than that of support vectors. Hence our algorithm achieves much higher efficiency than kernel SVM in the training phase.

We now solve the minimization problem in Equation (5). Taking the partial derivative of Q with respect to b and setting the derivative to 0, we have:

$$\frac{\partial Q}{\partial b} = 2C\mathbf{1}_s^T(\mathbf{K}\beta + \mathbf{1}_s b - \mathbf{y}) = 0 \tag{6}$$

$$\Rightarrow b = \frac{1}{s}\mathbf{1}_s^T(\mathbf{y} - \mathbf{K}\beta) \tag{7}$$

Let $\mathbf{k}_i$ denote the ith row vector of $\mathbf{K}$ and $\epsilon_i \triangleq y_i - \mathbf{k}_i\beta$ denote the error term for the ith support vector. Then the bias term b is basically the average error ϵ over all support vectors.

Taking the partial derivative of Q with respect to β and setting the derivative to 0, we have:

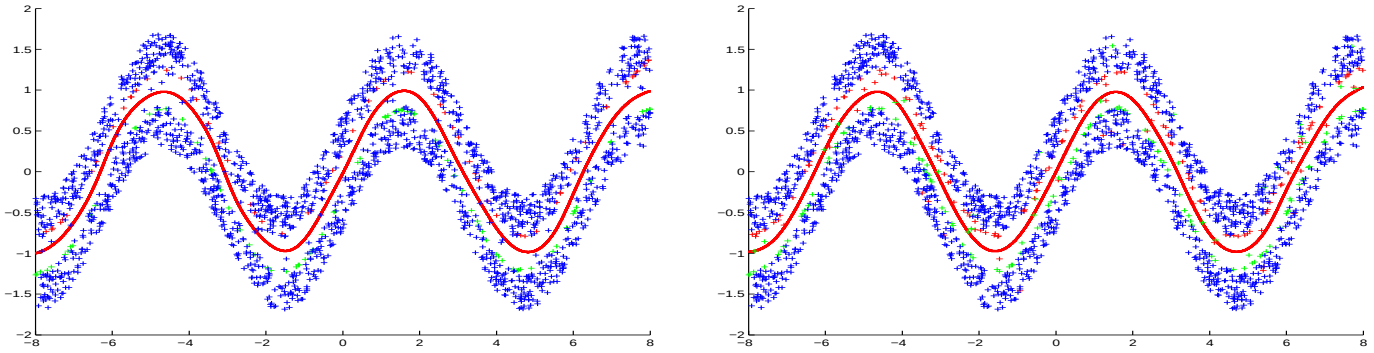
$$\frac{\partial Q}{\partial \beta} = \mathbf{K}\beta + 2C\mathbf{K}^T(\mathbf{K}\beta + \mathbf{1}_s b - \mathbf{y}) \tag{8}$$

$$= \mathbf{K}\beta + 2C\mathbf{K}^T\big(\mathbf{K}\beta + \frac{1}{s}\mathbf{1}_s\mathbf{1}_s^T(\mathbf{y} - \mathbf{K}\beta) - \mathbf{y}\big) = 0$$

$$\Rightarrow \beta = (\frac{1}{2C}\mathbf{I} + \overline{\mathbf{K}})^{-1}\bar{\mathbf{y}} \tag{9}$$

where $\mathbf{I}$ is an $s \times s$ identity matrix. Let $\Pi_1 \triangleq \mathbf{I} - \frac{1}{s}\mathbf{1}_s\mathbf{1}_s^T$ denote a projection matrix, and then $\overline{\mathbf{K}} \triangleq \Pi_1\mathbf{K}$, $\bar{\mathbf{y}} \triangleq \Pi_1\mathbf{y}$. When applying Π_1 to the matrix on the right-hand-side, the average row vector is subtracted from each row of the matrix. After obtaining β, the Lagrange multipliers $\{\alpha_i\}_{i=1}^{s}$ can be obtained by $\alpha_i = y_i\beta_i$.

