VOKNN: Voting-based Nearest Neighbor Approach for Scalable SVM Training

Saeed Salem[*]
Department of Computer Science
North Dakota State University
Fargo, ND 58102, USA
saeed.salem@ndsu.edu

Khedidja Seridi[†]
INRIA Dolphin Project
Opac LIFL CNRS
Villeneuve d'Ascq, France
khedidja.seridi@inria.fr

Loqmane Seridi
Math. and Comp. Sci. & Eng. Division
King Abdullah Uni. of Sci. and Tech.
Thuwal, Saudi Arabia 23955-6900
loqmane.seridi@kaust.edu.sa

Jianfei Wu
Department of Computer Science
North Dakota State University
Fargo, ND 58102, USA
jianfei.wu@ndsu.edu

Mohammed J. Zaki
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180, USA
zaki@cs.rpi.edu

ABSTRACT

Support Vector Machines (SVMs) are a powerful classification technique. For large datasets, SVM training is computationally expensive. In this paper, we propose **VOKNN**, a novel nearest neighbor-based voting data reduction algorithm for SVM training. The proposed approach eliminates part of the training dataset based on the votes each point gains from all the other points in the other class. We demonstrate the effectiveness and efficiency of **VOKNN** on several real datasets. SVM classification models built on the reduced datasets achieve comparable classification accuracy as those built on the original training datasets. In few cases, SVM classification accuracy has improved significantly when the SVM is trained on a reduced training dataset.

Categories and Subject Descriptors

H.2.8 [**Data mining**]: Support Vector Machines, Kernel Nearest Neighbors

1. INTRODUCTION & RELATED WORK

Support vector machines (SVM) [13] have attracted significant attention from the research community, thanks in part to their classification power. Recently it has been voted among the top 10 most influential data mining algorithms [15]. The classification power of SVM can be attributed to its excellent generalization performance [3]. One of the key features of SVM is its ability to learn non-linear decision functions. It does so by mapping the data from the original space, $\mathcal{L}$, into a higher dimensional Euclidean space, $\mathcal{H}$ [3]. The SVM then tries to find a linear separation between the classes in the new feature space. SVM utilizes the kernel function approach to compute the dot product of the points in the new space without having to map every point to its image in the new space. Define a mapping function $\Phi : \mathcal{L} \rightarrow \mathcal{H}$. For every two points, $\mathbf{x}_i$ and $\mathbf{x}_j$, in the original space, the kernel trick allows the computation of the dot product in the new space in terms of the coordinates of the points in the original space, i.e., $< \Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j) > \; = f(\mathbf{x}_i, \mathbf{x}_j)$.

The training step of SVM involves solving a large quadratic programming (QP) optimization problem which is computationally expensive, specially for large training datasets. For large datasets, the computational complexity of SVM training is $O(n^2)$, where n is the number of training points [3, 14]. To overcome the computational complexity of SVM training, several methods have been proposed which try to speed up SVM training. Many of these methods are aimed at speeding up the solution of the quadratic programming problem. The Chunking algorithm considers a chunk of the dataset iteratively [11]. The Chunking approach starts by a random subset of the data and iteratively keeps all the non-zero Lagrange multipliers and adds the examples that violate the optimality conditions [11]. Another method for speeding up SVM training is the Sequential Minimal Optimization (SMO) which breaks the large QP optimization problem into smaller more manageable QP problems [10]. The scalability

[*] Corresponding Author.

[†] This author contributed to this work while visiting Rensselaer Polytechnic Institute.

KDD-LDMTA'10, July 25, 2010, Washington, DC, USA.
Copyright 2010 ACM 978-1-4503-0215-9/10/07 ...$10.00.

problem of SVM training is an active area of research with an entire workshop on large scale kernel machines devoted to addressing parallel algorithms for SVM training and more efficient solvers [2].

Another direction of research aims at sampling the training dataset and then training the SVM on the reduced dataset. This line of work is built on the premise that in most cases only a small subset of the data are really support vectors (SVs). Thus, removing the points which are not support vectors will not harm the accuracy of the trained SVM classifier. The goal of data reduction here is to get a small subset of the training data without loosing any of the support vectors. Not eliminating any of the support vectors is a hard requirement and thus a relaxed condition is to reduce the data by removing the points which are most likely not support vectors.

