Lazy SVM: A Local Learning Algorithm for SVM

Jiun-Hung Chen and Adrienne X. Wang
jhchen@cs
axwang@cs

Abstract

In this paper, we propose a lazy learning algorithm on the support vector machines. Instead of compiling the entire training data set into a SVM model without considering any testing data, we store the training data and wait until a testing data set is presented, then construct a local SVM to classify a testing data point. However, a SVM model is developed for every testing data based on its nearest neighbors in the training data set and this model is only used to classify this testing data. We analyze two approaches to speed up the total training time. Our proposed lazy learning algorithm is tested on several classification and regression benchmark datasets and experimental results show that lazy learning algorithms with high efficiency can achieve comparable accuracy. For example, for the USPS database, lazy SVMs which use only one thousandth of training data can achieve a lower error rate than that of eager SVMs.

Introduction

Support vector machines (SVMs) (Vapnik 1995)(Muller *et al.* 2001) are very successful classifiers in many applications such as face detection (Osuna, Freund, & Girosi 1997), object recognition (Pontil & Verri 1998), and handwritten digit recognition (Vapnik 1995). Conventional SVMs are trained on the entire training data set using eager learning algorithms. An eager learner creates a global approximation for a problem and hopes to generalize before seeing any query points. Other examples of eager learning includes decision trees and back-propagation neural networks. In contrast, a lazy learner waits for query before generalizing and can create a local approximation for a problem. For example,k-nearest neighbors and weighted linear regression. In fact,if an eager learner and a lazy learner both use the same hypothesis space (i.e., linear functions), a lazy learner can represent a more complex function than an eager learner.

We propose Lazy SVMs, a local learning algorithm (Bottou & Vapnik 1992) to train support vector machines. *For each testing pattern, select the few training examples located in the vicinity of the testing pattern, train a SVM with only these few examples, and apply the resulting SVM to the testing pattern.* In (Bottou & Vapnik 1992), their algorithm can achieve a low error rate on a digit recognition database. On the same database, our algorithm which uses only one thousandth of training data can achieve a lower error rate than an eager SVM which uses all training data.

However, lazy learning can be very slow because a learning is performed for each query. On the other hand, with a huge set of data, the training process of eager learning algorithms may become very slow. It is not worth the time if a much smaller testing data set is presented. In addition, training data are rarely distributed evenly in the space. Global estimation may have even worse performance than a local estimation.

This paper attempts to demonstrate that SVMs trained with a much smaller training data set can achieve very high accuracy, comparable to those with eager learning algorithms, while the computation time on each training process is greatly reduced. However, a SVM trained on the entire training data set, although time consuming at the beginning, can be used later. We notice that analyzing each individual data increases the total training time since a new SVM is created each time a testing data is presented. When the testing data set is big, the increase can be overwhelming. Two algorithms are proposed to solve this dilemma and to improve efficiencies. Instead of training a SVM for each data, we relax the lazy learning algorithm a little bit. One SVM is trained for the entire testing data set. In the first approach, we pick training data by selecting the data which has shortest average distance to the testing data set. Experiments show that this algorithm can cut the total training time with a big factor, but has very bad performance. Another idea is to pick the nearest neighbors of all testing data, then train a SVM using this union. We get successful results on the second approach.

The remaining of this paper is organized as follows. Section 2 reviews the SVM. Section 3 proposes lazy SVMs. Then, we describe the experiment design and speedup algorithms in Section 4. Experimental results and discussions are shown in Section 5. Finally, conclusions and future work are given in Section 6.

SVMs

Consider a two-class classification problem. Let $\Omega_T = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n) | x_i \in R^d, y_i \in \{-1, 1\}, i = 1, 2, ..., n\}$ be a set of n input-output training data pairs, where R^d space is referred to as the input space. A SVM (Vapnik 1995)(Muller *et al.* 2001) first projects the input vectors onto some feature space by a non-linear function ϕ and then finds a linear separating hyperplane in the feature space H, that is, $w^t x + b = 0$ where $w \in H$ and $b \in R$. SVM classifiers are obtained by solving the objective function

$$min_{w,b,\epsilon_i} \frac{||w||^2}{2} + C \sum_{i=1}^{n} \epsilon_i \tag{1}$$

subject to

$$y_i(w^t \phi(x_i)) + b) \geq 1 - \epsilon_i \text{ for all } i = 1, \ldots, n$$
$$\epsilon_i \geq 0 \text{ for all } i = 1, \ldots, n$$

where the constant $C > 0$ determines the tradeoff between margin maximization and training error minimization. Figure 1 gives a diagram of the training process.