A sketch of our SVI algorithm is presented in Algorithm 1. It can be seen that it involves two major steps: identifying the support vectors $\{(\mathbf{x}_i, y_i)\}_{i=1}^{s}$ on the margin and then solving the corresponding Lagrange multipliers $\{\alpha_i\}_{i=1}^{s}$ and the bias parameter b. In the testing stage, prediction is made by the sign of the decision function $f(\mathbf{x}) = \sum_{i=1}^{s}\alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b$. Obviously, the prediction cost scales linearly with the number of support vectors. Since we only identify the support vectors on the margin, our algorithm is more efficient than kernel SVM



(a) SVI (b) kernel SVM

Fig. 2. Learned classifiers on the synthetic Sine dataset. The red points and the green points are the support vectors of the positive class and the negative class respectively.

Algorithm 1 SVI

Input: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N \subset \mathbb{R}^d \times \{-1, 1\}$, the number of sampling times M and the sampling rate θ

Output: The support vectors $\{(\mathbf{x}_i, y_i)\}_{i=1}^s$, the corresponding Lagrange multipliers $\{\alpha_i\}_{i=1}^s$ and the bias parameter b.

1: Randomly sample the training data with the sampling rate θ without replacement for M times, resulting in M subsets.
2: Train a kernel SVM on each of these subsets.
3: Combine the M SVMs to identify the support vectors $\{(\mathbf{x}_i, y_i)\}_{i=1}^s$ on the margin via Equation (1).
4: Estimate the parameters β using Equation (9) and then obtain the Lagrange multipliers $\{\alpha_i\}_{i=1}^s$ by $\alpha_i = y_i\beta_i$.
5: Calculate the bias parameter b via Equation (7).

during testing at the expense of only a moderate sacrifice in classification accuracy.

IV. Experiments

A. Synthetic Dataset

The synthetic dataset contains points randomly sampled from two Sine signals, as shown in Fig. 2. Each signal contains 1000 points and forms one class: the upper signal being the positive class and the lower signal being the negative class. We then apply our SVI model and RBF kernel SVM to this dataset. The decision boundaries produced by these two methods are displayed by red curves in Figures 2 (a) and (b) respectively. It can be seen that the decision boundary produced by SVI is quite similar to that of kernel SVM. The red points and the green points are the support vectors of the positive class and the negative class respectively. Obviously, SVI only identifies the support vectors that almost lie on the margin while kernel SVM also identifies the support vectors lying inside or outside the margin. Therefore, SVI only uses a small fraction of support vectors to produce a similar decision boundary to kernel SVM. Since the prediction complexity is linear in the number of support vectors, SVI is more efficient than kernel SVM in prediction. Please enlarge and view these two figures on the screen for better comparison.

B. Real Datasets

We use eight benchmark datasets: Banana, IJCNN, SKIN, Magic04, CIFAR, USPS, MNIST and LETTER. The Banana, USPS and MNIST datasets are used in [25] [26] [27]. The IJCNN dataset is obtained from the LibSVM website [28]. The preprocessed binary CIFAR dataset is taken from [23]. The others are available at the UCI repository [29]. The first five datasets are used for binary classification tasks, and the other three are multi-class datasets. All the datasets have been divided into training and testing sets except the Banana, SKIN, and Magic04 datasets. For Banana and Magic04, we randomly selected two thirds of examples for training and the rest for testing. For SKIN, we used half for training and the rest for testing. All the datasets are normalized to have zero mean and unit variance in each dimension. Table I gives a brief summary of these datasets.

TABLE I
SUMMARY OF THE REAL DATASETS IN OUR EXPERIMENTS

Datasets	# training	# test	# features	# classes
Banana	3,533	1,767	2	2
IJCNN	49,990	91,701	22	2
SKIN	122,529	122,528	3	2
Magic04	12,680	6,340	10	2
CIFAR	50,000	10,000	400	2
USPS	7,291	2,007	256	10
MNIST	60,000	10,000	784	10
LETTER	16,000	4,000	16	26