Several algorithms follow the sampling approach to reduce the training dataset. An algorithm proposed in [8] uses k-means clustering to identify groups in the data. Once the clusters are found, crisp clusters which have samples from the same class are extracted. Interior points are removed from the crisp clusters, thus resulting in a reduced dataset. The Sample Reduction by Data Structure (SR-DSA) approach introduced in [14] utilizes the structure of the data to keep the data points which are likely to be support vectors. SR-DSA adopts hierarchical clustering to cluster the data in each class. After the clusters in each class are found, interior points are removed. Moreover, the SR-DSA algorithm removes exterior points which are far from the opposite class. In SR-SDA, clustering can be done in either the input space or the kernel space. Another approach (KBK-SR) employs the Kernel Bisecting k-means to cluster the data in each class [9]. The KBK-SR algorithm uses similar removal techniques for data removal as the ones proposed in the SR-DSA algorithm. The only difference is the clustering method used. A neighborhood preprocessing approach proposed in [5] decides to keep or remove a data point based on its neighbors' (within a distance threshold) characteristics. A point is removed if the majority of its neighbors have the same class, or are from the opposite class. Moreover, a point is removed if it is an isolated sample.

Existing data reduction algorithm are still computationally expensive, Both the SR-DSA and KBK-SR algorithms require the computation of the kernel matrix since they do the clustering in the kernel space. Thus, they become as computational expensive as training the SVM itself. The neighborhood preprocessing approach looks at the neighborhood within a distance d. In datasets with different densities, it is hard to figure out a good value for d. Moreover, some isolated points can be support vectors. There might be no points around them within a distance d but at the same time, they can be neighbors for points from the other class.

In this paper, we propose, **VOKNN**, a data reduction algorithm for SVM training based on the notion of opposite k nearest neighbor voting. In **VOKNN**, every point votes for the k closest points from the other class. After the voting process is over, points which receive zero votes (or a user-specified percentage of the votes) are eliminated. The proposed approach is effective in reducing the number of training samples while at the same time retaining the points which are most likely to be support vectors. To summarize, we made the following key contributions in this work:

1. We define a new voting function where each point contributes to the score of the closest k opposite nearest neighbor. We propose a data reduction algorithm, **VOKNN**, based on this voting mechanism.

2. We show that the proposed approach works in the input space and kernel space as well.

3. We perform experiments on several real dataset to demonstrate the effectiveness and the performance of the proposed approach.

The rest of the paper is organized as follows. The next section introduces the notion of opposite k nearest neighbor and introduces the **VOKNN** algorithm. The experimental evaluation of **VOKNN** on real datasets in presented in Section 3. Section 4 concludes the paper.

2. VOTING-BASED DATA REDUCTION

Our **VOKNN** is based on the idea that points which are on the decision boundaries are more likely to be support vectors (SVs). We propose a voting mechanism that assigns more votes to the points on the decision boundaries.

Given a training dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$. The goal of the training data reduction is to choose a subset of the dataset, i.e., $\mathcal{D}_r \subseteq \mathcal{D}$. An effective data reduction algorithm must meet two criteria: First, the SVM model trained on the reduced data must have a comparable classification power as the SVM model trained on the original dataset. Second, the combined time of the reduction algorithm and SVM training on the reduced data must be less than the SVM training time on the original training dataset.

The main idea in the proposed approach is to assign a score to each point in the training dataset, and eliminate data points with low score. The scoring scheme we adopted is based on a voting mechanism. Each point votes for the closest k points from the other class. When the voting process is over, each point garners a number of votes which quantify how likely the point is around the boundaries. Points with low or zero votes are candidates to be eliminated from the training dataset.

For a data point $\mathbf{x}_i$, define the opposite k nearest neighbors, $oknn(\mathbf{x}_i)$, as the closet k points to $\mathbf{x}_i$ from the opposite class.