After solving the above optimization problem using Lagrangian techniques, we can obtain the classification rule of SVM

$$sgn(\sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + b), \tag{2}$$

where $K(x, x_i) = \phi(x)^T \phi(x_i)$ and α_is are Lagrange multipliers. Note that $K(.,.)$ is a Mercer's kernel (Vapnik 1995) and commonly used Mercer's kernels include Gaussian RBF, polynomial functions, and sigmoidal functions. For example, Gaussian RBFs are defined as

$$K(x, x') = \exp(-\frac{||x - x'||^2}{2\sigma^2}) \tag{3}$$

where $\sigma \in R$ is a kernel parameter. Note that $||x - x'||^2$ is typically defined as

$$||x - x'||^2 = (\sum_{j=1}^{d} |x_j - x'_j|^2) \tag{4}$$

To deal with multi-class classification problem, we use a standard one-against-one strategy in this paper. That is, for each pair of two different labels, we train a binary classifier to determine the label of a testing data. We then take the majority voting among these classifiers to determine the label of a testing data. Other ways to extend SVMs to multi-class classification problems and their comparisons can be found in (Hsu & Lin 2002).

An extension of SVM to ϵ-support vector regression (SVR) can be found in (Vapnik 1995). The regularized risk functional function used in ϵ-SVR is

$$\frac{||w||^2}{2} + C \sum_{i=1}^{n} |y_i - f(x_i)|_\epsilon \tag{5}$$

where $|y - f(x)|_\epsilon = \max\{0, |y - f(x)| - \epsilon\}$ which is so-called ϵ-insensitive loss function.

SVMs with Lazy Learning

In lazy learning algorithms, SVMs are constructed with the knowledge of testing data set

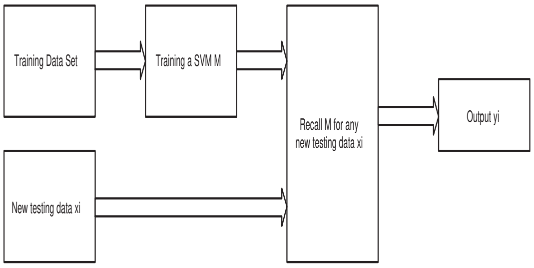

Figure 1: A SVM is trained based on the entire training data set and used to classify all testing data points.

$\Omega_S = \{(x_1{}^*, y_1{}^*, (x_2{}^*, y_2{}^*), ..., (x_m{}^*, y_m{}^*)| x_i{}^* \in R^d, y_i{}^* \in \{-1, 1\}, i = 1, 2, ..., m\}$.

To find the labeling $y_i{}^*$ of the testing data $x_i{}^*$, the hyperplane $w^t x + b = 0$ should separate Ω_{T_i}, which contains k nearest neighbors of $x_i{}^*$ in Ω_T.

The SVM is obtained by solving the objective function

$$min_{w,b,\epsilon_i} \frac{||w||^2}{2} + C \sum_{\Omega_{T_i}} \epsilon_i \tag{6}$$

subject to

$$y_i(w^t \phi(x_i)) + b) \geq 1 - \epsilon_i \text{ for all } i \text{ where } x_i \in \Omega_{T_i} \tag{7}$$
$$\epsilon_i \geq 0 \text{ for all } i \text{ where } x_i \in \Omega_{T_i} \tag{8}$$
$$\tag{9}$$

We find the k nearest neighbors of x_i* using different distance functions. Figure 2 gives a diagram for the training process of a lazy SVM.

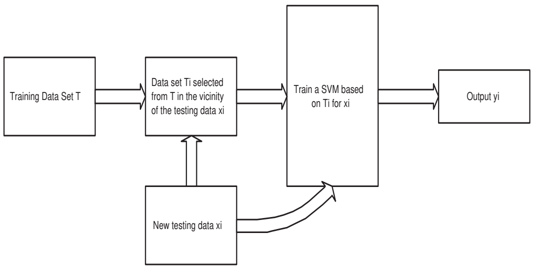

Figure 2: A SVM is trained on a subset of the training data set for each testing data point

Experiments

1. **Lazy learning vs. Eager learning** We test the proposed lazy learning algorithm on several benchmark datasets. The experiment is carried out through 10-fold cross-validation. For each benchmark dataset, we divide it into 10 subsets. Each time we leave out 1 of the subset as testing data set Ω_S, and use the other 9 subsets as training data set Ω_T. For each testing data point $x_i{}^*$ in Ω_S, we select its k nearest neighbors from Ω_T and form a new set Ω_{T_i}. A SVM model is constructed based on Ω_{T_i}. Then we

test the correctness of this SVM by comparing the classification result and $y_i{}^*$. The same experiment is repeated with different k.
For comparison, a SVM is trained on the entire training data set Ω_T. We compare the average error rate of the SVMs trained on each Ω_{T_i} and the error rate of the SVM trained on Ω_T. Figure 3 illustrates the algorithm we use to select the k nearest neighbors.

