Localized Support Vector Machine and Its Efficient Algorithm

Haibin Cheng Pang-Ning Tan Rong Jin

Abstract

Nonlinear Support Vector Machines employ sophisticated kernel functions to classify data sets with complex decision surfaces. Determining the right parameters of such functions is not only computationally expensive, the resulting models are also susceptible to overfitting due to their large VC dimensions. Instead of fitting a nonlinear model, this paper presents a framework called **Localized Support Vector Machine** (LSVM), which builds multiple linear SVM models from the training data. Since each model is designed to classify a particular test example, it has high computational cost. To overcome this limitation, we propose an efficient implementation of LSVM, termed **Profile SVM** (PSVM). PSVM partitions the training examples into clusters and builds a separate linear SVM model for each cluster. Our empirical results show that (1) Both LSVM and PSVM outperform nonlinear SVM on the majority of the evaluated data sets; and (2) PSVM achieves comparable accuracy as LSVM but with significant computational savings.

1 Introduction

Nonlinear Support Vector Machine (SVM) has been widely used in many applications, from text categorization to protein classification. Despite its well-documented successes, nonlinear SVM must employ sophisticated kernel functions to fit data sets with complex decision surfaces. Determining the right parameters of such functions is not only computationally expensive, the resulting models are also susceptible to overfitting when the number of training examples is small due to their large VC dimensions.

Instead of learning such a complex global model, an alternative strategy is to build simple models that fit the data in the local neighborhood around a test example. A well-known technique that employs such a strategy is the K-nearest neighbor (KNN) classifier [4]. KNN does not require any prior assumptions about characteristics of the data and its decision surfaces, thus avoiding the unnecessary bias of global function fitting [1]. Nevertheless, because of its lazy learning scheme, classifying test examples is computationally expensive.

In this paper, we present a framework called *Localized Support Vector Machine* (**LSVM**), which leverages the strengths of SVM and KNN. Instead of using sophisticated kernel functions, LSVM builds a linear SVM model for each test example using only the training examples located in the vicinity of the test example. We empirically show that such a strategy often leads to significant improvement in accuracy over nonlinear SVM.

Since each model is designed for a particular test example, LSVM can be very expensive when the number of test examples is large. To overcome this problem, we propose an efficient technique called *Profile Support Vector Machine* (**PSVM**). The intuition behind PSVM is that the models for test examples in the same neighborhood tend to have similar support vectors. Therefore, instead of building a separate model for each test example, PSVM partitions the training data into clusters and builds a linear SVM model for each cluster. PSVM also assigns each test example to its closest cluster and applies the corresponding linear SVM model to predict its class label. By reducing the number of constructed models, PSVM maintains the high accuracy of LSVM without its computational overhead.

2 Preliminaries 准备工作

Consider a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ is an instance of the input space $\Re^d$ and $y_i \in \{-1, +1\}$ is its corresponding class label. In KNN [4], the label of a test example is determined by the training examples in its local neighborhood. Since KNN is sensitive to the neighborhood size, the weighted K-nearest neighbor [6] classifier was introduced, which assigns a weight factor to each training example based on its similarity to the test example. The posterior probability of a test example $\mathbf{x}$ is computed as follows:

$$p(y|\mathbf{x}) = \frac{\sum_{i=1}^{n} \delta(y, y_i)\sigma(\mathbf{x}, \mathbf{x}_i)}{\sum_{i=1}^{n} \sigma(\mathbf{x}, \mathbf{x}_i)} \tag{2.1}$$

where $\sigma(\mathbf{x}, \mathbf{x}_i)$ is the similarity between $\mathbf{x}$ and $\mathbf{x}_i$ while

$$\delta(y, y_i) = \begin{cases} 1 & \text{if } y = y_i \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Without loss of generality, we assume that the weight factor σ is bounded between 0 and 1.

