**H-0260 (H0810-010) October 6, 2008**
**Computer Science**

# IBM Research Report

## Support Vector Machine Solvers: Large-scale, Accurate, and Fast (Pick Any Two)

**Haggai Toledano, Elad Yom-Tov, Dan Pelleg**
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

**Edwin Pednault, Ramesh Natarajan**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

**Research Division**
**Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Support vector machine solvers: Large-scale, accurate, and fast (Pick any two)

Haggai Toledano
IBM Haifa Research Lab
Haifa 31905, Israel
haggait@il.ibm.com

Elad Yom-Tov
IBM Haifa Research Lab
Haifa 31905, Israel
yomtov@il.ibm.com

Dan Pelleg
IBM Haifa Research Lab
Haifa 31905, Israel
dpelleg@il.ibm.com

Edwin Pednault
IBM T.J. Watson Research Center
Yorktown Heights, NY, 10598, USA
pednault@us.ibm.com

Ramesh Natarajan
IBM T.J. Watson Research Center
Yorktown Heights, NY, 10598, USA
nramesh@us.ibm.com

## Abstract

*Support vector machines (SVMs) have proved to be highly successful for use in many applications that require classification of data. However, training an SVM requires solving an optimization problem that is quadratic in the number of training examples. This is increasingly becoming a bottleneck for SVMs because while the size of the datasets is increasing, especially in applications such as bioinformatics, single-node processing power has leveled off in recent years. One possible solution to these trends lies in solving SVMs on multiple computing cores or on computing clusters.*

*We introduce a new parallel SVM solver based on the Forgetron algorithm. We compare this solver to a previously proposed parallel SVM solver and to a single node solver. The comparison covers accuracy, speed, and the ability to process large datasets. We show that while none of these solvers performs well on all three metrics, each of them ranks high on two of them. Based on these findings we discuss how practitioners should choose the most appropriate SVM solver, based on their requirements.*

## 1. Introduction

Support-vector machines (SVMs) [16] are a class of algorithms that have, in recent years, exhibited superior performance compared to other pattern classification algorithms. SVMs work by looking in the training data to find samples that support a hyperplane, which separates the classes of the data, with the objective of maximizing the margin between the classes. These samples are known as the support vectors. SVMs are further augmented by using a kernel function to optionally map the data onto a higher dimension. This makes it possible to find a good separating hyperplane, even when one does not exist in the original feature space.

Training an SVM requires solving a quadratic programming problem, whose size grows with the square of the number of training samples. Consequently, much work has been devoted to finding efficient solution methods for SVMs. (See Chapter 10 of [16] for a taxonomy of such methods.)

Recently, large datasets that would benefit from SVMs have become abundant. The problems in fields such as bioinformatics, textual processing in the Internet, medical imaging, and particle physics all generate huge amounts of data. For example, a dataset comprised of the pages saved for one year (2005) of the Internet archive is approximately 600 TB in size [1].

Diminishing improvements in the processing power of single processing cores [8] is another important trend. Increasing the speed of current processing cores is too expensive, both financially and in terms of power consumption. This is the reason why many microprocessor manufacturers are developing multi-core processors as their next offering for increased processing power.

These two complementary trends mean that current SVM solvers will not be sufficient for some datasets. A new approach based on parallel learning is required in order to build useful models for these data. However, such algorithms are not without their drawbacks. In this paper we consider three important objectives of SVM solvers: accuracy, speed for building the classifier, and the ability to learn from large amounts of data.

We begin by providing an outline of popular solution methods for SVM solvers. Interior point algorithms solve the optimization problem by simultaneously satisfying the primal and dual feasibility conditions of the quadratic pro-

gramming problem. These algorithms work by iteratively solving a set of equations. Many SVM solvers use subset selection to reduce the problem size. The initial idea for subset selection, known as 'chunking' [18], works by storing part of the data in memory, finding the support vectors for this partial problem, and replacing all the points that are not support vectors with new data, until convergence is met. This approach works well if the whole set of support vectors can be kept in memory. However, when this is not the case, chunking will converge extremely slowly.

Working Set algorithms implement a different approach to subset selection. These algorithms perform gradient descent on a subset of the variables, known as the working set, while freezing other variables. The working set approach is taken farthest in Platt's sequential minimal optimization (SMO) algorithm [14], where the working set is comprised of two samples and the analytic update to the variables is computed.