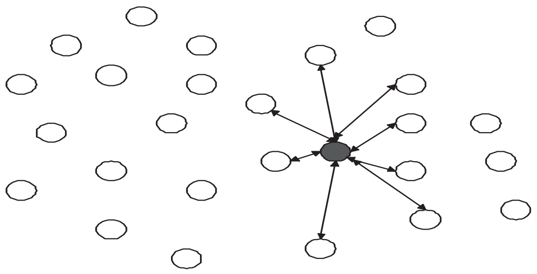

Figure 3: The k nearest neighbors are selected to train a SVM for each testing data point (White: Training data. Black: Testing data)

Time complexity

Let $|\Omega_T| = n$, $|\Omega_S| = m$, $S =$ number of support vectors, $R = \frac{\text{\#of selected data points}}{\text{\#of total training data points}}$ where $k = Rn$
An eager learner creates a global estimation model which can be used repeatedly. Its training process takes time $O(n^3)$ to solve the quadratic programming optimization problem. In the testing phase, each testing data needs to consult all S support vectors, which are the closest points to the separator. Classifying m testing data takes time $O(mS)$. Note that there are usually many fewer support vectors than data points.(Russell & Norvig 1995)
A lazy learner doesn't do any training offline until a testing data set is presented. For each testing data, it selects Rn data from Ω_T and takes $O((Rn)^3)$ to train a SVM. Each SVM can have at most Rn support vectors, so the classification part takes $O(Rn)$ for each testing data. In addition, we need to consider the computation time of selecting the $k = Rn$ nearest neighbors, which is $O(n)$ since we compute $d(x_i, x_i{}^*) \forall x_i$. In summary, a lazy learner takes time $O(mRn + m(Rn)^3 + mn)$ to process a testing data set Ω_S. We summarize the time complexity of eager SVMs and lazy SVMs in Table 1.

Table 1: Time complexity: eager learning vs. lazy learning

	Test	Train
eager learning	$O(mS)$	$O(n^3)$
lazy learning	$O(mRn + (Rn)^3 + mn)$	$O(1)$

2. **Speed up the training process** Analyzing each individual testing data may cause the total computation time to go up dramatically when given a big testing data set. Let $|\Omega_S| = m$, we need m SVMs in lazy learning, while there is only 1 SVM built offline and used repeatedly for the eager learner. We propose two approaches to reduce the number of SVMs constructed in the lazy learning algorithm to improve the efficiency. In this section, instead of building a SVM for each testing data point, we build one SVM for the entire testing data set and test its error rate.

(a) We define the distance function to be the average distance from one data point $x_i \in \Omega_T$ to each data point $x_i{}^* \in \Omega_S$.

$$\sum_{\forall x_i{}^* \in \Omega_S} d(x_i, x_i{}^*) \tag{10}$$

The k nearest neighbors using this newly defined distance function will be selected and put into the subset $\Omega_{T'}$. A SVM is trained on this smaller training data set $\Omega_{T'}$, and is used to classify all the testing data point. Figure 4 is an example of the algorithm we described above.

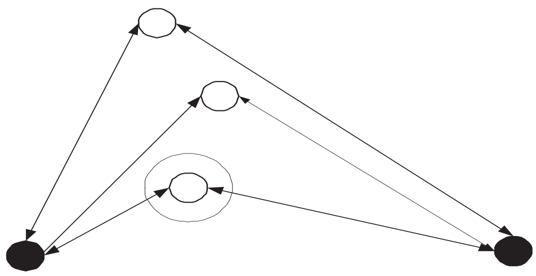

Figure 4: A SVM is trained on a subset which contains k nearest neighbors with shortest average distance to all testing data and used to classify all testing data points (White: Training data. Black: Testing data)

Time complexity

Compared with the first lazy algorithm, training the SVM takes time $O((Rn)^3)$ instead of $O(m(Rn)^3)$ since only one SVM is constructed for this testing data set. This algorithm doesn't improve the classification part and the selection part. This algorithm trains a SVM on a very small set of data, helps to get rid of the overhead of training a new SVM for each testing data point.

(b) Another approach is to pick k nearest neighbors of all testing data, and train a SVM using this union.

$$\bigcup \Omega_{T_i} \tag{11}$$

Again, a SVM is trained on this smaller training data set, and is used to classify all the testing data point. Note that in most cases the resulting training data set $\bigcup \Omega_{T_i}$ in this approach has more items than the one we build in the first approach. In the worst case, we may end up getting a set of n items, with n being the total number of items in the original training data set Ω_T. Figure 5 is an example of the second speedup algorithm.