Support Vector Machine [7] finds an optimal hyperplane to separate training examples from different classes by maximizing the classification margin [5, 2]. It

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

is applicable to nonlinear decision surfaces by employing a technique known as kernel trick, which projects the input data to a higher-dimensional feature space where a separating hyperplane can be found. During model building, a nonlinear SVM is trained to solve the following optimization problem:

$$\begin{aligned} (2.3) \qquad \max_{\alpha_1 \cdots \alpha_n} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s. t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \ i = 1, 2, \ldots, n \end{aligned}$$

where ϕ is the kernel function and α_i is the weight assigned to the training example $\mathbf{x}_i$. The kernel function ϕ is used to compute the dot product $\varphi(\mathbf{x}_i) \bullet \varphi(\mathbf{x}_j)$, where φ is a function that maps an instance to its higher dimensional space. Data points with $\alpha_i > 0$ are called support vectors. Once the weights have been determined, a test example $\mathbf{x}$ is classified as follows:

$$(2.4) \qquad y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \phi(\mathbf{x}, \mathbf{x}_i) \right)$$

3 Localized Support Vector Machine (LSVM)

To overcome the limitations of KNN and nonlinear SVM, a natural idea is to combine the strengths of both methods. Zhang et al. [10] proposed a hybrid algorithm called KNN-SVM [10] for visual object recognition. Their algorithm selects the K nearest neighbors of each test example and builds a local SVM model from its nearest neighbors [9]. This method is a straightforward adaptation of KNN to SVM and suffers from a number of limitations. First, it is sensitive to the neighborhood size. Second, it is not quite flexible because the neighborhood size is fixed for every test example. Finally, their method decouples nearest-neighbor search from the SVM learning algorithm. Once the K nearest neighbors have been identified, the SVM algorithm completely ignores their similarities to the given test example when solving the dual optimization problem given in (2.3).

This motivates us to develop a more integrated framework called Localized Support Vector Machine (LSVM), which incorporates the neighborhood information directly into SVM learning. The rationale behind LSVM is to reduce the impact of support vectors located far away from a given test example. This can be accomplished by weighting the classification error of each training example according to its similarity to the test example. The similarity is captured by a weight function σ, similar to the approach used by weighted KNN.

Let $\overline{\mathcal{D}} = \{\overline{x}_1, \overline{x}_2, ..., \overline{x}_m\}$ denote the set of test examples. For each $\overline{\mathbf{x}}_s \in \overline{\mathcal{D}}$, we construct its localized SVM model by solving the following optimization problem:

$$\begin{aligned} (3.5) \qquad \min_{\mathbf{w}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C \sum_{i=1}^{n} \sigma(\overline{\mathbf{x}}_s, \mathbf{x}_i) \xi_i \\ \text{s. t} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, 2, \ldots, n \end{aligned}$$

The solution to (3.5) identifies the decision surface as well as the local neighborhood of the test example. The function σ penalizes training examples that are located far away from the test example. As a result, the classification of the test example depends only on the support vectors in its local neighborhood.

To further appreciate the role of the weight function, consider the dual form of (3.5):

$$\begin{aligned} (3.6) \qquad \max_{\alpha_1 \cdots \alpha_n} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s. t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C\sigma(\overline{\mathbf{x}}_s, \mathbf{x}_i), \ i = 1, 2, \ldots, n \end{aligned}$$

Compared to (2.3), the only difference between LSVM and nonlinear SVM is that the constraint on the upper bound for α_i has been modified from C to $C\sigma(\overline{\mathbf{x}}_s, \mathbf{x}_i)$. Such modification has the following two effects: (1) It reduces the impact of far away support vectors, (2) Non-support vectors of the nonlinear SVM may become support vectors of LSVM.

Note that the KNN-SVM method [10] is a special case of our LSVM framework. If σ produces a continuous value output, our LSVM framework is called *Soft Localized Support Vector Machine* (**SLSVM**). Conversely, if σ produces a binary value output (0 or 1), it is called *Hard Localized Support Vector Machine* (**HLSVM**). For HLSVM, the upper bound for α_i is constrained to C, which is equivalent to the optimization problem for KNN-SVM. Finally, we use a linear kernel function for both HLSVM and SLSVM.