Definition 1. For any point $\mathbf{x}_i$, the opposite k nearest neighbors, denoted as $oknn(\mathbf{x})$, are the k closest points to $\mathbf{x}_i$ which belong to the other class:

$$oknn(\mathbf{x}_i) = \{closest\ k\ \mathbf{x}_j \mid y_i \neq y_j\}$$

Figure 1 illustrates the opposite k nearest neighbors idea with $k = 3$. An arrow from $\mathbf{x}_i$ to $\mathbf{x}_j$ indicates that $\mathbf{x}_j \in oknn(\mathbf{x}_i)$

The score of a point $\mathbf{x}_i$ is the number of times $\mathbf{x}_i$ falls in the opposite k nearest neighbors of other points.

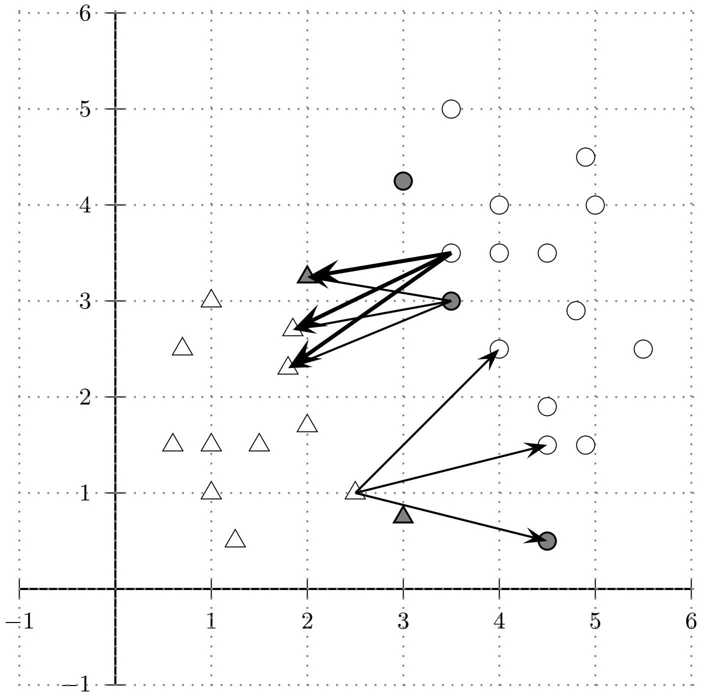


Figure 1: Voting Mechanism: Each point in one class votes for k closest points from the other class.

Definition 2. The score of any point $\mathbf{x}_i$ is cardinality of the set of points in whose opposite k nearest neighborhood $\mathbf{x}_i$ falls.

$$vote(\mathbf{x}_i) = |\{\mathbf{x}_j \mid \mathbf{x}_i \in oknn(\mathbf{x}_j)\}|$$

The pseudo-code for **VOKNN** is provided in Figure 2. It accepts the training dataset, $\mathcal{D}$ and a vote threshold, α. The output of the algorithm is the reduced training dataset, $\mathcal{D}_r$. The algorithm consists of two steps: the first step calculates how many votes each point garners (lines 4-8), and the second step is to eliminate points which garner low votes (lines 10-14).

The algorithm starts by initializing the vote for every data point in line 2. In the voting step, for every point, we first find the closest k points from the opposite class (line 6). This is the main idea of the algorithm. The vote of each point in the closest k points is incremented by 1. Once the voting step is over, the elimination step starts. In the elimination step, we only keep the points which have a number of votes that is larger than a percentage of the total number of votes a point can get. The threshold is specified by the variable $0 \leq \alpha < 1$. When $\alpha = 0$, we basically keep every point that has at least one vote and eliminate every point with zero vote. Points with large number of votes are the most likely border points and thus likely to be support vectors.

Next, we introduce two variations of **VOKNN**. The main difference between the two variations is the space they operate in. In terms of the algorithm, the only difference is in which space the opposite k nearest neighbors is computed (line 6). The first variation works in the input space in which the data lives, while the other variation works in the kernel space to which the SVM training is mapping the data.

```
VOKNN($\mathcal{D}, \alpha$):

Initialization:
1. $vote(i) = 0\ \forall \mathbf{x}_i \in \mathcal{D}$, and $\mathcal{D}_r = \emptyset$
2. Let $P = |\{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid y_i = +1\}|$, # of positive points
3. Let $N = |\{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid y_i = -1\}|$, # of negative points

Voting:
4. $\forall \mathbf{x}_i \in \mathcal{D}$
5.     Get the opposite $k$ nearest neighbors for point $\mathbf{x}_i$
6.     $S = oknn(\mathbf{x}_i)$
7.     $\forall \mathbf{x}_j \in S$, for every point in $S$, increment its vote.
8.         $vote(j) + +$

Elimination: Eliminate points with low votes.
10. $\forall \mathbf{x}_i \in \mathcal{D}$
11.     if $y_i == +1$ and $vote(i) > \alpha * N$
12.         $\mathcal{D}_r = \mathcal{D}_r \cup \mathbf{x}_i$
13.     if $y_i == -1$ and $vote(i) > \alpha * P$
14.         $\mathcal{D}_r = \mathcal{D}_r \cup \mathbf{x}_i$

15. return $\mathcal{D}_r$
```

