ICIC Express Letters
Volume **10**, Number **12**, December **2016**

 ISSN 1881-803X
pp. **2891–2897**

A KERNEL SUPPORT VECTOR MACHINE FOR BIG DATA MINING BASED ON BUFFERED K-D TREE

Xia Gao[1] and Ruijun Li[2,*]

[1]Department of Mathematics
[2]Department of Computer Science
Jining Normal College
No. 59, Gongnong Avenue, Wulanchabu 012000, P. R. China
*Corresponding author: 281292188@qq.com

Received April 2016; accepted July 2016

Abstract. *Kernel Support Vector Machine (KSVM) is a common machine learning model for both classification and regression tasks. However, the computational cost for training a KSVM is high, especially for large datasets, leading to the failure of its application in big data analysis, which is a hot topic in recent years. In this paper, we proposed a new method for training KSVM effectively by making use of a divide-and-conquer mechanism. The method adopts a buffered k-D tree to divide data samples in a training set into different leaf structures together with buffer cells. The original problem is divided into many small but independent sub-problems which can be solved effectively by common KSVM classifiers. A min-max expansion strategy is used for expressing each buffer cell as a single vector. The KSVM classifiers are updated according to the data samples buffered in k-D tree. We evaluate the proposed method on two benchmark datasets and compare to two state-of-the-art training methods implemented in LibSVM to show its effectiveness.*

Keywords: Kernel support vector machine, Big data mining, Buffer k-D tree, Machine learning

1. **Introduction.** In data mining and machine learning, the Support Vector Machine (SVM) model is a common model for both classification and regression tasks [1]. By using the kernel trick [2], a kernel function defined in sample space can be plugged into an SVM model reflecting inner product of each sample pair, namely Kernel Support Vector Machine (KSVM). A good kernel function can induce a space with high or event infinity dimensions where data samples belonging to different concept classes can be separated linearly. Currently big datasets with millions of samples are common in analysis tasks, and people are interested in information and knowledge implied in big datasets, which are valuable for decision making, planning, and other supports for management [3]. Though data mining and machine learning algorithms are well studied and many application systems have been developed, most of them cannot process huge datasets effectively. Hence many methods in traditional machine learning have been adjusted to meet the requirements of big data environment. The main ideas of these studies fall into two categories. On the one hand, some approximations have been made to downgrade the time complexity of current learning methods so as to be able to process big datasets. For example, the methods proposed in [4] fall into this category. And on the other hand, new methods and models are derived based on current successful with some well designed strategies.

In this paper, we go the second way as mentioned above to design our method. The motivation of this work is initialized with the observation of the difficulty of combining individual learners trained by partitions of a big dataset. In many previous works based on dataset partition, there is a strong assumption, i.e., the samples in all partitions of the training dataset are independent identical distribution (i.i.d). Hence the strategies of

ensemble learning or weighted combination can work well. However, the i.i.d assumption may be not true in many cases, leading to the failure of partition training. We attempt to tackle this problem in the partition framework by buffering some important information of each partition which keeps the correlation between partitions in a unified model. The correlation kept in our tree structure makes it possible to combine partitions according to their entropy information, which in fact relaxes the i.i.d assumption. We propose to apply a Kernel Support Vector Machine (KSVM) as the main model of this study, which is trained with the partitioned training big dataset. At the same time, for each part of the training dataset, a buffered k-D tree is built to store some abstract information of the part which would be used for modeling the correlation between different parts in the model combination stage.

The parameters of a KSVM, i.e., α and b can be learned through Sequential Minimal Optimization algorithm (SMO) [5], a famous iterative learning method for KSVM training. However, when the training dataset size N is large, the SMO method is difficult to converge in reasonable time. A key concept of SVM model is that the learned classification hyperplane or the curve is determined by a small number of data points, namely Support Vectors (SV), which is guaranteed by the empirical risk minimization theory. Mathematically, the hyper parameter αs should contain a large number of zeros and the remainder are SVs. It can be seen that if a large dataset is generated through a relatively simple function (with low VC dimensions), there are only a few SVs of its classification or regression KSVM. For buffered k-D tree, it is a tree-based model initially designed for classification. Different from the famous decision tree, buffered k-D tree is a balanced binary tree. Any inner node of a k-D tree represents a split of the original dataset to two disjoint subsets. k nearest neighbor search can be performed effectively on a k-D tree. For our method, we use a trained k-D tree and add a buffer region to store some hits of each part of the training dataset, which can be used for the combination of the trained KSVM models.