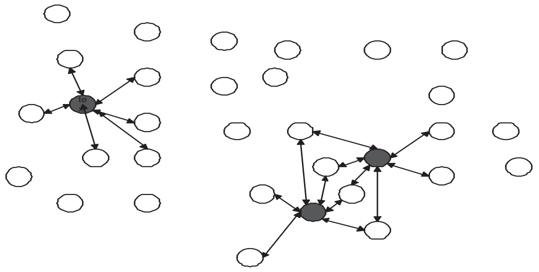

Figure 5: The union of k nearest neighbors of each testing data are selected to train a SVM for all testing data (White: Training data. Black: Testing data)

Time complexity

Define $k =$ number of nearest neighbors selected. Training this SVM takes time $O((km)^3))$ in the worst case where there are m disjoint sets of k nearest neighbors. The classification part takes time $O(km^2)$. Selecting the k nearest neighbors takes times $O(mn)$ for all data points. This algorithm has great advantage when the training data set is much bigger than the testing data set since m is the dominating factor in its time complexity. We end this section with Table 2 summarizing the time complexity of all the algorithms we described.

Table 2: Time complexity: eager learning vs. lazy learning vs. speed ups

	Test	Train
eager learning	$O(mS)$	$O(n^3)$
lazy learning	$O(mRn + (Rn)^3 + mn)$	$O(1)$
first approach	$O(Rn + (Rn)^3 + mn)$	$O(1)$
second approach	$O(km^2 + (km)^3 + mn)$	$O(1)$

Experimental Results and Discussions

Eager SVMs vs. Lazy SVMs

We use several benchmark datasets to test the performance of proposed method. These benchmarks datasets include add10, boston housing, cpu activities, fourclass, glass, heart, iris, vowel and wine [1] and their characteristics are summarized in Table 3. The LIBSVM [2] software is used to train SVMs for these problems. In addition, we scale each dimension of input data to -1 to 1 before we train SVMs on input data.

In our experiments, we use average ten-fold cross-validation error rates as the performance measure. For fair comparisons, model selection was performed for finding the best sets of parameters (σ, C) for Gaussion RBF kernels within a given range. The parameters R, σ and C were investigated within $[0.05, 1]$, $[2^{-10}, 2^{10}]$ and $[2^{-10}, 2^{10}]$ in our model selection process, respectively. Similar model selection is also performed for polynomial kernels.

We compare the performance of the SVM using the lazy learning algorithm, the two speedup algorithms, the KNN method. Please see Figure 6-15 for the results. The last point on the lazy SVM represents the performance of an eager learner since this SVM is trained on the entire training data set.

We summarize the performance of all lazy SVMs in Table 4 and 5. For these two tables, we can see that lazy SVMs give better performance than eager SVMs. The percentage of the training data utilized vary from 5% to 95%. There are some datasets which use few training data points, and achieve high correct rates, for example, the datasets fourclass, cpu and vowel. In these datasets, our lazy learner has a comparable time complexity.

Table 3: Benchmark data set. Note that - in # classes indicates regression problems.

	#data	#attributes	#classes
add10	327	10	-
boston	506	13	-
cpu	274	21	-
fourclass	862	2	2
glass	214	9	7
heart	270	13	2
iris	150	4	3
vowel	528	10	11
wine	178	13	3

KNN vs. lazy SVMs

Our experiments also include a test case on the k-nearest-neighbor method. KNN and lazy SVMs both try to classify data points using a locally estimated separator. We summarize the performance of the nearest neighbor methods in Table 6 and 7. From comparisons between Tables 4 and 5 and Tables 6 and 7, we found lazy SVMs perform better than the k-nearest-neighbor method. As our assumption, the separator can be locally estimated. However, how to determine the vicinity of data vectors through measuring the distance between them is a hard problem. Usually, if two data points are close to each other, then it's more likely that they belong to the same class. In the original input space, the separator tends to be non-linear in most cases. So the intuition doesn't hold most of the time. On the contrary, kernel machines which project the data points into a space of high dimension can obtain a linear separator. Our experiment which compares the performance of KNN and our lazy SVM verify our claim.