Iterative solvers sample the training data repeatedly and the weights are updated accordingly. This can be achieved using a modified Perceptron algorithm [6] or through algorithms such as the AdaTron algorithm [7].

Theoretically, SVM solution methods have $O(n^3)$ time complexity and $O(n^2)$ space complexity [17]; though, in practice, the computational complexity is usually closer to $O(n^2)$ [3]. Recently some approximation algorithms have claimed to further reduce the computational complexity of SVMs to $O(n)$ complexity [17]. However, a recent study [13] suggested that this reduction may come at the cost of higher error rates. These complexities imply that only small to medium sized datasets can be solved by single node SVMs.

Some research has been devoted to solving SVMs in parallel on multiple computing nodes. In [3] the SVM solver is parallelized by training multiple SVMs, each on a subset of the training data, and aggregating the resulting classifiers into a single classifier. The training data is then redistributed to the classifiers according to their performance and the process is iterated until convergence is reached. A more low-level approach is taken in [21], where the quadratic optimization problem is divided into smaller quadratic programs (similar to the Active Set methods), each of which is solved on a different node. The results are aggregated and the process is repeated until convergence. Graf et al. [9] partition the data and solve an SVM for each partition. The support vectors from each pair of classifiers are then aggregated into a new training set, for which an SVM is solved. The process continues until a single classifier remains. The aggregation process can be iterated using the support vectors of the final classifier in the previous iteration to seed the new classifiers.

Implementing a parallel SVM solver has its drawbacks in that the programming effort needed to realize it is significantly larger than that of a single node solver. This was mitigated by the recent introduction of the IBM Parallel Machine Learning (PML) toolbox[1]. Using this toolbox, users can easily code parallel learning algorithms by implementing a small number of functions. All parallel algorithms described in this paper were realized using the PML.

The main contributions of this article are as follows. First, we demonstrate a fast new solver for SVMs that includes attractive properties for many real-life applications. The second contribution is the comparison of this solver to two other solvers, illustrating the strengths and weaknesses of these solvers.

## 2. Problem statement

In this section we describe the SVM classifier and the optimization problem required for its construction. For simplicity we consider a binary SVM classifier, although SVMs have been used for regression, ranking, and other discriminative learning tasks. We begin with a training set:

$$D = \{(\mathbf{x}_i, y_i)\}, \quad i = 1, \ldots, N, \quad \mathbf{x}_i \in \Re^m, \quad y_i \in \{-1, 1\} \tag{1}$$

The goal of the SVM classifier is to learn a mapping from $\mathbf{x}_i$ to $y_i$ such that the error in mapping, as measured on a test dataset, would be minimal. SVMs learn to find the linear weight vector that separates the two classes so that:

$$y_i \left(\mathbf{x_i} \cdot \mathbf{w} + b\right) \geq 1 \;\; for \;\; i = 1, \ldots, N \tag{2}$$

There may exist many hyperplanes that achieve such separation, but SVMs find a weight vector $\mathbf{w}$ and a bias term $b$ that maximize the margin, $2/\left\|\mathbf{w}\right\|$. Therefore, the optimization problem that needs to be solved is:

$$Minimize \;\; J_D(\mathbf{w}) = \frac{1}{2}\left\|\mathbf{w}\right\| \tag{3}$$

$$Subject\ to\ y_i \left(\mathbf{x_i} \cdot \mathbf{w} + b\right) \geq 1 \;\; for \;\; i = 1, \ldots, N \tag{4}$$

Any points lying on the hyperplane $y_i \left(\mathbf{x_i} \cdot \mathbf{w} + b\right) = 1$ are called *support vectors*.

If the data cannot be separated using a linear separator, a slack variable $\xi \geq 0$ is introduced and the constraint is relaxed to:

$$y_i \left(\mathbf{x_i} \cdot \mathbf{w} + b\right) \geq 1 - \xi_i \;\; for \;\; i = 1, \ldots, N \tag{5}$$

The optimization problem then becomes:

$$Minimize J_D(\mathbf{w}) = \frac{1}{2}\left\|\mathbf{w}\right\| + C\sum_{i=1}^{N}\xi_i \tag{6}$$

$$Subject\ to\ y_i \left(\mathbf{x_i} \cdot \mathbf{w} + b\right) \geq 1 - \xi_i \;\; for \;\; i = 1, \ldots, N \tag{7}$$