This paper is structured as follows. Section 2 proposes the main method of this paper. We first give a formal definition of the problem to be solved. Then present the buffered k-D tree construction and training of KSVM learners, as well as the combination strategy of individual learners. In Section 3 we report the evaluation results of the proposed model on some benchmark data sets with comparison to two state-of-the-art methods. And finally we conclude the paper in Section 4.

2. **Training KSVM with Buffered k-D Tree on Big Dataset.**

2.1. **Problem definition.**

Before going further, we give a formal definition of the problem to be solved. We confine the problem to be classification in this study. Suppose there is a labeled dataset with huge amounts of samples: $D = \{(x, y)|x \in X, y \in \{-1, +1\}\}$, where $X \subseteq R^d$. Only binary classification is concerned. And we should note that the proposed method can apply to multiple-class dataset by using a one-vs-rest strategy. The goal is to learn a classifier $h : X \to \{-1, +1\}$ such that the loss of h on all future unseen test samples is minimized. Since the real loss of all future unseen data samples cannot be precisely evaluated, in this study we use the minimal loss on training dataset as the optimization goal instead. Meanwhile, a zero-one loss function is used for the classification accuracy evaluation, as shown in Equation (1):

$$Loss_{01}(h, D) = \frac{1}{|D|} \sum_{i=1}^{|D|} \sigma(h(x_i), y_i) \tag{1}$$

where h stands for a classifier trained with D and σ is a zero-one loss function evaluating the difference between the output of $h(x_i)$ and the ground truth value y_i. For brevity, we denote $H = \{h_1, h_2, \cdots, h_m\}$ to be m sub-learners trained by a KSVM-based learner

which is trained as the main learner by combination several learners with some important hits stored in a k-D tree. The final model can be expressed as the following:

$$h^* = Comb(T, H) \tag{2}$$

where $Comb$ is a procedure that adjusts all h_i in H through a weighted combination guided by a buffered k-D tree and T stands for a vector of combination weights.

2.2. **Building buffered k-D tree.** A buffered k-D tree locates at the core of our methods. We first give a formal definition together with a tree building algorithm. Generally speaking, a buffered k-D tree is a k-D tree associated with a set of buffers. As shown in Figure 1, a buffered k-D tree is composed of the following components: a balanced binary tree, a set of leaves, buffer structure, buffer processor and input queues.