Speedup performance

We measured the performance of the proposed speedup algorithms. We show the results in Tables 8, 9 and 10. The

[1] These benchmark datasets are available at UCI Repository of machine learning databases http://www.ics.uci.edu/∼mlearn/MLRepository.html or http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/

[2] Available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

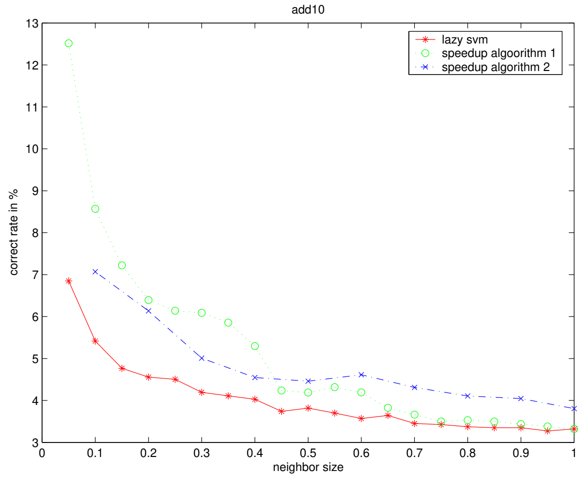

Figure 6: Performance comparisons on the add10 dataset.

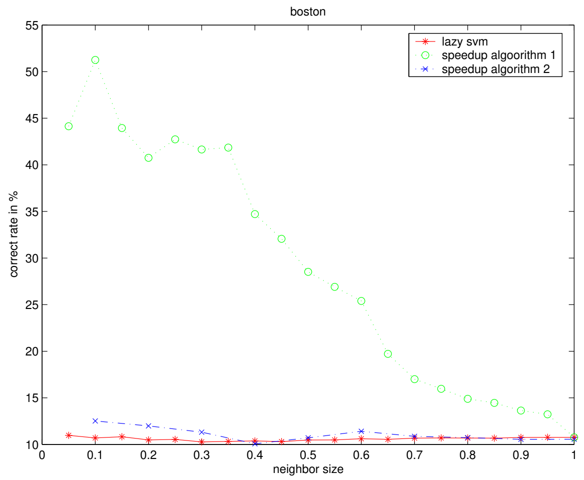

Figure 7: Performance comparisons on the boston dataset.

second speedup algorithm has similar performance as the lazy learning algorithm with a reduced time complexity. The first speedup algorithm is not as good.

Polynomial kernel vs. Guassian RBF kernel

We did all experiments using polynomial kernels and Guassian RBF kernels. The results show that Guassian RBF kernels perform better than polynomial kernels. Please refer to Table 4 vs. Table 5 and Table 9 vs. Table 10.

Case Study: USPS Handwritten Digit Recognition

As a real application, we test the lazy SVMs on USPS handwritten digit recognition database. The USPS database contains 7291 handwritten digits for training and 2007 handwritten digits for testing. It is well known that the USPS test set is rather difficult and the human error rate is 2.5% (Bromley & Sackinger 1991). The error rate of an eager SVM which uses all training data is 4.3% with Gaus-

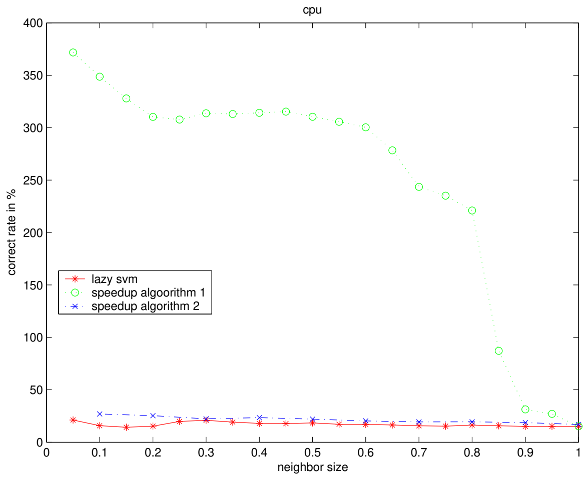

Figure 8: Performance comparisons on the cpu dataset.

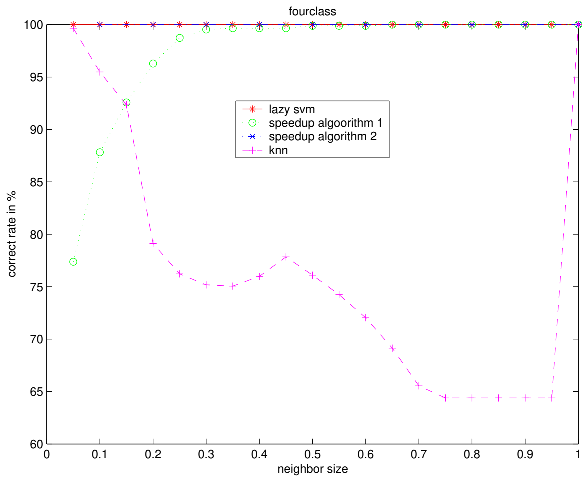

Figure 9: Performance comparisons on the fourclass dataset.