Figure 2: Pseudo-code for VOKNN: SVM training data reduction using voting.

2.1 Reduction using Nearest Neighbors

In this variation, we find the k nearest neighbors from the opposite class, $oknn(\mathbf{x}_i)$, in the input space. We first build two kd-trees, kd^+ and kd^-, one for each set of data points in each class. Then for each point $\mathbf{x}_i$, we query the tree with the opposite class label to get the k nearest neighbors with the opposite class label.

2.1.1 Computational Complexity

Building the two kd-trees takes $O(P. \log P\ +\ N. \log N)$, where P and N are the sizes of the positive (class $y_i = +1$) and negative (class $y_i = +1$) data sets respectively and $P + N = n$. Considering that both P and N are $O(n)$, the total construction time is $O(n \log n)$ [6]. Querying for the k nearest neighbors in a kd-tree with m points in d dimensions takes $O(m^{1-\frac{1}{d}} + k)$. Thus complexity of the all the kd-tree queries is $O(P * (N^{1-\frac{1}{d}} + k) + N * (P^{1-\frac{1}{d}} + k))$, which is $O(n * (n^{1-\frac{1}{d}} + k))$. The total time complexity for **VOKNN** in the input space is thus $O(n \log n\ +\ n * (n^{1-\frac{1}{d}} + k))$. When $n > 2^d$, the complexity becomes $O(n \log n)$. A more efficient way of computing all the queries is to use the all nearest neighbors with a total complexity of $O(k.n \log n)$ [12]. Again the total time complexity for **VOKNN** is $O(n \log n)$.

2.2 Reduction using Kernel Nearest Neighbors

In the previous section we computed the opposite k nearest neighbors in the input space. That would work fine if the dataset is linearly separable. In most cases, the data is not linearly separable and the SVM finds a separating hyperplane in the kernel space. Here we consider the case where the classes overlap to some extent so that a perfect separation is not possible. Can we extend the same idea of computing the opposite k nearest neighbors to kernel space and eliminate data points based on how the data is structured in the kernel space?

Table 1: Comparison of the performance of KBK-SR and VOKNN. For each dataset, we report the accuracy and the training time of the SVM trained on the original dataset without reduction.

Dataset	Method	# of points (Reduction%)	Sampling and Training Time (s)	Accuracy (%)
ijcnn1	No sampling	49990	97.32	92.79
	VOKNN	16122 (67.75)	19.69	92.74
	KBK-SR	n/A	N/A	N/A
w8a	No sampling	33166	63.31	98.06
	VOKNN	4920 (85.17)	39.48	93.65
	KBK-SR	N/A	N/A	N/A
diabetes	No sampling	512	0.02	76.95
	VOKNN	480 (6.25)	0.02	76.95
	KBK-SR	90 (82.42)	–	66.80
fourclass	No sampling	575	0.01	80.84
	VOKNN	361 (37.22)	0.00	80.84
	KBK-SR	91 (84.17)	–	79.09
german	No sampling	766	0.06	74.79
	VOKNN	747 (2.48)	0.09	74.79
	KBK-SR	139 (81.85)	–	69.66

We propose the kernel opposite k nearest neighbors. One key feature of the SVM approach is the ability to solve problems with a non-linear decision boundary. The main idea is to map the original d-dimensional points into a higher dimensional space with $d' > d$ dimensions, where the points can possibly be linearly separated. Define a mapping from the input space to the kernel space as follows: $\Phi : \mathcal{L} \rightarrow \mathcal{F}$. Define the kernel function between any two points $\mathbf{x}_i$ and $\mathbf{x}_j$ in terms of the inner product of the the points image in the kernel space as follows:

$$K(\mathbf{x}_i, \mathbf{x}_j) \;=\; < \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) >$$