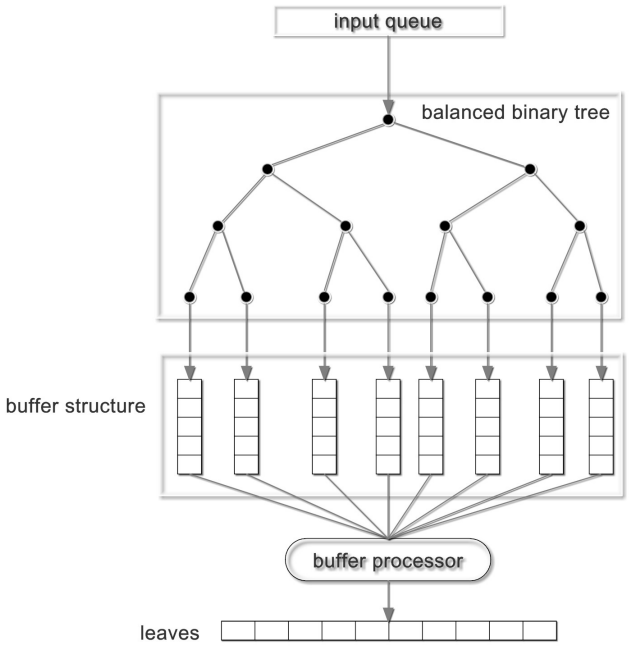

FIGURE 1. A sample buffered k-D tree

The components of a buffered k-D tree are working together in the following way. Data samples from training dataset are fed into the buffered k-D tree through the input queue component. The nodes of balanced binary tree represent median values of all dimensions of the training data set. For an inner node, its left branch represents the value is smaller than the corresponding median value, and its right branch otherwise. The balanced binary tree structure is stored in an array at a compressing way. The root node is stored in position 0. The left child of the node stored in position i is stored in position $2 * i$ and the right child is in position $2 * i + 1$. The leaf structure is a set of queues, each of which contains the training samples satisfied with the conditions set by the k-D tree. The buffered structure is designed for compressing the data samples in the queue of each leaf. Here the intuition is to generate some signature of data samples belonging to each leaf. A min-max expansion method is used for the compression. For each attribute, the minimal and maximal values among all data samples containing in this leaf are recorded and placed in a vector as the representation of this leaf. In this way the min-max vector has $2 * d$ elements where d is the dimension of the training dataset. Note that at this stage the concept label is not taken into consideration. Algorithm 1 gives the whole steps for constructing a buffered k-D tree effectively.

In Algorithm 1, two collection variables are preserved for leaf structure and buffer cells. Note that though a d balanced binary tree has $2^d - 1$ nodes, in our case there are many

Algorithm 1 Construction of buffered k-D tree

Require:
training data set $D = \{(x, y)\}$, dataset dimension d
Ensure:
leaf set L, buffer cells set B
initialize L to be a collection of 2^d empty queues
initialize B to be a collection of d empty arrays and m to be a d-ary vector
for $r = 1$ to d **do**
  $m(r) = mean(D(r))$
end for
for $t = 1, 2, \cdots, |D|$ **do**
  $index = 0$
  for $r = 1, 2, \cdots, d$ **do**
    if $x_i(r) < m(r)$ **then**
      $index = append(0)$
    else
      $index = append(1)$
    end if
  end for
  $leaf = Findleaf(L, index)$
  Add x_i to the queue indexed by $leaf$
  Min-max expansion of all remaining queues in L
  Update buffer cells B
end for
Remove empty queues out of L
return L and B

empty leaves, i.e., the associated queue is empty, which can be removed before buffering. To further avoid generating too many leaves, some attributes with small variances (less than some preset threshold) can be removed ahead. The procedure $Findleaf$ locates a leaf cell by the binary coding $index$, which means the actual path from the root to a leaf when processing a data sample. Note that at this stage the buffered k-D tree provides two kinds of information about the dataset. On the one hand, the leaves contain all data samples clustered by all fields which describe the dataset distribution. On the other hand, a set of buffer cells gives a brief representation of each leaf, which summarizes the main characteristic of the data samples belonging to the leaf. In case of processing big datasets, data samples stored in leaves would have to be discarded due to the limitation of processor and storage systems.

2.3. **KSVM training and model combination.** In this subsection, we are going to propose the training of KSVM models of sub training sets and combine them by the hits provided by the buffered k-D tree. A divide-and-conquer strategy is proposed to train a KSVM classifier. which is originated from the work proposed by Hsieh et al. [6]. The training task of KSVM is to solve the following problem:

$$\min_{\alpha} f(\alpha) = \alpha^T V \alpha - e^t \alpha \quad \text{s.t. } 0 \leq \alpha \leq C \tag{3}$$

where V is a Gram matrix whose element is defined as $V_{ij} = y_i y_j K(x_i, x_j)$. e is a vector of all elements equal to one. C is a preset constant balancing the loss and model complexity. The problem can be effectively solved by a quadratic optimizer when the training dataset does not contain too many samples [7]. For a test data sample x_t, the prediction of KSVM is $f(x_t) = \sum_{i=1}^{|D|} \alpha_i^* y_i K(x, x_i)$. However, when $|D|$ is very large, the traditional method to solve KSVM will fail. We apply a divide-and-conquer strategy to avoiding

the problem of directly solving large training dataset. We divide the original problem into s subproblems, which is equal to dividing the model parameters α into s groups and optimization is performed in each group individually. We denote the optimal parameters for the ith group as $\alpha(i)$. We propose to make use of a buffered k-D tree to provide some hits for which parts should be retrained in the conquer step. Generally, the algorithm searches all sub classifiers and their corresponding training data subsets and find one that has the largest error among all classifiers to retrain with its original sub dataset and the dataset far way from it. The distance between sub datasets is evaluated by the buffered k-D tree. Algorithm 2 shows the main steps of the above procedure.