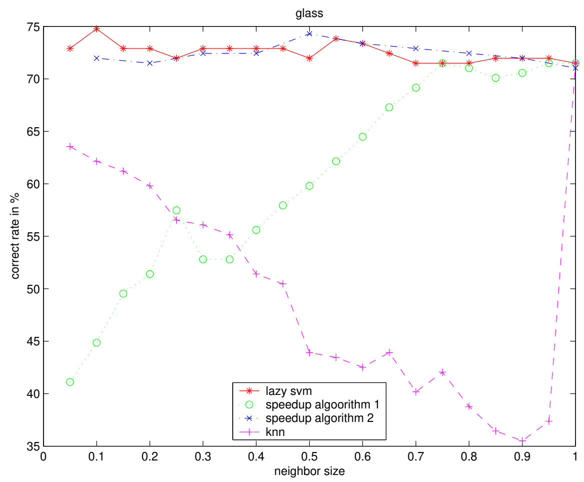

Figure 10: Performance comparisons on the glass dataset.

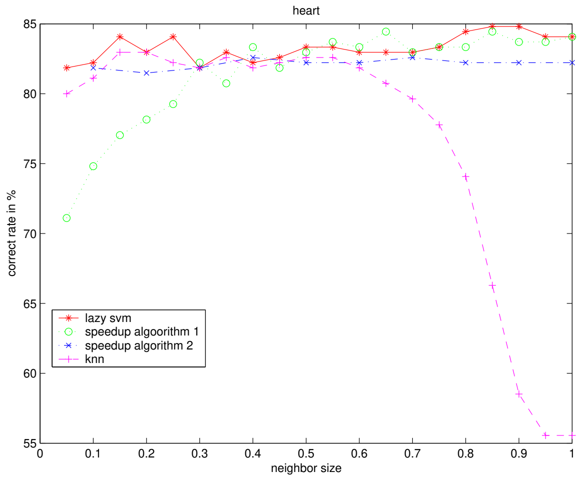

Figure 11: Performance comparisons on the heart dataset.

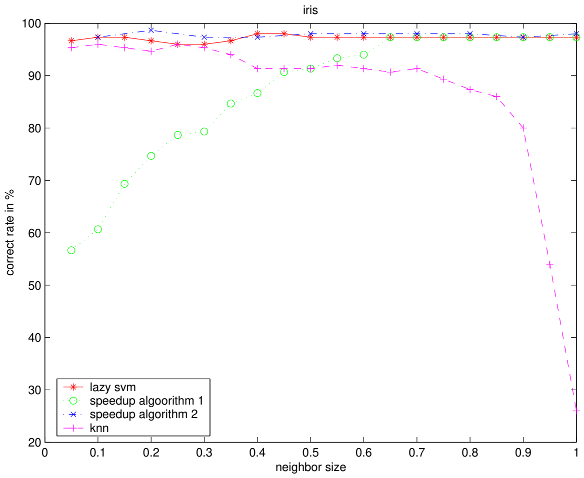

Figure 12: Performance comparisons on the iris dataset.

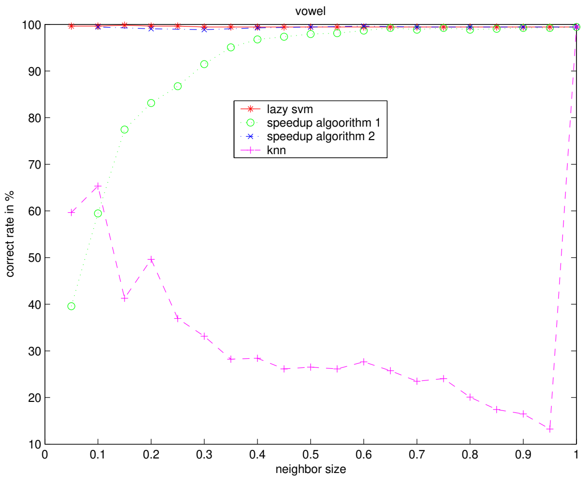

Figure 13: Performance comparisons on the vowel dataset.

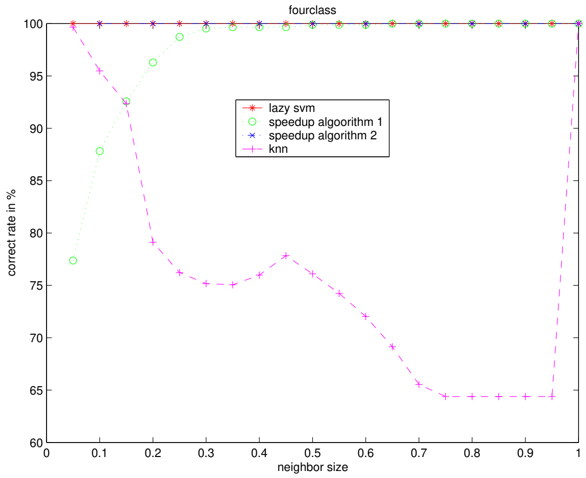

Figure 14: Performance comparisons on the wine dataset.