We compare our SVI model with six previously-mentioned methods: Linear SVM, Kernel SVM, SpSVM, LLSVM, CSVM and LDKL. We use LibLinear [2] and LibSVM [28] for training linear SVM and Gaussian kernel SVM respectively. The regularization parameter C is tuned by 5-fold cross validation on the training set for both linear and kernel SVMs. Cross validation is also used to tune the kernel width of the Gaussian kernel for kernel SVM. For the proposed method SVI, the number of sampling times M is set to five and the sampling rate θ is set to 0.1. The parameters of all the other methods are set as in their original papers, with most parameters set by cross validation. For those methods

TABLE II
COMPARISON OF DIFFERENT CLASSIFIERS IN TERMS OF CLASSIFICATION ACCURACY (%)

Datasets	Banana	IJCNN	SKIN	Magic04	CIFAR	USPS	MNIST	LETTER
Linear SVM	55.29	92.25	93.34	79.30	69.09	91.87	91.91	57.52
Kernel SVM	91.05	98.52	98.98	86.70	81.62	94.86	97.84	97.57
SpSVM	89.86±0.24	96.59±0.43	96.14±0.55	84.87±0.36	74.25±0.63	92.33±0.47	96.10±0.32	94.37±0.19
LLSVM	89.71±0.09	96.86±0.25	95.24±0.46	85.19±0.28	72.60±0.29	93.67±0.21	96.50±0.17	95.03±0.32
CSVM	89.64±0.17	97.25±0.48	97.41±0.29	85.01±0.51	73.37±0.21	92.85±0.18	95.75±0.24	94.58±0.13
LDKL	90.15±0.45	98.31±0.26	98.08±0.38	86.22±0.23	76.04±0.48	93.83±0.34	97.24±0.31	97.15±0.37
SVI	90.62±0.37	98.36±0.32	98.25±0.46	86.51±0.16	76.29±0.27	93.53±0.39	97.61±0.28	97.24±0.30

TABLE III
COMPARISON OF DIFFERENT CLASSIFIERS IN TERMS OF TRAINING TIME (IN SECONDS)

Methods	Banana	IJCNN	SKIN	Magic04	CIFAR	USPS	MNIST	LETTER
Linear SVM	0.0079	2.5636	0.1390	0.0364	2.4807	3.1580	99.6904	0.4587
Kernel SVM	1.54	68.94	7.71	20.62	16713.40	18.56	695.84	24.35
SpSVM	1.368±0.056	29.56±0.188	2.842±0.060	12.61±0.390	453.9±3.856	13.11±0.077	395.4±4.681	10.60±0.379
LLSVM	7.641±0.329	24.92±0.145	21.33±0.052	9.318±0.299	165.0±1.801	24.33±0.836	2429±10.17	122.8±1.440
CSVM	0.693±0.054	18.80±0.889	9.344±0.035	4.274±0.044	416.8±6.508	38.62±1.175	826.3±5.572	6.979±0.015
LDKL	0.892±0.003	15.11±0.395	3.978±0.021	9.539±0.256	272.1±10.29	16.07±0.153	476.1±3.994	10.42±0.186
SVI	0.086±0.015	9.074±0.202	0.954±0.079	4.703±0.019	256.9±5.003	5.832±0.297	153.6±2.889	3.647±0.026

TABLE IV
COMPARISON OF DIFFERENT CLASSIFIERS IN TERMS OF TESTING TIME (IN SECONDS)

Methods	Banana	IJCNN	SKIN	Magic04	CIFAR	USPS	MNIST	LETTER
Linear SVM	0.0029	0.1475	0.0212	0.0043	0.1631	0.0315	0.3339	0.0076
Kernel SVM	0.74	357.06	68.58	10.82	594.08	43.17	867.59	13.49
SpSVM	0.183±0.015	39.05±0.127	5.882±0.293	1.558±0.026	69.01±1.484	9.563±0.194	77.32±1.093	4.566±0.027
LLSVM	0.035±0.009	0.851±0.023	0.647±0.048	0.090±0.020	2.909±0.142	0.482±0.016	6.415±0.090	0.117±0.011
CSVM	0.069±0.017	4.327±0.295	1.581±0.033	0.402±0.041	21.77±0.098	3.658±0.018	60.81±0.551	1.641±0.013
LDKL	0.010±0.004	0.391±0.032	0.275±0.027	0.026±0.004	0.900±0.029	0.098±0.008	1.731±0.080	0.082±0.004
SVI	0.064±0.009	4.452±0.151	1.759±0.054	0.574±0.003	32.11±0.160	2.415±0.031	20.32±0.775	1.053±0.009