Thanks to the the kernel trick, the inner product of any two points in the kernel space is computed without mapping the data to the new space:

$$< \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) > \;\; = f(\mathbf{x}_i, \mathbf{x}_j)$$

Moreover, the distance between any two points in the feature space can be computed using the kernel function and the vectors of the two points in the input space [16]. Therefore, we can compute the squared Euclidean distance in the kernel space as follows:

$$\begin{aligned} d^2(\Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)) &= \|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_j)\|^2 \\ &= K(\mathbf{x}_i, \mathbf{x}_i) - 2K(\mathbf{x}_i, \mathbf{x}_j) + K(\mathbf{x}_j, \mathbf{x}_j) \end{aligned}$$

2.2.1 *Complexity of* **VOKNN** *using kernel knn*

To compute the *oknn* in the kernel space, first we have to find $K(\mathbf{x}_i, \mathbf{x}_j)$ for all $\mathbf{x}_i$ in once class and $\mathbf{x}_j$ is in the other. The complexity for this step is $O(P \times N)$. To find $oknn(\mathbf{x}_i)$ for a given point, we find the distances (in the kernel space) between $\mathbf{x}_i$ and all the other points in the other class. We select the k points with the smallest distances using a simple variation of the quicksort algorithm. This can be achieved in $O(n)$. Thus, to complete the voting step and query for every point its k nearest neighbor in the opposite class takes a total of $O(PN + NP)$. Considering that both P and N are of $O(n)$, the total time complexity of **VOKNN** in the kernel space is $O(n^2)$.

It has been shown that for some specific kernel functions or parameters, the kernel nearest neighbor degenerates to the conventional nearest neighbor [16]. Therefore, for some kernel functions, the two variations of **VOKNN** are equivalent in terms of the reduction, and thus in accuracy.

3. EXPERIMENTS

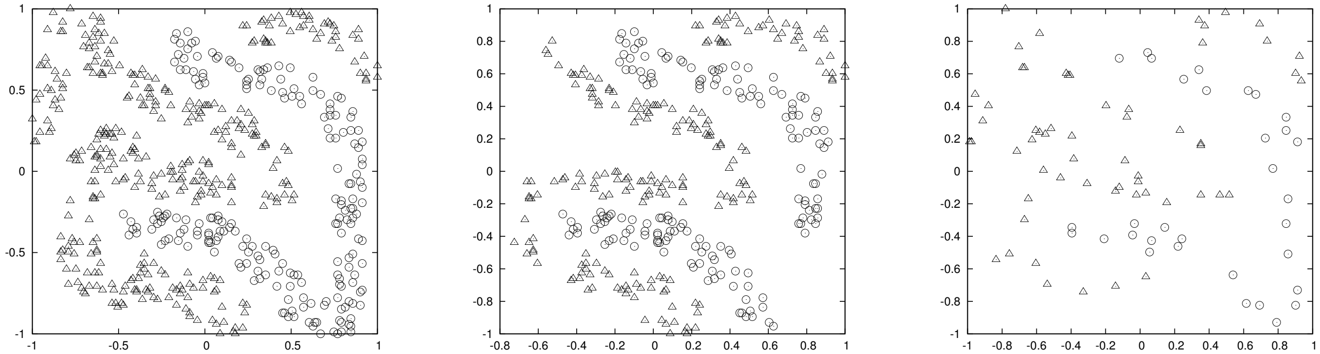
To assess the performance and the effectiveness of the **VOKNN** approach, we tested the proposed reduction algorithms on several real datasets from the UCI Machine Learning Repository [1], and other real datasets from different sources [10, 4]. The objective of the experiments is to demonstrate the following: First, **VOKNN** significantly reduces the training datasets. Second, the SVM model trained on the reduced training dataset has comparable (sometimes better) generalization performance when compared to the SVM model trained on the whole dataset. Last, the time required for reducing the data and SVM training on the reduced data is less than the time required to train the SVM on the whole dataset.