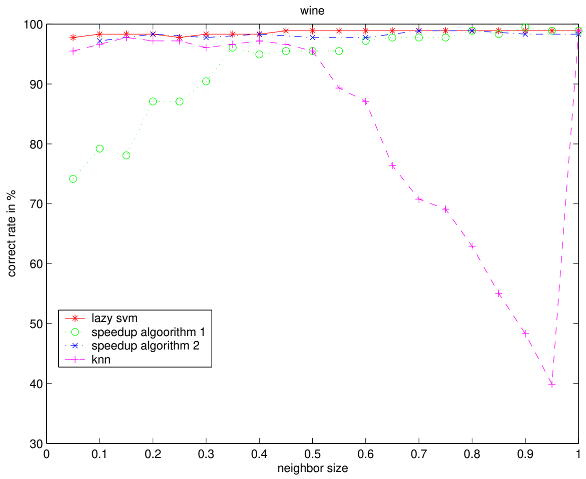

Figure 15: Performance comparisons on the fourclass dataset.

sian RBF kernels (Scholkopf & Smola 2002). In contrast, lazy SVMs which use only one thousandth of training data (7 training data for each testing data) can achieve a lower error rate, 4.13%. Nearest neighbor can achieve 5.9% error rate with the help of additional training data from other sources (Scholkopf & Smola 2002). In this experiment, the time complexity of the eager learner is $O(mS) = 5.35 \times 10^6 C_1$. The time complexity of the lazy learner is $O(mRn + m(Rn)^3 + mn) = 1.54 \times 10^7 C_2$. C_1 and C_2 are some constants. So the lazy learner has better performance using comparable amount of time as the eager learner.

Conclusion and Future Work

In this paper, we propose lazy SVM which is a local learning algorithm for SVMs. From our experimental results, the lazy SVMs perform better than the eager SVMs and the k-nearest neighbor method. For the USPS database, lazy SVMs which use only one thousandth of training data can achieve a lower error rate, 4.13%, than that of eager SVMs. Our proposed speedup algorithms make lazy SVMs have comparable time complexity as eager SVMs on classification.

There are many important future directions. In this paper, we demonstrate the performance that lazy SVMs can potentially achieve. However, how to achieve the best performance (i.e., model selection) for lazy SVMs deserves further research. At the same time, how to determine a good distance measure for finding closest neighbors is another important problem, which can not only improve the performance, but also reduce the algorithms' time complexity. In this paper, we didn't explore the theoretical explanation of the lazy SVMs. It is worth further study.

Acknowledgments

We greatly thank Xu Miao for his valuable help in many directions.

References

Bottou, L., and Vapnik, V. 1992. Local learning algorithms. *Neural Computation* 4(16):888–901.

Bromley, J., and Sackinger, E. 1991. Neural-network and k-nerest-neighbor classifiers. *Technical Report 11359-910819-16TM, ATT*.

Hsu, C.-W., and Lin, C.-J. 2002. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks* 13:415–425.

Muller, K. R.; Mika, S.; Ratsch, G.; Tsuda, K.; ; and Scholkopf, B. 2001. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks* 12:181–201.

Osuna, E.; Freund, R.; and Girosi, F. 1997. Training support vector machines: an application to face detection. *Proceedings of Computer Vision and Pattern Recognition* 130–136.

Pontil, M., and Verri, A. 1998. Support vector machines for 3-d object recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence* 20:637–646.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence A Modern Approach*. New Jersey: Prentice Hall.

Scholkopf, B., and Smola, A. J. 2002. *Learning with kernels -support vector machines, regularization, optimzation and beyond*. Cambridge: The MIT Press.

Vapnik, V. 1995. *The Nature of Statisticl Learning Theory*. New York: Springer.

Table 4: Performances of lazy SVMs on different datasets with Gaussian RBF kernels. For classification problems, correct rates all are shown in %. For regression problems, correct rates are mean square errors. Note that for a dataset, the best result is shown in bold face, and second emphasized. Neighbor size is the percent of training data a testing data uses in the lazy SVM which has the best correct rate. Lazy SVMs give better performance than eager SVMs.

	best correct rate	neighbor size	correct rate with 5% of training data per testing data	Eager SVM
add	**3.2743**	95	6.8509	*3.3216*
boston	**10.2687**	30	10.9722	*10.7647*
cpu	**14.2857**	15	21.1976	*15.2426*
fourclass	**100**	5	**100**	**100**
glass	**74.7664**	10	*72.8972*	71.4953
heart	**84.8148**	85	81.8519	*84.0741*
iris	**98**	40	96.6667	*97.3333*
vowel	**99.8106**	15	*99.6212*	99.4318
wine	**98.8764**	45	*97.7528*	**98.8764**