Algorithm 2 Divide-and-conquer KSVM training

Require:
training data set $D = \{(x, y)\}$, partition number s, error threshold θ
Ensure:
trained classifier h for the whole dataset D
randomly divide D into s disjoint subsets
initialize a buffered k-D tree T
let E be a prior queue for recording the loss of each classifier
for $k = 1$ to s **do**
  train a KSVM with $D(k)$, mark as h_k
  add $D(k)$ to T through Algorithm 1
  add the loss of h_k to E
end for
while *true* **do**
  find the classifier that has the largest error with the help of E
  if the error is smaller than the threshold θ, break
  denote $h(k)$ to be the found classifier and $D(*)$ to be the sub dataset for training $h(k)$
  search T and find a leaf which is farthest from $D(*)$, denoted as $D(\#)$
  retrain h_k with $D(*)$ and $D(\#)$
end while
combine all h_k to be h
return h

In Algorithm 2, it first divides the training dataset into s subsets. Then train KSVM classifier and construct the buffered k-D tree at the same time. After that it launches an updated procedure to refine the trained KSVM classifiers. During this procedure, the KSVM classifier with largest training error will be retrained by its original training sub dataset and the one that is farthest away from it. We use Euclidean distance between buffers represented by min-max expansion vectors to evaluate the distance between two sub datasets. The complexity of Algorithm 2 depends on the error threshold θ and can be manually configured for comprising the model accuracy and convergence time.

3. **Evaluations.** The proposed method is evaluated on two benchmark datasets from UCI data repository [8] which are widely used for evaluation in big data analysis. Table 1 lists some details for the two datasets.

The two datasets are both multiple-label in which Amazon has 4 classes and Act has 6 classes. Considering the proposed method performs a binary classification, a one-vs-rest evaluation strategy is applied. In our evaluation, we set one category as positive and the others as negative in turn and mark the average accuracy as the model performance. However, the one-vs-rest strategy in building training set may lead to different prior distribution of concept labels since different categories may contain different amounts of

Table 1. Evaluation datasets from UCI data repository

Name	Size	Number of attributes	Category	Attribute type
Amazon	1500	10000	4	real
Act	9120	5625	6	real

samples. We use a training and test set division to avoid this unbalance. First a ratio between the sizes of training and test set is preset. Then a random division operation is performed to divide both the positive and negative sets according to the ratio. Thus, the percentage of positive and negative can be kept in all cases. For a stable result, we use a ten-fold validation for evaluation and the mean accuracy and variance are reported. In this evaluation the kernel function is radius basic function and the width parameter is set by a cross-validation tool provided by the implementation software package. The loss function for evaluating the model accuracy is zero-one loss, which is widely used for evaluating performance of classifiers. The KSVM is implemented by the LibSVM project [9] in Matlab platform.

Meanwhile, to show the effectiveness of the proposed method, we implement two current state-of-the-art methods for comparison. Table 2 shows the details of the methods for comparison.

Table 2. Two methods for performance comparison

Name	Paper	Description
Met1	Mu et al. [10]	compact hash bits KSVM training
Met2	Tian et al. [11]	inductive semi-supervised KSVM learning

First of all, we report the overall accuracy, as well as the variance, of three methods on two benchmark datasets. We highlight the best result on each dataset. From Table 3 we can see that the proposed method achieves the best result among all methods. The overall accuracy comparison indicates that the proposed method captures the essential distribution of the evaluation dataset. Though three methods adopt partition strategy, the proposed method has an effective combination mechanism with the help of a buffered k-D tree.

Table 3. Overall accuracy and variance of three methods (%)

	Met1	Met2	KSVM-DT
Amazon	80.3 ± 3.1	78.8 ± 2.9	$\mathbf{84.2 \pm 2.6}$
Act	82.0 ± 4.5	83.1 ± 3.8	$\mathbf{86.9 \pm 2.3}$

Next we report the relation between the number of divisions and the model convergence time. A fast convergence time is important for a big data analysis model. In this evaluation, we record the number of iterations before the point that the model loss is less than the threshold. Figure 2 shows the evaluation results for both datasets. From Figure 2 we can see that the number of divisions has much effect on the convergence time. And the best convergence time for both datasets is around 80.