[1] http://www.alphaworks.ibm.com/tech/pml

$$\xi_i \geq 0 \;\; for \;\; i = 1, \ldots, N \tag{8}$$

The weights of the linear function can be found directly or by converting the problem into a dual problem. Using the notation of [19], the dual problem is thus:

$$Maximize \;\; L_D(\mathbf{h}) = \sum_i h_i - \frac{1}{2}\mathbf{h} \cdot \mathbf{D} \cdot \mathbf{h} \tag{9}$$

$$subject\ to \;\; 0 \leq h_i \leq C, \;\; i = 1, ..., N \tag{10}$$

where $\mathbf{D}$ is a matrix such that $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ and $K(\cdot, \cdot)$ is either an inner product of the samples or a function of these samples. In the latter case, this function is known as the *kernel function*, which can be any function that complies with the Mercer conditions [16]. For example, these may be polynomial functions, radial-basis (Gaussian) functions, or hyperbolic tangents. If the data is not separable, $C$ is a tradeoff between maximizing the margin and reducing the number of misclassifications.

The classification of a new data point is then computed using the following equation:

$$f(\mathbf{x}) = sign\left(\sum_{i \in SV} h_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right) \tag{11}$$

Therefore, finding a solution to the SVM optimization problem can be formulated as finding a weight vector (as in Equation 6) or the Lagrange multipliers, as in Equation 9.

## 3. Description of algorithms

In this section we describe three SVM solvers that have different merits. One is SVMlight [10], a popular single node algorithm, which is fast and accurate for small datasets but performs poorly on large ones. Another is a modification of a sequential parallel solver [20], which is as accurate as SVMlight but is slow to converge. The third proposed approach is is a parallel version of the Forgetron algorithm [4], which is extremely fast and can handle large datasets but is less accurate compared to the other solvers.

**SVMlight** is an efficient single-node solver for SVMs. The solver is based on a working set method but is enhanced by two important improvements: a heuristic for selecting only some of the training data for the optimization problem (also referred to as shrinking) and the caching of kernel values. The latter improvement is important because, as shown in [20], kernel evaluations account for most of the computational load in problems that are not easily separable. However, because SVMlight is a single-node solver, the full kernel matrix cannot usually be held in memory. This implies that, depending on the specific dataset and the size of the memory, some kernel values have to be recomputed numerous times during optimization procedure.

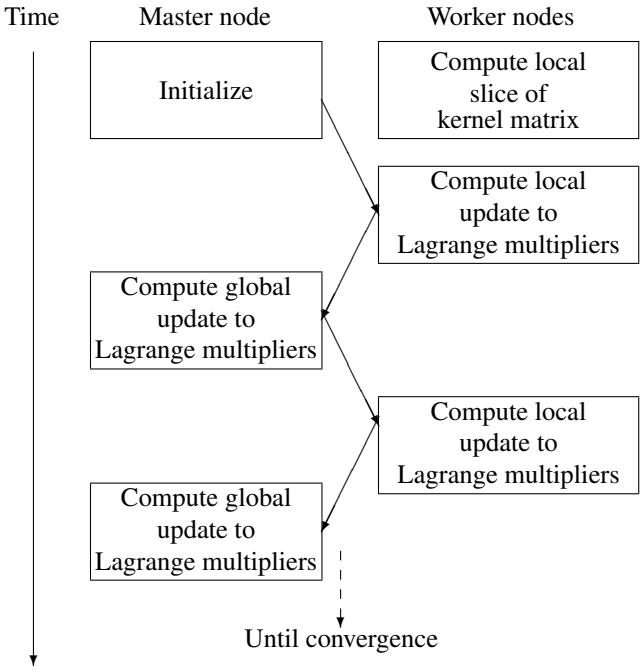


**Figure 1. Schematic diagram of the parallel batch SVM solver (SVM-PB) showing the timing and the division of work between master and worker nodes**

The repeated computation of kernel matrix values is is the motivation for the parallel SVM solver proposed in [20]. This algorithm is a parallel batch solver (**SVM-PB**) based on a modification of the sequential solver developed in [19]. The idea is to compute the full kernel matrix and hold it in a distributed memory, where each node holds part (several rows) of the kernel matrix. The optimization problem is solved iteratively. (See Figure 1 for a schematic diagram.) A master node first generates an initial solution comprised of an initial guess to the Lagrange multipliers. Each node updates this solution based on the part of the kernel matrix it holds in memory and sends the updated solution to the master node. The master node aggregates these updates and sends them to the worker nodes, until the process converges. The main drawback of this algorithm is the communication load associated with transferring updates between nodes, which degrade the execution speed of the algorithm.