We used the training and testing dataset as originally provided in [1, 10, 4]. When the training and testing are not separated and for these datasets we have divided the whole dataset into training dataset (two thirds) and testing dataset (one third) using stratified selection.

The only other reduction algorithm for which we were able to obtain an implementation was the KBK-SR algorithm. The authors have provided us with a MATLAB implementation. Therefore, it is not fair to compare the timing of the KBK-SR algorithm against our **VOKNN**, thus the time for KBK-SR is not reported. For other algorithms mentioned in the related work section, we could not get any implementation. All the experiments were run on a 2.1GHz PC machine with 4GB RAM with Linux OS. We have implemented **VOKNN** in C++ and used the ANN library[1] for k nearest neighbor search.

For SVM training, we used the LIBSVM package [4] with the radial basis function (RBF) with default parameters. Since kernel nearest neighbor in the RBF kernel space degenerates to the conventional nearest neighbor [16], we only show the results **VOKNN** in the original space. Reducing

[1] Available at: `http://www.cs.umd.edu/~mount/ANN`



(a) Original Training Data (b) Reduction with **VOKNN** (c) Reduction with KBK-SR

Figure 3: Training Data Reduction for the fourclass dataset: SVM trained on the original data (shown in (a)) has an accuracy of 80.84%**. SVM has an accuracy of** 80.84% **when trained on the reduced data in (b) and** 79.09% **when trained on data in (c).**

the data using the kernel nearest neighbor is computationally more expensive than the reduction in the original space and we only do it if necessary, e.g., if we only have the kernel matrix and no coordinates.

Table 1 shows the comparison between the KBK-SR algorithm and **VOKNN**. The accuracy of the SVM trained on the reduced data using the **VOKNN** algorithm is very comparable to the accuracy of the SVM trained on the original training dataset.

For small datasets, there is no saving in the running time. In fact, SVM training on the original data is faster than **VOKNN** for small datasets. This is due to the time of building the kd-trees and the I/O time in **VOKNN**. For larger datasets (the first two), the advantage of data reduction becomes clear. For the ijcnn1 dataset [10], SVM trained on the reduced data with **VOKNN** has the same accuracy as the one trained on the original datasets. In fact, we can get higher accuracy (93.10%) for the ijcnn1 dataset if we set $k = 2$. Training on the reduced data is almost 5 times faster than training on the original data. KBK-SR ran for hours on the first two datasets without reporting any results. Figure 3 shows how the training data is reduced for the fourclass dataset [7]. KBK-SR reduces the data much more than **VOKNN**, however, this reduction is at the cost of accuracy as evident in the case of german and diabetes datasets.

3.1 Effect of k on Reduction and Accuracy

The reduction rate, accuracy, and training time depend on the value k. Figure 4 shows the effect of changing k on the reduction rate (a), on the SVM classification accuracy trained on the reduced dataset (b), and on the combined reduction and training time (c). Empirically for $k = 5$, we get a good reduction rate while getting high accuracy and significant time saving. The value k controls the voting process: for large k values, we have more votes being used and thus fewer points have zero votes. So, it is natural that the reduction rate decreases as the value of k increases. Moreover, when we eliminate less data points, the chances of accidently removing support vectors are less and thus the accuracy improves as the value k is increased.

4. CONCLUSION

In this paper, we proposed a scalable data reduction algorithm for SVM training. The proposed approach is based on the simple idea that support vectors tend to be around the decision boundaries. In the proposed approach, each point votes for the k closest points from the other class. Points with zero votes are removed from the datasets.

Our experimental evaluation shows that this simple approach, **VOKNN**, is effective in reducing the datasets while at the same time retaining all (or most) of the support vectors. Retaining all (or most) of the support vectors allows the SVM trained on the reduced training dataset to have comparable accuracy. In few cases, the SVM trained on the reduced dataset had higher accuracy. This can be due to outliers in the initial training dataset.

Our next steps are to try to investigate how using different distance measures affect the reduction rate and quality. We also want to improve our approach to ensure that the cases where some points which might be support vectors but are themselves outliers are not removed.