4 Profile Support Vector Machine (PSVM)

Since LSVM builds a separate model for each test example, it is computationally expensive when the size of the test set is large. PSVM aims to provide a reasonable approximation to LSVM by reducing the number of constructed models via clustering.

To understand the intuition behind PSVM, let $\vec{\sigma}_s = [\sigma(\overline{\mathbf{x}}_s, \mathbf{x}_1), \ldots, \sigma(\overline{\mathbf{x}}_s, \mathbf{x}_n)]^T$ denote a column vector of similarities between a test example $\overline{\mathbf{x}}_s$ to each training example x_i ($\forall i \in \{1, \cdots, n\}$). From (3.6), notice

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

that the local optimization problem to be solved for each test example $\overline{\mathbf{x}}_s$ is almost identical, except for the upper bound constraint on α_i, which depends on $\sigma(\overline{\mathbf{x}}_s, \mathbf{x}_i)$. Since α_i determines whether x_i is a support vector, we expect test examples with similar $\vec{\sigma}_s$ to share many common support vectors. Therefore, if we can find a set of prototype vectors $\widetilde{\vec{\sigma}}_1, \widetilde{\vec{\sigma}}_2, \ldots, \widetilde{\vec{\sigma}}_\kappa$ such that the similarity vector of each test example is closely approximated by one of the κ prototypes, we need to build only κ linear SVM models (instead of building a separate model for each test example).

4.1 Supervised Clustering for PSVM

Let Σ be an $n \times m$ weight matrix, where n is the training set size, m is the test set size, and the j-th column of the matrix corresponds to the similarity vector $\vec{\sigma}_j$. Our clustering approach is equivalent to approximating Σ by the product of two lower rank matrices $\Lambda = [\lambda]_{n\times\kappa}$ and $\Gamma = [\gamma]_{\kappa\times m}$. The jth column of Λ denote the membership of each training example in cluster j, whereas the ith row of Γ denote the membership of each test example in cluster i.

Our clustering task is somewhat different than conventional unsupervised clustering. First, the data matrix to be clustered is Σ, which contains the similarities between the training and test examples. Second, conventional clustering methods consider only the proximity between examples and often end up grouping training examples from the same class into the same cluster. Because such clusters tend to be pure, their induced models are trivial. Therefore, the clustering criterion must be modified to ensure each cluster contains training examples from all classes.

In this paper, we propose the "**MagKmeans**" algorithm, which modifies the clustering criterion of the k-means algorithm to incorporate the class distribution of training examples within each cluster. The data to be clustered consists of two parts: (1) the matrix Σ and (2) the class label vector $Y = (y_1, \ldots, y_n)^\top$ of the training examples. The objective function for MagKmeans is:

$$\min_{Z,C} \sum_{j=1}^{\kappa}\sum_{i=1}^{n} Z_{i,j}\|X_i - C_j\|_2^2 + R\sum_{j=1}^{\kappa}\left|\sum_{i=1}^{n} Z_{i,j}Y_i\right|,$$

where X_i^T is the i-th row vector in Σ, C_j is the centroid of the jth cluster, Y_i is an element of the vector Y, $R > 0$ is a scaling parameter, and Z is the cluster membership matrix, whose (i,j)-th element is one if the ith training example is assigned to the jth cluster, and zero otherwise. Note that the first term in the objective function is identical to the cluster cohesion criterion used by regular k-means. Minimizing this term would lead to compact clusters. The second term in the objective function measures the class imbalance within the clusters. This term is minimized when every cluster contains equal number of positive and negative examples. Minimizing this term enforces the requirement that the class distribution within each cluster must be balanced.