We propose a third approach based on an ensemble of smaller SVMs. These SVMs are trained using the Forgetron algorithm, applied to test data, and then classified using majority voting. This approach is attractive because it reduces the communication load to a minimum. The idea of using an ensemble of SVMs was proposed in [11], but only in the context of using the ensemble as a basis for a boosting algorithm. In the case of parallel solvers, this takes away the benefits associated with reduced communication and is thus

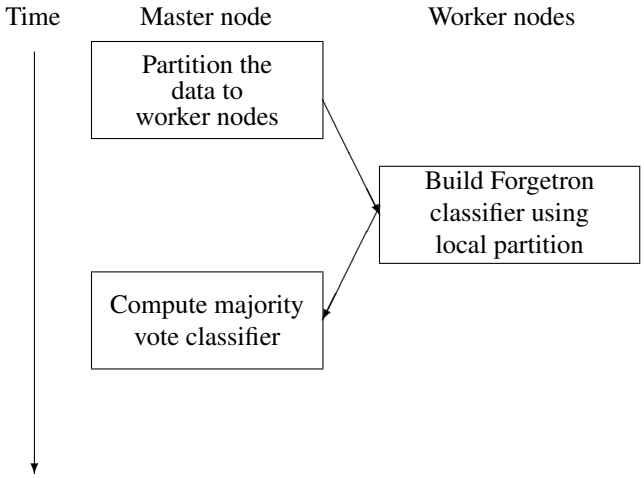


**Figure 2. Schematic diagram of the parallel Forgetron SVM solver (SVM-PF) showing the timing and the division of work between master and worker nodes**

impractical.

The Forgetron algorithm [4] is a kernel-based on-line SVM solver. It is especially useful in cases where much data exists because it imposes a limit on the resources used by the algorithm: the number of support vectors in the solutions is pre-specified by the user. This makes it possible to trade accuracy for memory. Thus, when the training data is abundant, the sparseness of the solution can be set according to the memory constraints of the resulting classifier. The parallel version of the Forgetron algorithm we propose in this article uses multiple instances of the Forgetron, each of which learns a subset drawn without replacement from the training dataset, such that the union of subsets is the complete training set. The final classifier is a majority vote on the output of those classifiers. We label this algorithm **SVM-PF**.

The schematic process required for constructing this classifier is shown in Figure 2. Note that only a single communication round between worker node and master node is required. This greatly speeds up computation, compared to the SVM-PB algorithm, where multiple iterations are required.

The computational load associated with this setup is drastically reduced during the training phase. Every $M$ Forgetron solvers processes a kernel matrix, which is smaller by a factor of $M$, hence the computational load is reduced by a factor of approximately $M^2$. Clearly, we are interested in knowing how many Forgetron instances should be used. Dividing the data between many Forgetrons will speed training because each Forgetron trains using a small subset of the data. However, this also degrades the accuracy of each individual classifier, which sees only a small sample of the data. That said, using many Forgetrons will improve the accuracy of the majority vote stage [12]. This question is investigated in the Results section.

## 4. Experimental setup

We compared the performance of the three SVM solvers on eight medium-sized datasets taken from the UCI repository [2]: `Adult`, `Isolet`, `Letter recognition`, `Mushrooms`, `Nursery`, `Page blocks`, `Pen digits`, and `Spambase`. The characteristics of all datasets are summarized in Table 1. We used medium-sized datasets rather than very large ones because our goal was to show how solvers scale as a function of the dataset size. This goal is important since, while processor speed is leveling off, larger datasets will always be available, and the more important question is therefore how different solvers scale as a function of dataset size, rather than their run-times on current machines. Such scaling can only be computed if datasets can be run on single nodes.

We followed an experimental protocol described in [15], as follows: Nominal attributes with $t$ possible values were substituted by $t$ binary features, where the $i-th$ binary feature was set to 1 if and only if the corresponding nominal attribute took the $i-th$ possible value. For each feature, we computed its average and standard deviation over the training set, and used these to normalize the data (training and testing) by subtracting the average and dividing by the standard deviation.