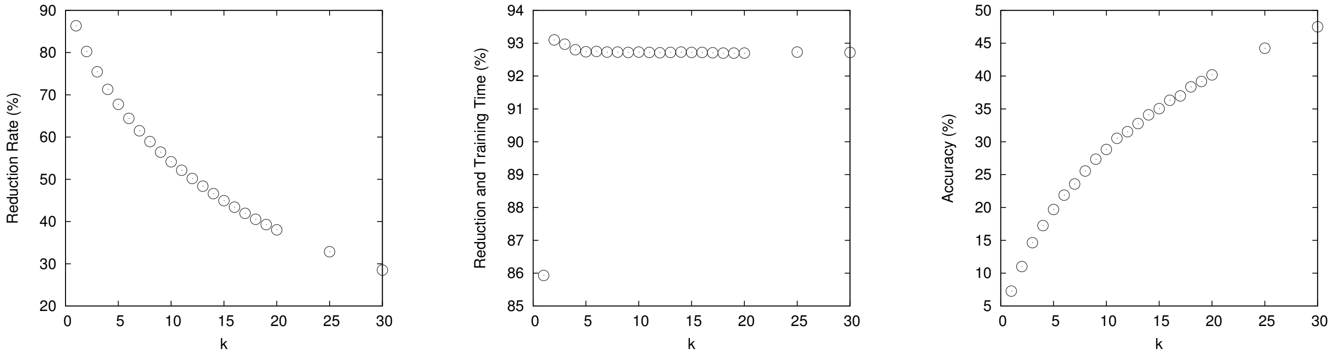
5. ACKNOWLEDGMENTS

We would like to extend our thanks to Xiao-Zhang Liu, and Guo-Can Feng, the authors of the KBK-SR algorithm, for providing us with a MATLAB implementation of their algorithm. We also thank the anonymous reviewers for their comments and suggestions. This work was supported in part by NSF Grants EMT-0829835 and EIA-0103708, NIH Grant 1R01EB0080161-01A1, and NIH grant number P20 RR016741 from the INBRE program of the National Center for Research Resources.

References

[1] A. Asuncion and D. J. Newman. UCI machine learning repository, 2007. Irvine, CA: University of California, Irvine, School of Information and Computer Sciences, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[2] L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors. *Large Scale Kernel Machines.* MIT Press, Cambridge, MA., 2007.



(a) Reduction Rate vs. k (b) Accuracy vs. k (c) Time vs. k

Figure 4: Effect of varying k for the ijcnn1 dataset. (a) on the reduction rate, (b) on the accuracy, and (c) on the running time of reduction and training combined.

[3] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[4] C. Chang and C. Lin. LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

[5] G. Chen, J. Xu, and X. Xiang. Neighborhood preprocessing svm for large-scale data sets classification. *Fuzzy Systems and Knowledge Discovery, Fourth International Conference on*, 2:245–249, 2008.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[7] T. K. Ho and E. M. Kleinberg. Building projectable classifiers of arbitrary complexity. In *the 13th International Conference on Pattern Recognition*, pages 880–885, Vienna, Austria, August 1996.

[8] R. Koggalage and S. Halgamuge. Reducing the number of training samples for fast support vector machine classification. *Neural Information Processing*, 2(3):57–65, 2004.

[9] X. Z. Liu and G. C. Feng. Kernel bisecting k-means clustering for svm training sample reduction. In *19th International Conference on Pattern Recognition*, Tampa, FL, USA, December 2008.

[10] J. Platt. Fast training of support vector machines using sequential minimal optimization. *In SchÃűlkopf, B., Burges, C.J.C. and Smola, A.J. (eds), Advances in Kernel Methods - Support Vector Learning*, pages 185–208, 1999.

[11] J. Thorsten. Making large-scale support vector machine learning practical. *Advances in kernel methods: support vector learning*, pages 169–184, 1999.

[12] P. M. Vadyia. An o(n log n) algorithm for the all-nearest-neighbors problem. *Discrete and Computational Geometry*, 4(1):101–115, 1989.

[13] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[14] D. Wang and L. Shi. Selecting valuable training samples for svms via data structure analysis. *Neurocomputing*, 71(13-15):2772–2781, 2008.

[15] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[16] K. Yu, L. Ji, and X. Zhang. Kernel nearest-neighbor algorithm. *Neural Processing Letters*, 15(2):147–156, 2002.