4. **Conclusions.** In this paper we proposed a method for big data classification based on a division strategy. The method adopts kernel support vector machine as base classifier and constructs a buffered k-D tree for retraining the poor classifiers. The method works under a divide-and-conquer framework and it can avoid extremely large computational cost. A buffered k-D tree stores some hints of sub datasets to KSVM classifiers, which is used for retraining so as to improve the quality of classifiers bounded by a threshold.

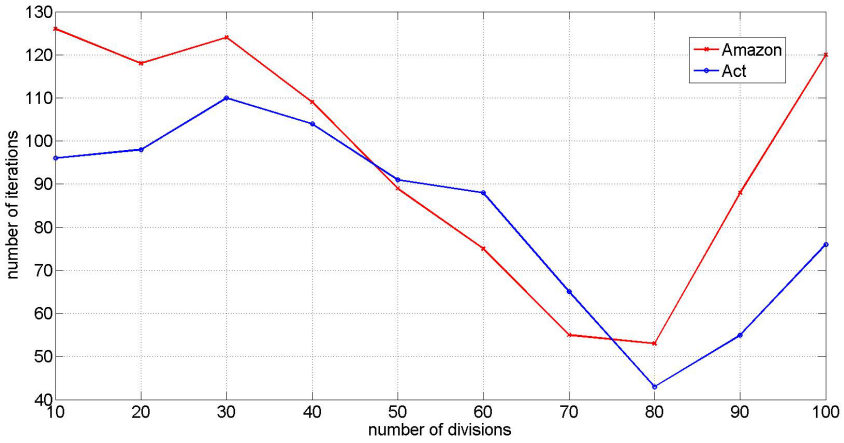

FIGURE 2. The relation between the number of divisions and the model convergence time

A min-max expansion expression is imposed on each buffered partition, which is a set of data samples, to generate a unique representation. Such expansion method has been widely used in many machine learning tasks. The proposed method is evaluated on two benchmark datasets previously applied for big data analysis and the results show that it is superior to current state-of-the-art methods. The proposed method and its division strategy can be used in other application context of big data mining and machine learning. Future research directions include new data representation for the buffered k-D tree and the extensions of the proposed method to multiple-class cases and regression.

Acknowledgment. This work is supported by the university scientific research project of Inner Mongolia (No. NJZY283, NJZC14292 and NJZY283).

REFERENCES

[1] T. Howley and M. G. Madden, The genetic kernel support vector machine: Description and evaluation, *Artif. Intell. Rev.*, vol.24, nos.3-4, pp.379-395, 2005.

[2] G. Wu, E. Y. Chang and N. Panda, Formulating distance functions via the kernel trick, *Proc. of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, New York, NY, USA, pp.703-709, 2005.

[3] J. Huang, Z. Kalbarczyk and D. M. Nicol, Knowledge discovery from big data for intrusion detection using LDA, *Proc. of the IEEE International Congress on Big Data*, Washington, D.C., USA, pp.760-761, 2014.

[4] C. K.-S. Leung, R. K. MacKinnon and F. Jiang, Reducing the search space for big data mining for interesting patterns from uncertain data, *Proc. of the IEEE International Congress on Big Data*, Washington, D.C., USA, pp.315-322, 2014.

[5] F. R. Bach, G. R. G. Lanckriet and M. I. Jordan, Multiple kernel learning, conic duality, and the SMO algorithm, *Proc. of the 21st International Conference on Machine Learning*, New York, NY, USA, pp.6-12, 2004.

[6] C. Hsieh, S. Si, and I. S. Dhillon, A divide-and-conquer solver for kernel support vector machines, *CoRR*, http://arxiv.org/abs/1311.0914, 2013.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] K. Bache and M. Lichman, *UCI Machine Learning Repository*, http://archive.ics.uci.edu/ml, 2013.

[9] C.-C. Chang and C.-J. Lin, Libsvm: A library for support vector machines, *ACM Trans. Intell. Syst. Technol.*, vol.2, no.3, pp.1-27, 2011.

[10] Y. Mu, G. Hua, W. Fan and S.-F. Chang, Hash-SVM: Scalable kernel machines for large-scale visual classification, *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, Washington, D.C., USA, pp.979-986, 2014.

[11] X. Tian, G. Gasso and S. Canu, A multiple kernel framework for inductive semi-supervised SVM learning, *Neurocomput.*, vol.90, pp.46-58, 2012.