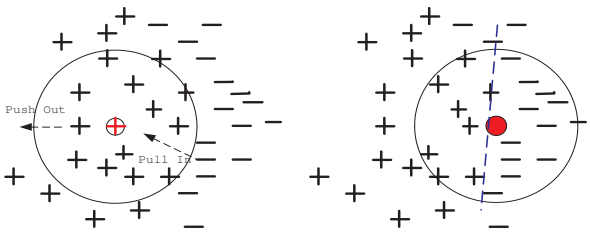

Figure 1: An illustration of the MagKmeans clustering algorithm

Our algorithm iteratively performs the following two steps to optimize the clustering objective function. First, we compute the cluster membership matrix Z by fixing the centroid C_j for all j. Next, we compute the centroid C_j by fixing the cluster memberships Z. These steps are repeated until the algorithm converges to a local minimum.

When C_j is fixed, Z can be computed efficiently using linear programming. To do this, we first transform the original optimization problem into the following form using κ slack variables t_j $(j = 1, \cdots, \kappa)$:

$$\begin{aligned}
\min_{Z_{i,j}} \quad & \Big(\sum_{j=1}^{\kappa}\sum_{i=1}^{n} Z_{i,j}(X_i - C_j)^2 + R\sum_{j=1}^{\kappa} t_j\Big) \\
\text{s. t.} \quad & -t_j \le \sum_{i=1}^{n} Z_{i,j}Y_i \le t_j \\
& t_j > 0,\ 0 \le Z_{i,j} \le 1 \\
& \sum_{j=1}^{\kappa} Z_{i,j} = 1
\end{aligned}$$

When the cluster membership matrix Z is fixed, the following equation is used to update each centroid:

$$C_j = \frac{\sum_{i=1}^{n} Z_{i,j}X_i}{\sum_{i=1}^{n} Z_{i,j}}$$

Figure 1 illustrates how the MagKmeans algorithm works. The initial cluster (the left figure) contains only positive examples. As the algorithm progresses, some positive examples are expelled from the cluster while some negative examples are absorbed into the cluster (the right figure). By ensuring that the cluster has almost equal representation from each class, one can then build a linear SVM from the training examples.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

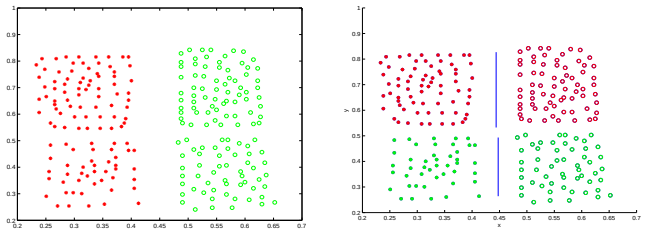

Figure 2: Regular Kmeans clustering result (left). MagKmeans clustering result (right). Clusters are represented by different colors.

Finally, we illustrate the difference between the resulting clusters produced by regular k-means and MagKmeans using the synthetic data set shown in Figure 2. Based on the cluster cohesion criterion, the k-means algorithm produces clusters that correspond to each class, whereas MagKmeans generates clusters with representatives from both classes.

4.2 Model Building and Testing

After applying the MagKmeans algorithm, we obtain a prototype matrix $\Lambda = [\lambda]_{n\times\kappa}$, where each element $\lambda_{i,j} = Z_{i,j}$, i.e., it represents the membership of each training example x_i in cluster j. We then build a separate linear SVM model for each cluster by solving the following optimization problem:

$$\begin{aligned} \max_{\widetilde{\alpha}} \quad & \sum_{i=1}^{n} \widetilde{\alpha}_i - \frac{1}{2}\sum_{i,j=1}^{n} \widetilde{\alpha}_i\widetilde{\alpha}_j y_i y_j \mathbf{x}_i\mathbf{x}_j \\ \text{s. t.} \quad & \sum_{i=1}^{n} \widetilde{\alpha}_i y_i = 0 \\ & 0 \le \widetilde{\alpha}_i \le C\lambda_{i,k},\ i = 1, 2, \ldots, n \end{aligned}$$

Since $\lambda_{i,k} \in \{0, 1\}$, this is equivalent to building a linear SVM using only the training examples assigned to the cluster k. The models obtained from the clusters form a piecewise decision surface consisting of κ hyperplanes.