involving K-means clustering or other random factors, we take the average results and the standard deviation over 10 random repetitions. The comparison results in terms of classification accuracy are presented in Table II. The results with respect to training time and testing time are given in Tables III and IV respectively.

Unsurprisingly, linear SVM does not perform well on all the datasets as it can not handle nonlinear data. Kernel SVM achieves the best performance overall. Nevertheless, kernel SVM is expensive especially for large datasets as shown in Table III and Table IV. The proposed SVI achieves comparable performance to kernel SVM, but it is much more efficient in both training and prediction. This is because that during training SVI only needs to train kernel SVM on a small subset of training data for several times, which is much more efficient than training kernel SVM on the whole training data. Since we only identify the support vectors on the margin, the support vectors produced are less than that produced by standard kernel SVM, which makes our algorithm more efficient than kernel SVM during prediction. SpSVM is a kind of approximate solver for the kernel SVM by reducing the number of basis functions. It starts with an empty set of basis functions and greedily chooses new basis functions to improve the primal objective function. To achieve a satisfactory classification performance, it requires quite a lot of basis functions. Thus it needs to make a tradeoff between accuracy and computation time. Even though LLSVM performs well on some datasets, it is slow due to the nearest neighbor search and the local coordinate coding. LLSVM is sometimes slower than kernel SVM [16]. CSVM groups the data into several clusters using K-means and then trains a linear SVM in each cluster. Meanwhile, CSVM requires the weight vector of each linear SVM to align with a global reference weight vector, which may not hold in some applications. Additionally, its

performance is directly determined by the initial K-means clustering results. If the initial K-means clustering is poor, the performance of CSVM will likewise be poor. Therefore, CSVM is not robust. LDKL achieves quite good performance overall, but SVI still gains a margin of advantage over LDKL on seven out of eight datasets. The reason lies in the fact that LDKL learns a tree-based primal feature embedding to partition the feature space, which may lead to non-smooth partition over the feature space and abrupt change across region boundaries. Moreover, LDKL is intended to reduce prediction costs, and its training speed is not fast enough [30]. In general, the proposed algorithm SVI achieves a better balance between accuracy and time complexity.

V. Conclusion

In this paper, we have proposed the SVI model, which identifies the support vectors on the margin and estimates the Lagrange multipliers. Then the decision function of kernel SVM is approximated by these support vectors and the corresponding Lagrange multipliers. Since SVI only needs to train kernel SVM on a small subset of training data for several times, it is much more efficient during training. Furthermore, SVM on each subset can be trained separately, which makes our algorithm easily parallelized for further speedup. SVI is also more efficient than kernel SVM during prediction due to using only the support vectors on the margin for prediction. Experimental results on benchmark datasets demonstrate that SVI outperforms state-of-the-art methods. It also achieves much higher efficiency than kernel SVM with comparable classification performance.

Acknowledgement

This work is partly supported by the 973 basic research program of China (Grant No. 2014CB349303), the Natural Science Foundation of China (Grant No. 61472421), the Project Supported by CAS Center for Excellence in Brain Science and Intelligence Technology, and the Project Supported by Guangdong Natural Science Foundation (Grant No. S2012020011081).

References

[1] T. Joachims, "Training linear SVMs in linear time," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '06, 2006, pp. 217–226.

[2] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *The Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[3] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear SVM," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08, 2008, pp. 408–415.

[4] E. Hazan, T. Koren, and N. Srebro, "Beating SGD: Learning SVMs in sublinear time," in *Advances in Neural Information Processing Systems*, ser. NIPS '11, 2011, pp. 1233–1241.