We used ten-fold cross-validation (10xCV); namely, in each fold, the union of nine out of ten equally-sized random subsets was used for training, and the tenth for testing. In all our experiments, we used a radial basis function (RBF) kernel. To optimize the RBF kernel parameter $\sigma$ and the classifier cost parameter, C, we followed [15] and used a simple greedy search via 10xCV over the training set. Initial values of $\sigma$ and C were set to 1. The value of $\sigma$ was then increased or decreased by a factor of 2 until no improvements were observed for three consecutive attempts. Then, $\sigma$ was fixed at the best value found and an identical optimization was performed over C. For the `adult` dataset, we used default parameters ($\sigma = 1, \quad C = 0.5$), mainly because SVMlight would not converge within a reasonable amount of time using other sets of parameters.

The computing nodes were a cluster of 2.4 - 3.4 GHz Intel Pentium machines with 2 GB memory running Linux.

The convergence criterion we used for SVMlight was the default stopping criterion. SVM-PB was limited to 250 iterations or an update to the gradient smaller than $10^{-3}$. SVM-PF does not require a stopping criterion because it performs a single pass over the data.

| Dataset | **Number of examples** | **Number of features** |
|---|---|---|
| Adult | 32561 (16281) | 105 |
| Isolet | 6238 (1559) | 617 |
| Letter | 20000 | 16 |
| Mushrooms | 8124 | 117 |
| Nursery | 12960 | 25 |
| Page blocks | 5473 | 10 |
| Pen digits | 7494 (3498) | 16 |
| Spambase | 4601 | 57 |

**Table 1. Summary of datasets. The number of examples in parentheses is the number of test examples (if a train/test partition exists)**

## 5. Results

Table 2 shows the error rates obtained for the three solvers. The rates for the Forgetron algorithm are shown for five computing nodes, under two budget constraints: 10% and 50%. These budgets represent the pecentage of the original data points which could be selected as support vectors.

We tested the error rates from Table 2 for statistical significance using the comparison methodology for multiple algorithms described in [5]. Specifically, we computed the $F_F$ statistics of Iman and Davenport (1980) (see [5]) with a confidence level of 90% to determine if all compared algorithms exhibited the same performance. According to this test, the error rates of the algorithms (with respect to the entire collection of datasets) were statistically different with a confidence level of 90%. We then applied a Bonferroni-Dunn test (Dunn, 1961; see [5]) to the error rates. This test showed that the Forgetron with 10% budget was significantly worse than the other three classifiers. The difference between the other three classifiers was not significant statistically.

We therefore conclude that SVMlight and SVM-PB offer superior accuracy compared to SVM-PF with a low budget. If used with a high budget, SVM-PF demonstrates accuracy that is not significantly different from that of SVMlight and SVM-PB.

Figure 3 compares the run-times and speed-ups of the three solvers as a function of the number of computing nodes for seven of the smaller datasets. We excluded the Adult dataset because running SVM-PB on a single node caused excessive cache misses and consequently prohibitively long execution times. The run-time for SVMlight can only be computed on a single node. Therefore, it is denoted by a circle on the figure in a location corresponding to a single node.

Speedup for $N$ nodes is defined as the time required for completing the computation using a single node divided by the time required using $N$ nodes. Doubling the computational power does not usually entail halving the computational time because of communication overheads. As a result, linear speedup is usually considered good performance for a parallel algorithm.

In Figure 3, SVM-PF is shown with a budget of 50% support vectors. The budget has a negligible effect on run-times.

The run-time figures show that SVM-PF is the fastest solver even on a single node, followed by SVMlight. SVM-PF is approximately two orders of magnitude faster than SVM-PB, averaged over all datasets and node numbers.

Interestingly, SVM-PF shows superlinear speedup for the nursery the dataset. That is, doubling the number of nodes more than halves the computation time. Furthermore, there is a high correlation (Spearman $\rho = 0.71$) between the maximal speedup obtained using the Forgetron algorithm and the square of the number of samples in the data.

This speedup can be explained by looking at the allocation of memory blocks. Halving the number of samples for each Forgetron means allocating memory for a kernel matrix that is only one quarter the size of the kernel matrix required for the full sample set. In many operating systems allocating a smaller memory block is much quicker. Furthermore, only one quarter of the kernel computations are performed at each Forgetron. Thus, it seems that the higher the number of nodes, the higher the speedup that SVM-PF will obtain. However, this picture is misleading. First, at a high number of nodes the communication rates will cause a degradation in performance. Moreover, the accuracy of the resulting classifier is dependent on the number of nodes.

As the number of Forgetrons increases (due to additional computing nodes), the accuracy of the majority vote improves. However, since the dataset size is constant, each Forgetron is trained using fewer samples. This implies that the error for each Forgetron will likely be higher. Thus, there is a trade-off between two opposing trends.