Let C denote the $\kappa \times m$ centroid matrix obtained by the MagKmeans algorithm. During the testing step, we need to determine which local model should be used to classify a test example. Since the (i, j)-th element of the centroid matrix indicates the similarity between a test example $\overline{\mathbf{x}}_j$ to cluster i, we assign the test example to the cluster with highest similarity. More specifically, we construct the lower rank matrix Γ by setting:

$$\gamma_{i,j} = \begin{cases} 1 & \text{if } i = \arg\max_i C_{i,j} \\ 0 & \text{otherwise} \end{cases}$$

The class label of the test example $\overline{\mathbf{x}}_j$ is determined by applying the corresponding linear SVM model of its assigned cluster.

In short, PSVM uses the MagKmeans algorithm to identify the κ prototypes. It then trains a local SVM model for each prototype. In our experiments, we found that the number of clusters κ tends to be much smaller than m and n. We found that this criterion usually delivers satisfactory performance. Since κ is generally much smaller than the number of training examples n and the number of test examples m, PSVM reduces the computational cost of LSVM significantly. R is another parameter in PSVM that needs to be determined. In our work, we empirically set it to $1/\kappa$ times the diameter of the data set.

5 Experimental Evaluation

We have conducted extensive experiments to evaluate the performance of our proposed algorithms in comparison to KNN and nonlinear SVM algorithms. To implement the proposed LSVM algorithm, we modified the C++ code of the LIBSVM tool developed by Chang and Lin [3] to use $C\sigma$ as its upper bound constraint for α instead of C. For PSVM, we have implemented the MagKmeans algorithm to cluster the weight matrix Σ into κ prototypes. All our experiments were conducted on a Windows XP machine with 3.0GHz CPU and 1.0GB RAM.

5.1 Comparisons of Support Vectors and Decision Boundaries

In this experiment, we illustrate the piecewise linear decision boundaries formed by PSVM. The top panel of Figure 3 shows the data distributions for two synthetic data sets with nonlinear decision boundaries. The bottom panel illustrates their corresponding decision boundaries generated using PSVM. For the first data set in the left of Figure 3, the horse-shoe shaped decision boundary is now approximated by 11 piecewise linear decision boundaries. The spiral-shaped decision boundary of the data set in the right figure is also approximated by 11 piecewise linear decision boundaries. In short, the results of this experiment show the ability of PSVM to fit a complex decision boundary using multiple piecewise linear segments.

5.2 Performance Comparison

We use eight data sets from the UCI repository [8] to compare the performances of HLSVM, SLSVM, and PSVM against KNN and nonlinear SVM in terms of their accuracy and computational time. Some of the data sets such as "Breast", "Glass", "Iris", "KDDcup IDS", "Physics", "Yeast", "Robot" and "Forest" are

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

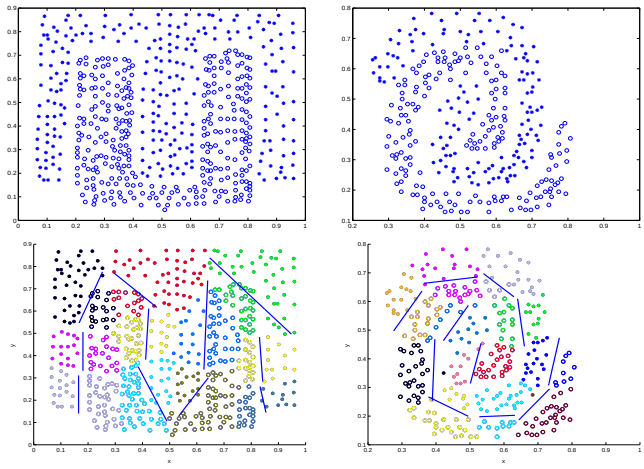

Figure 3: The top panel shows two synthetic data sets. Each data set is comprised of two classes marked by $\circ$ and $*$. The bottom panel shows the decision boundaries generated by PSVM. Each color represents a different cluster produced by the MagKmeans algorithm.