[5] B. Schölkopf and A. J. Smola, *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[6] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core vector machines: Fast SVM training on very large data sets," in *Journal of Machine Learning Research*, 2005, pp. 363–392.

[7] M. Li, J. T. Kwok, and B.-L. Lu, "Making large-scale Nyström approximation possible," in *Proceedings of the 27th International Conference on Machine Learning*, ser. ICML '10, 2010, pp. 631C–638.

[8] C. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems*, ser. NIPS '01, 2001, pp. 682–688.

[9] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, "Nyström method vs random fourier features: A theoretical and empirical comparison," in *Advances in Neural Information Processing Systems*, ser. NIPS '12, 2012, pp. 476–484.

[10] S. S. Keerthi, O. Chapelle, and D. DeCoste, "Building support vector machines with reduced classifier complexity," *The Journal of Machine Learning Research*, vol. 7, pp. 1493–1515, 2006.

[11] A. Vedaldi and A. Zisserman, "Sparse kernel approximations for efficient classification and detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '12, 2012, pp. 2320–2327.

[12] M. Cossalter, R. Yan, and L. Zheng, "Adaptive kernel approximation for large-scale non-linear SVM prediction," in *Proceedings of the 28th International Conference on Machine Learning*, ser. ICML '11, 2011, pp. 409–416.

[13] T. Joachims and C.-N. J. Yu, "Sparse kernel SVMs via cutting-plane training," *Machine Learning*, vol. 76, no. 2-3, pp. 179–193, 2009.

[14] I. W. Tsang, A. Kocsor, and J. T. Kwok, "Simpler core vector machines with enclosing balls," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07, 2007, pp. 911–918.

[15] J. Zhu, N. Chen, and E. P. Xing, "Infinite SVM: a Dirichlet process mixture of large-margin kernel machines," in *Proceedings of the 28th International Conference on Machine Learning*, ser. ICML '11, 2011, pp. 617–624.

[16] Q. Gu and J. Han, "Clustered support vector machines," in *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, ser. AISTATS '13, 2013, pp. 307–315.

[17] H. Zhang, A. C. Berg, M. Maire, and J. Malik, "SVM-KNN: Discriminative nearest neighbor classification for visual category recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '06, 2006, pp. 2126–2136.

[18] E. Blanzieri and F. Melgani, "An adaptive SVM nearest neighbor classifier for remotely sensed imagery," in *IEEE International Conference on Geoscience and Remote Sensing Symposium*, 2006, pp. 3931–3934.

[19] J. C. van Gemert, J.-M. Geusebroek, C. J. Veenman, and A. W. Smeulders, "Kernel codebooks for scene categorization," in *European Conference on Computer Vision*, ser. ECCV '08. Springer, 2008, pp. 696–709.

[20] K. Yu, T. Zhang, and Y. Gong, "Nonlinear learning using local coordinate coding," in *Advances in Neural Information Processing Systems*, ser. NIPS '09, 2009, pp. 2223–2231.

[21] L. Liu, L. Wang, and X. Liu, "In defense of soft-assignment coding," in *IEEE International Conference on Computer Vision*, ser. ICCV '11, 2011, pp. 2486–2493.

[22] L. Ladicky and P. Torr, "Locally linear support vector machines," in *Proceedings of the 28th International Conference on Machine Learning*, ser. ICML '11, June 2011, pp. 985–992.

[23] C. Jose, P. Goyal, P. Aggrwal, and M. Varma, "Local deep kernel learning for efficient non-linear SVM prediction," in *Proceedings of the 30th International Conference on Machine Learning*, ser. ICML '13, 2013, pp. 486–494.

[24] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. CRC Press, 2012.

[25] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for adaboost," *Machine Learning*, vol. 42, no. 3, pp. 287–320, 2001.

[26] J. J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, 1994.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[29] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[30] C.-J. Hsieh, S. Si, and I. S. Dhillon, "A divide-and-conquer solver for kernel support vector machines," in *Proceedings of the 31st International Conference on Machine Learning*, ser. ICML '14, 2014.