Figure 4 shows the accuracy of SVM-PF as a function of the number of nodes, using a budget of 50% support vectors. Similar trends were obtained when the budget was 10%. For some datasets (for example, Mushroom) the difference between the minimal and the maximal error is large (in the case of Mushrooms, it is fivefold), with most datasets demonstrating a clear minimum error. Thus, the number of computing nodes should be taken as an optimization parameter to obtain the best possible performance from the algorithm, especially when the size of the training data is small.

Finally, we note that a careful analysis of the results provided above shows that there is a high correlation between the run times of the three algorithms and a multiplication of the square of the number of examples in a data set and the

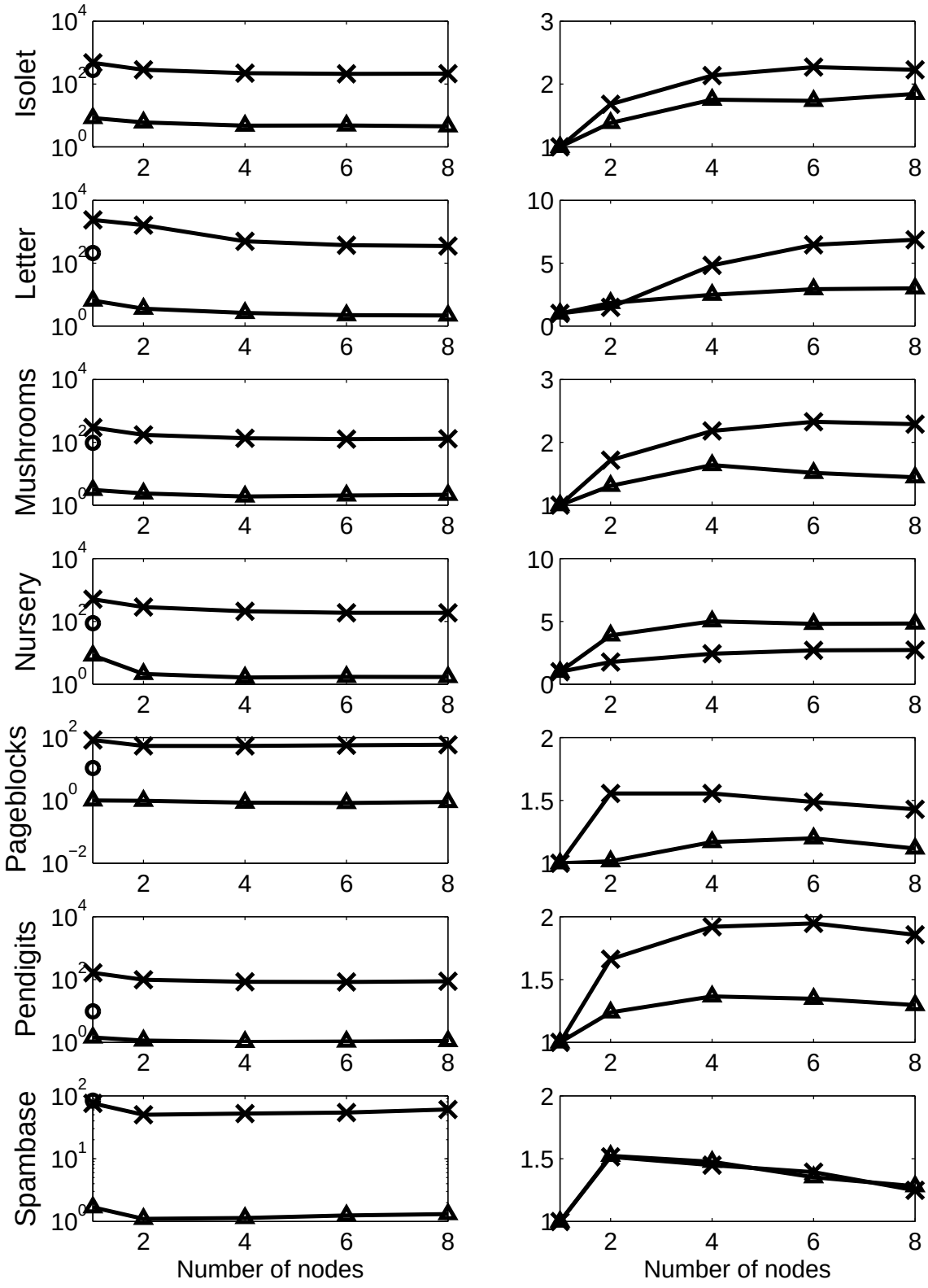


**Figure 3. Run-times (left) and speedup (right) versus the number of computing nodes for the different datasets. SVM-PB is denoted by crosses, SVM-FB by triangles, and SVMlight by circles. Run-times (in seconds) are plotted on a logarithmic scale.**

| Dataset | **SVM-PF 10% bud.** | **SVM-PF 50% bud.** | **SVM-PB** | **SVMlight** |
|---|---|---|---|---|
| Adult | 19.81 | **17.74** | 18.08 | 19.90 |
| Isolet | 50.03 | 50.05 | **5.84** | 49.97 |
| Letter | 13.59 | 8.39 | **2.06** | 2.30 |
| Mushrooms | 6.06 | 1.48 | 0.05 | **0.02** |
| Nursery | 11.71 | 6.81 | 5.29 | **0.02** |
| Page blocks | 9.05 | 8.68 | 4.08 | **2.74** |
| Pen digits | 5.23 | 4.15 | **1.37** | 1.57 |
| Spambase | 15.28 | 11.59 | 16.57 | **6.57** |

**Table 2. Error rates for the parallel Forgetron (SVM-PF) with 10% budget and 50% budget, the parallel batch solver (SVM-PB) and SVMlight. The lowest error rates for each datasets are marked in bold.**

number of features in it. This is to be expected because an RBF kernel requires computing a sum over all features for each pair of examples. Using a power law curve, an $R^2$ of 0.91, 0.61, and 0.70 was obtained for SVM-PF, SVM-PB, and SVMlight, respectively. However, the scaling of the curve is much shallower for SVM-PF and SVM-PB compared to SVMlight: For the two parallel solvers it is in the order of 0.35 while for SVMlight it is as high as 0.71. Thus, as datasets grow in size and in the number of features that they comprise of, the run-time of SVMlight grow far faster compared to the two parallel solvers.

We therefore conclude this section by summarizing that SVM-PF demonstrates superior scaling and run-time compared to SVM-PB. This is further shown in Figure 5, where run-times are plotted as a function of error rates for each of the eight datasets using the two parallel solvers. As this figure shows, most datasets processed using SVM-PB fall in the top left corner of the graph, indicating low error rates but high run-times. Datasets processed using SVM-PF fall in the bottom right corner of the graph, with a higher error rate but shorter run times.

The ability to run on a parallel platform also means that large datasets can be easily handled by SVM-PF and SVM-PB, compared to SVMlight, which is accurate for small datasets, but cannot scale to large ones.

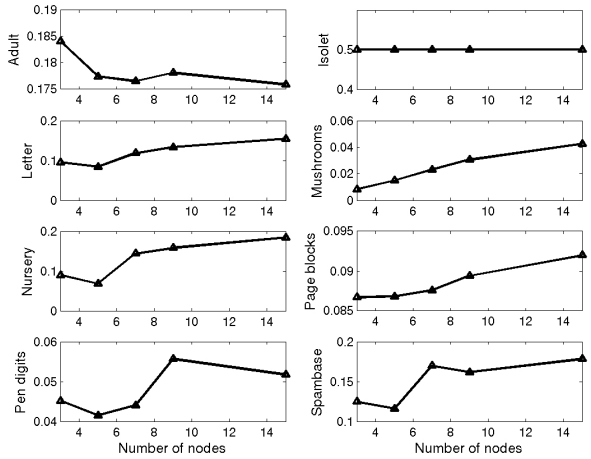


**Figure 4. Error versus the number of computing nodes.**

## 6. Summary

Support vector machines have been demonstrated to outperform many other classification techniques in terms of accuracy. Unfortunately, this accuracy is achieved at the price of a computationally expensive optimization problem. Although datasets become ever larger, single node processing power has leveled off in recent years. Therefore, it is important to find solutions to using SVMs in ways that overcome these scaling constraints.

In this article we dealt with three important attributes of classifiers: accuracy, the ability to train using large datasets, and the speed at which classifiers are trained. Our experiments indicate that SVMlight, a single-node solver, is accurate and fast, but does not scale to very large datasets. Using a distributed computational platform enables the use of new parallel SVM solvers. The parallel batch SVM (SVM-PB) we tested holds the full kernel matrix in memory. This means that it is both accurate and can scale to large datasets, but it is relatively slow, especially when used on easily separable datasets. (In such cases, it is usually unnecessary to compute the whole kernel matrix as shown in [20].) Finally, a parallel version of the Forgetron algorithm was shown to be both fast and scalable, but at the expense of accuracy.

Our experiments thus indicate how practitioners can choose the most appropriate solver for their needs. If computational resources are limited to a single node, the dataset

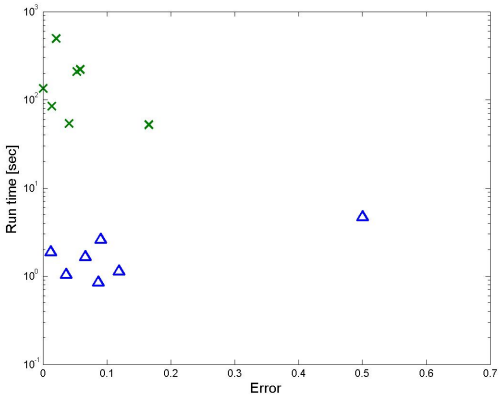


**Figure 5. Run time versus error rates for the eight datasets using the two parallel solvers. SVM-PB is denoted by crosses and SVM-FB by triangles. The vertical axis is log scaled.**

is small, and accuracy is essential, the single node solver should be chosen. If several computing nodes are available and data is abundant, SVM-PB should be preferred when accuracy is the significant factor and SVM-PF should be used when speed or compactness of the classifier is of greater importance.

We compared solvers using the three attributes that are most commonly of interest. However, there are other attributes that may be important under certain circumstances. For example, there are cases where privacy of data is a concern. In other cases, data might be distributed in geographically separate locations and merging the data would entail a high overhead. Under these circumstances, distributed solvers such as the ones discussed in this article will have additional benefits compared to the single node solvers. For example, when data is divided between several parties and cannot be shared between them because of privacy concerns, SVM-PF can be utilized. A Forgetron is trained on a subset of the data and merging takes place at the voting stage, i.e., each Forgetron labels a test example and only provides this label to the classifier.

## References


[1] W. Y. Arms, S. Aya, P. Dmitriev, B. J. Kot, R. Mitchell, and L. Walle. Building a research library for the history of the web. In *JCDL '06: Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, pages 95–102, New York, NY, USA, 2006. ACM Press.

[2] C. L. Blake, E. J. Keogh, and C. J. Merz. UCI repository of machine learning databases, 1998.

[3] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. In *Advances in Neural Information Processing Systems*. MIT Press, 2002.

[4] O. Dekel, S. Shalev-Shwartz, and Y. Singer. The forgetron: A kernel-based perceptron on a fixed budget. In *Advances in Neural Information Processing Systems 18*, pages 259–266, 2006.

[5] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[6] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley and Sons, Inc, New-York, USA, 2001.

[7] T.-T. Friess;, N. Cristianini, and C. Campbell. The kernel-adatron algorithm: A fast and simple learning procedure for support vector machines. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 188–196, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[8] D. Geer. Industry trends: Chip makers turn to multicore processors. *Computer*, 38(5):11–13, 2005.

[9] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. In *Advances in Neural Information Processing Systems*, 2004.

[10] T. Joachims. Making large-scale svm learning practical. In *Advances in Kernel Methods - Support Vector Learning*, 1999.

[11] H.-C. Kim, S. Pang, H.-M. Je, D. Kim, and S. Y. Bang. Pattern classification using support vector machine ensemble. *icpr*, 02:20160, 2002.

[12] L. Kuncheva, C. Whitaker, C. Shipp, and R. Duin. Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis and Applications*, 6(1):22–31, 2004.

[13] G. Loosli and S. Canu. Comments on the "core vector machines: Fast svm training on very large data sets". *Journal of Machine Learning Research*, 2007.

[14] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. In *Advances in Kernel Methods - Support Vector Learning*, pages 185–208, 1999.

[15] R. Rifkin and A. Klautau. In defense of One-vs-All classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[16] B. Schölkopf and A. J. Smola. *Leaning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press, Cambridge, MA, USA, 2002.

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

[18] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.

[19] S. Vijayakumar and S. Wu. Sequential support vector classifiers and regression. In *International conference on soft computing*, pages 610–619, 1999.

[20] E. Yom-Tov. A distributed sequential solver for large scale svms. In *Large scale kernel machines*, pages 141–156, 2007.

[21] G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel computing*, 29:535–551, 2003.