Table 1: Classification accuracies (%) for SVM, KNN, HLSVM (KNN-SVM), SLSVM , and PSVM on the UCI datasets.

Data	SVM	KNN	HLSVM	SLSVM	PSVM
Breast	94.57	95.70	95.42	96.85	96.52
Glass	64.33	54.30	62.67	66.39	66.91
Iris	92.75	89.75	74.71	96.42	97.06
KDDCup	98.29	94.04	98.13	99.67	99.22
Physics	82.23	66.37	83.77	86.72	85.77
Yeast	92.31	93.36	94.62	96.00	95.83
Robot	86.51	84.54	85.45	89.01	90.21
Covtype	86.21	67.40	73.33	90.22	90.39

multi-class prediction problems. Since our proposed algorithm is designed for binary classification problems, we divide the classes for these data sets into two groups and relabel one group as the positive class and the other as the negative class. For some of the larger data sets such as "KDDcup IDS", "Physics", and "Robot", we randomly sample 600 records from each class to form the data sets. The attributes for all the eight data sets are normalized based on the maximum value of the attribute. The parameters of the classification algorithm, i.e. the K in KNN, C in SVM, bandwidth λ in RBF kernel and κ in PSVM are determined by 10-fold cross validation on the training set.

5.2.1 Accuracy Comparison The experimental results reported in this study are obtained based on applying a 5-fold cross validation on the data sets. To make the problem more challenging, we use 1/5 of the entire data set for training and the remaining 4/5 for testing. Each experiment is repeated ten times and the accuracy reported is obtained by averaging the results over ten trials. Table 1 summarizes the results of our experiments. First, observe that, for most data sets, nonlinear SVM outperforms the KNN algorithm. The most noticeable case is the "Covtype" data set, for which the accuracy of SVM is 86.21% while the accuracy of KNN is only 67.40%. Second, observe that HLSVM, which is the hard version of LSVM, fails to improve the accuracy over nonlinear SVM. In fact, the performance of HLSVM degrades significantly on data sets such as "Glass", "Iris", and "Covtype". The most noticeable case is "Iris", where the classification accuracy drops from 92.75% to 74.71% when using HLSVM instead of nonlinear SVM. One possible explanation for the poor performance of HLSVM is the difficulty of choosing the right number of nearest neighbors (K) when the number of training examples is small.

We observe that the SLSVM algorithm consistently outperforms nonlinear SVM for all the data sets. In fact, with the exception of "KDD Cup", the difference in classification accuracies of SLSVM and nonlinear SVM is found to be statistically significant according to Student's t-test. This should not come as a surprise because nonlinear SVM is a special case of SLSVM by setting the kernel width to ∞. Finally, we observe that PSVM, which is an efficient implementation of LSVM, achieves comparable accuracy as SLSVM and outperforms nonlinear SVM for all the data sets.

5.2.2 Runtime Comparison The purpose of this experiment is to evaluate the efficiency of LSVM and PSVM. Recall from Section 3 that the main drawback of LSVM is its high computational cost since a separate LSVM model must be trained for each test example. PSVM alleviates this problem by grouping the training examples into a small number of clusters and building a linear SVM model for each cluster. Figure 4 shows the runtime comparison (in seconds) among the different classification methods using the "Physics" data set. To evaluate the performance, we choose the two largest classes of the data set and randomly sample 300 records from each class to form the training set. We then apply LSVM and PSVM to the data set, while varying the number of test examples from 600 to 4800. The left panel of Figure 4 shows the overall runtime for each method, which includes the training and testing times. Notice that the computational times for both HLSVM and SLSVM grows rapidly as the test set size increases. In contrast, the runtime of PSVM grows more gradually as the number of test examples increases. The right top