Table 5: Performances of lazy SVMs on different datasets with polynomial kernels. For classification problems, correct rates all are shown in %. For regression problems, correct rates are mean square errors. Note that for a dataset, the best result is shown in bold face, and second emphasized. Neighbor size is the percent of training data a testing data uses in the lazy SVM which has the best correct rate. Lazy SVMs give better performance than eager SVMs.

	best correct rate	neighbor size	correct rate with 5% of training data per testing data	Eager SVM
fourclass	**100**	5	100	**100**
glass	**75.7009**	40	73.3645	*72.4299*
heart	**84.444**	80	81.8519	*83.7037*
iris	**98.667**	30	96.6667	*98.000*
vowel	**99.6212**	5	99.6212	*98.8636*
wine	**98.3146**	10	95.5056	**98.3146**

Table 6: Performances of KNN on different datasets with Gaussian RBF kernels as a similarity measure. Correct rates all are shown in %. Neighbor size is the percent of training data a testing data uses in KNN which has the best correct rate.

	best correct rate	neighbor size	correct rate with 5% of training data per testing data
fourclass	100	100	99.6520
glass	71.4953	100	63.5514
heart	82.9630	15	80.000
iris	96.000	10	95.3333
vowel	99.4318	100	59.6991
wine	98.8764	100	95.5056

Table 7: Performances of KNN on different datasets with polynominal kernels as a similarity measure. Correct rates all are shown in %. Neighbor size is the percent of training data a testing data uses in KNN which has the best correct rate.

	best correct rate	neighbor size	correct rate with 5% of training data per testing data
fourclass	99.768	100	99.6520
glass	72.4299	100	63.5514
heart	83.7037	100	80.000
iris	98.000	100	96.000
vowel	98.8636	100	58.3333
wine	98.3146	100	95.5056

Table 8: Performances of lazy SVMs using the first speedup algorithm on different datasets with Gaussian RBF kernels. For classification problems, correct rates all are shown in %. For regression problems, correct rates are mean square errors. Note that for a dataset, the best result is shown in bold face, and second emphasized. Neighbor size is the percent of training data a testing data uses in the lazy SVM which has the best correct rate.

	best correct rate	neighbor size	correct rate with 5% of training data per testing data	Eager SVM
add	**3.3216**	100	12.5163	**3.3216**
boston	**10.7647**	100	44.1430	**10.7647**
cput	**15.2426**	100	371.700	**15.2426**
fourclass	**100**	65	77.3782	**100**
glass	**71.4953**	75	41.1215	**71.4953**
heart	**84.444**	65	71.1111	*84.0741*
iris	**97.3333**	65	56.6667	**97.3333**
vowel	**99.4318**	100	39.5833	**99.4318**
wine	**99.4382**	90	74.1573	**98.8764**

Table 9: Performances of lazy SVMs using the second speedup algorithm on different datasets with Gaussian RBF kernels. For classification problems, correct rates all are shown in %. For regression problems, correct rates are mean square errors. Note that for a dataset, the best result is shown in bold face, and second emphasized. Neighbor size is the number of training data a testing data uses in the lazy SVM which has the best correct rate.

	best correct rate	neighbor size	correct rate with one of training data per testing data	Eager SVM
add	*3.8058*	10	7.0696	**3.3216**
boston	**10.0725**	4	12.5183	*10.7647*
cput	*16.9028*	10	26.9222	**15.2426**
fourclass	**100**	1	**100**	**100**
glass	**74.2991**	5	*71.9626*	71.4953
heart	*82.5926*	4	81.8519	**84.0741**
iris	**98.6667**	2	*97.3333*	*97.3333*
vowel	**99.6212**	6	*99.4318*	*99.4318*
wine	**98.8764**	7	*97.1910*	**98.8764**

Table 10: Performances of lazy SVMs using the second speedup algorithm on different datasets with polynomial kernels. For classification problems, correct rates all are shown in %. For regression problems, correct rates are mean square errors. Note that for a dataset, the best result is shown in bold face, and second emphasized.Neighbor size is the number of training data a testing data uses in the lazy SVM which has the best correct rate.

	best correct rate	neighbor size	correct rate with one training data per testing data	Eager SVM
fourclass	**99.8840**	3	99.6520	*99.7680*
glass	**75.2336**	6	71.9626	*73.8318*
heart	*82.5926*	3	82.2222	**84.0741**
iris	**98.6667**	2	96.6667	*98.000*
vowel	**99.0530**	4	97.9167	*98.8636*
wine	**98.3146**	8	97.1910	**98.3146**