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited

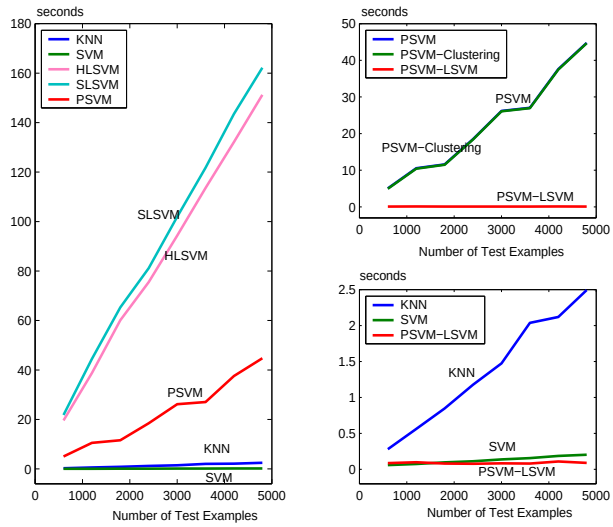

Figure 4: The left panel shows the overall runtime of SVM, KNN, HLSVM, SLSVM, and PSVM. The right top panel shows the runtime of PSVM during clustering. The right bottom panel shows the runtime of each method when predicting the class labels of test examples.

panel shows a more detailed runtime analysis for PSVM. It compares the time needed to perform the MagKmeans clustering (represented by the PSVM-clustering line) and the time needed to build the LSVMs of different clusters (represented by the PSVM-LSVM line). The result suggests that PSVM spends the majority of its time clustering the training data. Furthermore, if the clustering time of PSVM is discounted, then the time needed to train the multiple linear SVMs of the clusters as well as to apply the models to the test examples is faster than training and testing times for nonlinear SVM, as shown in the right bottom panel of Figure 4.

To summarize, the SLSVM algorithm generally outperforms both SVM and KNN but at the expense of incurring a much higher computational cost. PSVM is able to improve its computational efficiency, while achieving comparable accuracy as the SLSVM algorithm.

6 Conclusion and Future Work

In this paper, we proposed a framework for Localized Support Vector Machine, which utilizes a weight function to constrain the maximum weights that can be assigned to training examples based on their similarity to the test example. We tested our LSVM framework on a number of data sets and showed that its soft version (SLSVM) outperforms both KNN and nonlinear SVM in terms of model accuracy. Nevertheless, SLSVM is computational expensive since it requires training a separate model for each test example. To overcome this problem, we propose an approximation algorithm called PSVM, which reduces the training time by extracting a small number of clusters and building linear SVM models only for the clusters. Our analysis further showed that PSVM outperforms both KNN and SVM and is comparable in accuracy but much more computationally efficient than LSVM. In the future, we plan to expand our framework to multi-class problems. For MagKmeans, this can be accomplished by modifying the clustering criterion to maximize the cluster impurity. We will also investigate the possibility of using other clustering algorithms such as spectral clustering to further enhance the results.

References

[1] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, April 1997.

[2] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. In *Knowledge Discovery and Data Mining*, page 2(2), 1998.

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

[4] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions in Information Theory*, pages IT–13,21–27, 1967.

[5] S. R. Gunn. Support vector machines for classification and regression. Technical report, University of Southampton, 1997.

[6] K. Hechenbichler and K. Schliep. Weighted k-nearest-neighbor techniques and ordinal classification. *Discussion Paper 399, SFB 386*, 2006.

[7] T. Joachims. Transductive inference for text classification using support vector machines PRODIGY. In *International Conference on Machine Learning*, San Francisco, 1999. Morgan Kaufmann.

[8] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, http://www.ics.uci.edu/∼mlearn/mlrepository., 1998.

[9] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. *Advances in Neural Information Processing Systems 12*, pages pp. 547–553,MIT Press, 2000.

[10] H. Zhang, A. C. Berg, M. Maire, and J. Malik. Svm-knn: discriminative nearest neighbor for visual object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited