Theoretical and Empirical Analysis of a Spatial EA Parallel Boosting Algorithm

Uday Kamath kamathuday@gmail.com
Computer Science Department, George Mason University, Fairfax, Virginia, 22030, USA

Carlotta Domeniconi cdomenic@gmu.edu
Computer Science Department, George Mason University, Fairfax, Virginia, 22030, USA

Kenneth De Jong kdejong@gmu.edu
Computer Science Department, George Mason University, Fairfax, Virginia, 22030, USA

doi:10.1162/EVCO_a_00202

Abstract

Many real-world problems involve massive amounts of data. Under these circumstances learning algorithms often become prohibitively expensive, making scalability a pressing issue to be addressed. A common approach is to perform sampling to reduce the size of the dataset and enable efficient learning. Alternatively, one customizes learning algorithms to achieve scalability. In either case, the key challenge is to obtain algorithmic efficiency without compromising the quality of the results. In this article we discuss a meta-learning algorithm (PSBML) that combines concepts from spatially structured evolutionary algorithms (SSEAs) with concepts from ensemble and boosting methodologies to achieve the desired scalability property. We present both theoretical and empirical analyses which show that PSBML preserves a critical property of boosting, specifically, convergence to a distribution centered around the margin. We then present additional empirical analyses showing that this meta-level algorithm provides a general and effective framework that can be used in combination with a variety of learning classifiers. We perform extensive experiments to investigate the trade-off achieved between scalability and accuracy, and robustness to noise, on both synthetic and real-world data. These empirical results corroborate our theoretical analysis, and demonstrate the potential of PSBML in achieving scalability without sacrificing accuracy.

Keywords

Spatial evolutionary algorithms, parallel boosting, large margin classifiers, scalability, machine learning.

1 Introduction

Many real-world applications, such as web mining, social-network analysis, and bioinformatics, involve large datasets with millions, or even billions of data items. Many traditional supervised learning algorithms such as support vector machines (SVM) have training times of $O(n^3)$ and space complexity of $O(n^2)$, where n is the size of the training set (Tsang et al., 2005). In order to handle the large training sets required by these applications, one of two approaches is typically taken: (1) perform some form of sampling on the dataset to reduce its size, or (2) customize the learning algorithm

Manuscript received: January 29, 2016; revised: July 25, 2016; accepted: November 21, 2016.

to improve the running time via parallelization. In the first approach, sampling techniques frequently introduce unintended biases that reduce the accuracy of the results. Similar reductions in accuracy often result from modifications to a learning algorithm to improve its speed. Very few general procedures, under minimal assumptions, can parallelize any given machine learning algorithm, while keeping the desired balance between speed and accuracy.

Recently, a parallel spatial EA boosting machine learner (PSBML) was introduced (Kamath et al., 2012, 2013). PSBML uses concepts from spatially structured evolutionary algorithms (SSEAs) and ensemble learning to implement a parallel boosting process for handling large training datasets. In this article, we provide both a theoretical and an empirical analysis of PSBML to investigate its behavior and properties. Our findings reveal that the significance of PSBML is two-fold. First, it provides a general framework for parallelization, and as such it can be used in combination with a variety of learners. Second, it is capable of achieving effective speed-ups without sacrificing accuracy.

We use Gaussian mixture models (GMMs) combined with the mean-shift procedure to establish an analytical model of PSBML, and show that it converges to a data distribution whose modes are centered on the margin of the classification boundary. As such, the algorithm inherits the properties of good generalization and resilience to noise that are associated with large margin classifiers. We perform extensive experiments to evaluate the performance of PSBML with a variety of learners, to measure the speed and accuracy trade-offs achieved, and to test the effect of noise. All results confirm the strength of PSBML anticipated by our theoretical findings. To the best of our knowledge this is the first attempt in the literature that establishes a theoretical connection between spatial evolutionary algorithms and machine learning concepts.

2 Related Work

The PSBML approach incorporates concepts from spatially structured evolutionary algorithms (SSEAs), ensemble learning, and boosting techniques.

SSEAs are evolutionary algorithms in which individuals are embedded in a metric space which constrains how individuals in the population may interact, and how they are compared and updated (Sarma and De Jong, 1996; Giacobini et al., 2005). SSEAs have been theoretically analyzed using diffusion models (Alba and Luque, 2004; Giacobini et al., 2005; Sarma and De Jong, 1997). In particular, various important properties such as how the shape and the structure of spatial topography affect diversity, the distribution of the best individuals, and convergence have been analyzed (Sarma and De Jong, 1996; Tomassini, 2005). Some of these findings constitute an important foundation for our theoretical and empirical analysis.

In ensemble learning, multiple classifiers are generated and combined to make a final prediction. It has been shown that ensemble learning, through the consolidation of different predictors, can lead to significant reductions in generalization error (Bennett et al., 2002). Of particular relevance is the AdaBoost technique and its variants, such as confidence-based boosting (Schapire and Singer, 1999). AdaBoost induces a classification model by estimating the hard-to-learn instances in the training data (Freund and Schapire, 1996). A formal analysis of the AdaBoost technique has derived theoretical bounds on the margin distribution to which the approach converges (Schapire et al., 1998).

In statistical learning theory, a formal relationship between the notion of *margin* and the generalization classification error has been established (Vapnik, 1995). As a result, classifiers that converge to a large margin perform well in terms of generalization error.

One of the most popular examples of such classifiers is support vector machines (SVMs). The classification boundary provided by an SVM has the largest distance from the closest training point. SVMs have been modified to scale to large data sets (Fan et al., 2008; Fung and Mangasarian, 2002; Bottou and Bousquet, 2008; Tsang et al., 2007; Joachims, 1999). Many of these adaptations introduce a bias caused by the approximation used, such as sampling the data or assuming a linear model, that can lead to a loss in generalization while trying to achieve speed.

To achieve scalable solutions with large data, algorithm-specific parallelizations have been performed using distributed architectures and network computing (Chang et al., 2007; Drost et al., 2010; Chu et al., 2007; Woodsend and Gondzio, 2009). These modifications have been conducted on algorithms like decision trees, rule inductions, and boosting algorithms (Shafer et al., 1996; Svore and Burges, 2011). In most cases, the underlying algorithm needs to be changed in order to achieve a parallel computation. For example, in SVMs or tree node learning, matrix transformations need to be performed. Many of the parallel paradigms use a communication infrastructure like message passing interfaces (MPI) for exchanges. MapReduce gives a generic framework for a divide-and-conquer–based approach and has been used in conjunction with learning algorithms to scale to large datasets (Chu et al., 2007). Ensemble-based learning on parallel networks has also been employed on various tree-based algorithms for learning on massive datasets (Svore and Burges, 2011).

In this article, to analyze the PSBML algorithm we make use of Gaussian Mixture Models (GMMs) and the mean-shift procedure. A GMM is a parametric probabilistic model consisting of a linear combination of Gaussian distributions with unknown parameters. Typically the parameter values are estimated so that the resulting model is the one that best fits the data (Rasmussen, 2000).

Mean-shift is a local search algorithm whose aim is to find the modes (i.e., local maxima) of a distribution. It achieves this goal by performing kernel density estimation, and iteratively locating the local maxima of the kernel mixture as the zeros of the corresponding gradient function (Carreira-Perpiñán and Williams, 2003). Convergence to local maxima is guaranteed from any starting point. Furthermore, it has been shown that, when combined with GMMs, mean-shift is equivalent to an expectation-maximization (EM) algorithm (Carreira-Perpiñán, 2005). The key advantage of using the mean-shift algorithm for mode finding on a given density is two-fold: (1) the approach is deterministic and nonparametric, since it is based on kernel density estimation; and (2) it poses no a priori assumptions on the number of modes (Carreira-Perpiñán and Williams, 2003; Carreira-Perpiñán, 2005).

3 The PSBML Algorithm

The algorithm for parallelizing boosting (PSBML) can be described as a meta-learning algorithm that uses any classifier capable of producing confidence measures on predictions. It has at its core a standard SSEA (Sarma and De Jong, 1996) in which a population of individuals (in this case, training data instances) are distributed over a two-dimensional toroidal grid. Associated with each node on the grid are two processes: an EA process and a machine learning (ML) process. As in SSEAs, these processes interact only with neighboring nodes as predefined by the neighborhood structure (see Figure 1). The goal of these two processes is to produce a boosting effect on the local population of training data instances, that is, obtain an effective balance between the diversity of instances in the local training set and the elimination of easy classification instances. This is achieved by having the local ML process generate estimates of

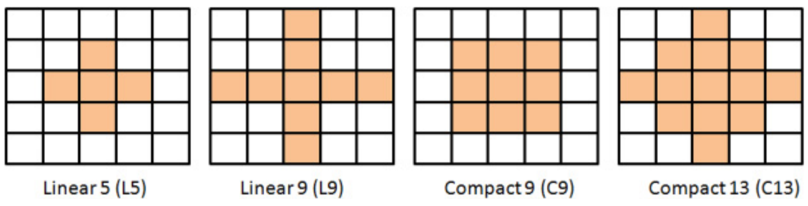

Figure 1: Two-dimensional grid with various neighborhood structures.

classification difficulty of the local training instances in its neighborhood, and the local EA uses this information to make training set modifications.

Because each individual training instance lies in multiple overlapping neighborhoods, the classification difficulty values are in fact ensemble estimates based on the estimates from each of the ML processes associated with those neighborhoods. The emergent collective effect is a global boosting process produced by a highly parallel set of local processes and distributed training data.

In the following, we describe in detail the different phases of the algorithm. The pseudocode of PSBML is given in Algorithm 1. The parameters for grid configuration, that is, width and height of the grid, replacement probability, and maximum number of iterations, are all included in *GridParam*.

3.1 Initialization

Given a collection of labeled data, an independent fixed validation set is created and the rest is used for training. PSBML uses the concept of wraparound toroidal grid to distribute the training data to each node in the grid. The training data is distributed across all the nodes using stratified uniform sampling of the class labels (line 1 of Algorithm 1).

3.2 Node Behavior at Each Cycle

The algorithm runs for a pre-determined number of iterations (or generational cycles) (line 4). The behavior of a node at each cycle can be divided into two phases: an ML phase and an EA phase. The ML phase consists of two steps: training and testing. During training, a node performs a standard machine learning process using its local data to generate a classifier (line 6). In the testing step, the classifier is applied to both the local training data and the training data in its neighborhood as defined by the SSEA (shown in Figure 1). For each training data instance, the classifier makes a classification prediction and outputs a confidence value for that prediction. In the subsequent EA phase, these confidence values are used to assign fitnesses to the instances, allowing the local EA in the node to select the most difficult instances for the next generation. That is, every node updates its local training data for the successive training cycle by probabilistically selecting instances based on the assigned fitness values (line 9).

Since each instance is a member of the neighborhood of multiple nodes, an ensemble assessment of difficulty is performed, similar to the boosting of the margin in AdaBoost (Schapire et al., 1998). Specifically, in PSBML the fitness v_i of an instance i is set equal to the smallest confidence value obtained from any node and for any class:

$$v_i = \min_{n \in N_i} v_{ni}$$

where N_i is a set of indices defined over the neighborhoods to which instance i belongs, and v_{ni} is the confidence credited to instance i by the learner corresponding to neighborhood n. These fitness values are then normalized through linear rescaling:

$$v_i^{norm} = \frac{v_i - v_{min}}{v_{max} - v_{min}},$$

where v_{min} and v_{max} are the smallest and the largest fitness values obtained across all the nodes, respectively. The selection probability (or weight) w_i associated to instance i is then set to:

$$w_i = 1 - v_i^{norm};$$

w_i is used in fitness-proportional selection so that the smaller the confidence credited to an instance i is (i.e., the harder to learn instance i is), the larger the probability for instance i to be selected will be. Instead of replacing the entire training data set at a node with new instances, in Kamath et al. (2012), a user-defined replacement probability P_r was introduced. Experimentally, it was determined that an overlapping-generation model for the local EA was more effective and the most effective value for P_r was found to be 0.20 (Kamath et al., 2012), which we will be using in all the experiments.

Algorithm 1: PSBML*(Train, Validation, GridParam)*

```
 1: InitializeGrid(Train, GridParam)         ▷ Distribute the instances over the nodes in grid
 2: currentMin ← 100
 3: Pr ← GridParam.pr                                        ▷ Probability of replacement
 4: for i ← 0 to GridParam.iter do
 5:     for j ← 0 to GridParam.nodes do                                       ▷ ML phase
 6:         TrainNodes(j)
 7:         TestAndWeighNodes(j)             ▷ Collect neighborhood data and assign weights
 8:     for j ← 0 to GridParam.nodes do                                       ▷ EA phase
 9:         NeighborData ← CollectNeighborData(j)
10:         NodeData ← NodeData ∪ NeighborData
11:         ReplaceData ← FitnessPropSel(NodeData, Pr)
12:         PrunedData ← Unique(ReplaceData)  ▷ Unique keeps one copy of instances in set
13:     ValClassifier ← createNew(GridParam.classifier)     ▷ New classifier for validation
14:     error ← Validate(PrunedData,Validation,ValClassifier)
                                                     ▷ Use validation set to track model learning
15:     currentMin ← error
16:     bestClassifier ← ValClassifier
17:     marginData ← PrunedData
18: return bestClassifier, marginData
```

3.3 Grid Behavior at Each Cycle

At each iteration, once all nodes have completed the process of generating a new local training dataset, a global assessment of the grid is performed to track the "best" classifier throughout the entire iterative process. The unique instances from all the nodes are collected and used to train a new classifier (lines 10 and 11). The independent validation set created during initialization is then used to test the classifier (line 12). This procedure resembles the "pocket algorithm" used in neural networks, which has shown to converge to the optimal solution (Gallant, 1990). This incremental estimate of the best classifier is output at each cycle and used to make predictions for unseen test instances (line 17).

3.4 Iterative Process

The fitness-proportional selection process and the interaction of neighboring nodes enable the hard instances to migrate throughout the various nodes, due to the wraparound nature of the grid. The rate at which the instances migrate depends on the grid structure, and more importantly on the neighborhood size and shape. Thus, the grid topology of classifiers and the data distribution across the nodes provide the parallel execution, while the interaction between neighboring nodes and the confidence-based instance selection give the ensemble and boosting effects.

4 Theoretical Analysis: PSBML Is a Large Margin Classifier

We use Gaussian Mixture Models (GMMs) combined with the mean-shift algorithm to model the behavior of PSBML. Specifically, we formally show that PSBML, through the fitness-proportional selection process, iteratively changes the data distribution, and converges to a distribution whose modes are centered around the margin, that is, around the hardest points to classify.

Each grid node in the PSBML algorithm, along with its neighborhood structure, represents a sample of the whole dataset, where each point is weighted according to how difficult it is to be classified. In our analysis, we fit a Gaussian mixture model on the weighted points, and apply the mean-shift procedure to locate the modes of the resulting distribution. We will show that, throughout the iterations of PSBML, as more data closer to the boundary are being selected, the data distribution will grow higher modes centered around the margin. These modes will be the ones visited by the mean-shift procedure, irrespective of the starting point.

Since each node in the toroidal grid has the same behavior, they all fit a Gaussian mixture model on their respective neighborhood. By consolidating the micro-behavior of the mean-shift procedure at each node, we obtain an overall convergence to a distribution with peaks centered around the boundary. Our analysis below, and the empirical results in Section 5, confirm this.

4.1 Distribution of a Node at Time $t = 1$

After the completion of the first iteration of PSBML, each classifier in the grid has been trained with its own data, and is tested on the instances of the neighbors, to which it assigns confidence values. A common approach to assess the confidence of a prediction for an instance is to measure its distance from the estimated decision boundary: the smaller the distance, the smaller the confidence will be. The resulting weight values drive the probability for a point to be selected for the successive iterations. Below we use a Gaussian mixture to model this process.

Consider a Gaussian mixture density of M components $p(\mathbf{x}) = \sum_{m=1}^{M} p(m)p(\mathbf{x}|m)$, where the $p(m)$ are the mixture proportions such that $p(m) > 0$, $\forall m = 1, \ldots, M$, and $\sum_{m=1}^{M} p(m) = 1$. Each mixture component is a Gaussian distribution in $\mathbf{R}^D$; that is, $\mathbf{x}|m \sim \mathcal{N}_D(\mu_m, \Sigma_m)$, where $\mu_m = \mathrm{E}_{p(\mathbf{x}|m)}[\mathbf{x}]$ and $\Sigma_m = \mathrm{E}_{p(\mathbf{x}|m)}[(\mathbf{x} - \mu_m)(\mathbf{x} - \mu_m)^T]$ are the mean and covariance matrix of the Gaussian component m.

Let us first consider a known result for the mean-shift procedure applied to Gaussian mixture models to find the modes of the distribution (Carreira-Perpiñán and Williams, 2003). No closed-form solution exists to this problem, so numerical iterative approaches have been developed. In particular, the fixed-point iterative method gives the following fixed-point solution (Carreira-Perpiñán and Williams, 2003): $\mathbf{x}^{(t+1)} = \mathbf{f}(\mathbf{x}^{(t)})$ where

$$\mathbf{x} = \mathbf{f}(\mathbf{x}) = \left(\sum_{m=1}^{M} p(m|\mathbf{x}) \Sigma_m^{-1} \right)^{-1} \sum_{m=1}^{M} p(m|\mathbf{x}) \Sigma_m^{-1} \mu_m. \tag{1}$$

Let us assume now that we model the sample data assigned to a node and to its neighbors using a Gaussian mixture distribution of M components in $\mathbf{R}^{\mathrm{D}}$. In our analysis, we consider only the distribution of one class; the argument stays the same for the other class due to the symmetry with respect to the boundary. We need to embed the fitness-proportional selection process performed by PSBML in our Gaussian mixture modeling. Let's assume the optimal boundary between classes is known. Let $\mathbf{s} \in \mathbf{R}^D$ be a point on the boundary. We estimate the distance of a point $\mathbf{x}$ from the boundary by considering its distance from $\mathbf{s}$. At each iteration of the PSBML algorithm, the weights bias the selection towards those points which are closer to the boundary: the larger the weight of a point is, the larger is the probability of being selected. To embed this mechanism in the Gaussian mixture modeling, we set the m^{th} component to be $p'(\mathbf{x}|m) = w(\mathbf{x}) * p(\mathbf{x}|m)$, where $w(\mathbf{x})$ is a Gaussian weghting function centered at $\mathbf{s}$:

$$w(\mathbf{x}) = (2\pi)^{-D/2} |\Sigma_{\mathbf{s}}|^{-1/2} e^{-1/2(\mathbf{x}-\mathbf{s})^T \Sigma_{\mathbf{s}}^{-1}(\mathbf{x}-\mathbf{s})}$$

and

$$p(\mathbf{x}|m) = (2\pi)^{-D/2} |\Sigma_m|^{-1/2} e^{-1/2(\mathbf{x}-\mu_m)^T \Sigma_m^{-1}(\mathbf{x}-\mu_m)}.$$

We compute the gradient of $p'(\mathbf{x}|m)$ with respect to the independent variable $\mathbf{x}$, while keeping the parameters μ_m and Σ_m fixed:

$$\frac{\partial p'(\mathbf{x}|m)}{\partial \mathbf{x}} = w(\mathbf{x}) \frac{\partial p(\mathbf{x}|m)}{\partial \mathbf{x}} + p(\mathbf{x}|m) \frac{\partial w(\mathbf{x})}{\partial \mathbf{x}}. \tag{2}$$

Considering each derivative:

$$\frac{\partial p(\mathbf{x}|m)}{\partial \mathbf{x}} = p(\mathbf{x}|m) \Sigma_m^{-1} (\mu_m - \mathbf{x}),$$
$$\frac{\partial w(\mathbf{x})}{\partial \mathbf{x}} = w(\mathbf{x}) \Sigma_{\mathbf{s}}^{-1} (\mathbf{s} - \mathbf{x}),$$

and substituting these results in Equation (2), we obtain:

$$\frac{\partial p'(\mathbf{x}|m)}{\partial \mathbf{x}} = w(\mathbf{x}) p(\mathbf{x}|m) \Sigma_m^{-1} (\mu_m - \mathbf{x}) + p(\mathbf{x}|m) w(\mathbf{x}) \Sigma_{\mathbf{s}}^{-1} (\mathbf{s} - \mathbf{x}). \tag{3}$$

We now turn to the mixture of M Gaussian distributions. By the linearity property of the differential operator, we obtain:

$$\frac{\partial p(\mathbf{x})}{\partial \mathbf{x}} = w(\mathbf{x}) \sum_{m=1}^{M} p(m) p(\mathbf{x}|m) \Sigma_m^{-1} (\mu_m - \mathbf{x}) + (\mathbf{s} - \mathbf{x}) w(\mathbf{x}) \sum_{m=1}^{M} p(m) p(\mathbf{x}|m) \Sigma_{\mathbf{s}}^{-1}.$$

By setting the above gradient to $\mathbf{0}$ and simplifying $w(\mathbf{x})$, we derive a fixed point iteration procedure that finds the modes of the distribution (Carreira-Perpiñán and Williams, 2003):

$$\sum_{m=1}^{M} p(m) p(\mathbf{x}|m) \Sigma_m^{-1} (\mu_m - \mathbf{x}) = (\mathbf{x} - \mathbf{s}) \sum_{m=1}^{M} p(m) p(\mathbf{x}|m) \Sigma_{\mathbf{s}}^{-1}.$$

Solving for $\mathbf{x}$, we obtain:

$$\mathbf{x} = \frac{\sum_{m=1}^{M} p(m)p(\mathbf{x}|m)\Sigma_{\mathbf{s}}^{-1}\mathbf{s} + \sum_{m=1}^{M} p(m)p(\mathbf{x}|m)\Sigma_m^{-1}\mu_m}{\sum_{m=1}^{M} p(m)p(\mathbf{x}|m)\Sigma_{\mathbf{s}}^{-1} + \sum_{m=1}^{M} p(m)p(\mathbf{x}|m)\Sigma_m^{-1}}.$$

Using Bayes rule and simplifying $p(\mathbf{x})$:

$$\mathbf{x} = \frac{\sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_{\mathbf{s}}^{-1}\mathbf{s} + \sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_m^{-1}\mu_m}{\sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_{\mathbf{s}}^{-1} + \sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_m^{-1}}.$$

Rearranging, we obtain our fixed-point solution:

$$\mathbf{x} = \left(\sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_{\mathbf{s}}^{-1} + \sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_m^{-1}\right)^{-1} \times \left(\sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_{\mathbf{s}}^{-1}\mathbf{s} + \sum_{m=1}^{M} p(m|\mathbf{x})\Sigma_m^{-1}\mu_m\right). \tag{4}$$

Comparing Equations (1) and (4) we can see that, by weighting the points according to their distance from the boundary, the modes of the resulting distribution become the weighted average of the means μ_m and $\mathbf{s}$. That is, each local classifier, by assigning weights to points according to the confidence of the prediction, causes the modes to shift towards the points closest to the estimated boundary, that is, towards its margin.

4.2 Distribution of the Grid at Time t = 1

The whole grid itself is modeled as a Gaussian mixture (given by the collection of GMMs at each node). Thus, the same derivation given above, applied to the grid, shows that the overall data distribution will have the same modes emerging from the individual nodes, that is, centered around the margin of the boundary.

4.3 Final Distribution of the Grid

After a number of iterations, at each node, data will be selected according to the current distribution. We can show that all the nodes will converge to the same mode. Suppose that a node i, at time t, has a neighborhood with means $T(t) = \{\mu_1^{(t)}, \ldots, \mu_l^{(t)}\}$, and one of these means, say $\mu_g^{(t)}$, is the closest (globally) to the boundary. During successive iterations, the sampling process causes the elimination of modes that are far from the boundary. Thus, after $k > 0$ steps, the local distribution of node i will have a smaller number of modes: $T(t+k) = \{\mu_1^{(t+k)}, \ldots, \mu_{l-m}^{(t+k)}\}$, with $l - m > 0$. Due to the fitness-proportional selection mechanism (note that the sample size remains constant at each iteration), $\mu_g^{(t)} = \mu_g^{(t+k)} \in T(t+k)$. The whole process converges when $T(t+1) = T(t)$, or the mean shift is negligible, and at convergence $T(t) = \{\mu_g^{(t)}\}$. We observe that spatially structured replication-based evolutionary algorithms show a similar behavior, where the global best is spread deterministically across the nodes, until all the nodes in the grid converge to the same individual according to logistic takeover curves (Sarma and De Jong, 1997).

5 Empirical Analysis of PSBML and GMMs with Mean-Shift

We performed a number of experiments to verify the established relationship between PSBML and GMMs with mean-shift. Using datasets representing two distinct dataset properties, we performed the following sequence:

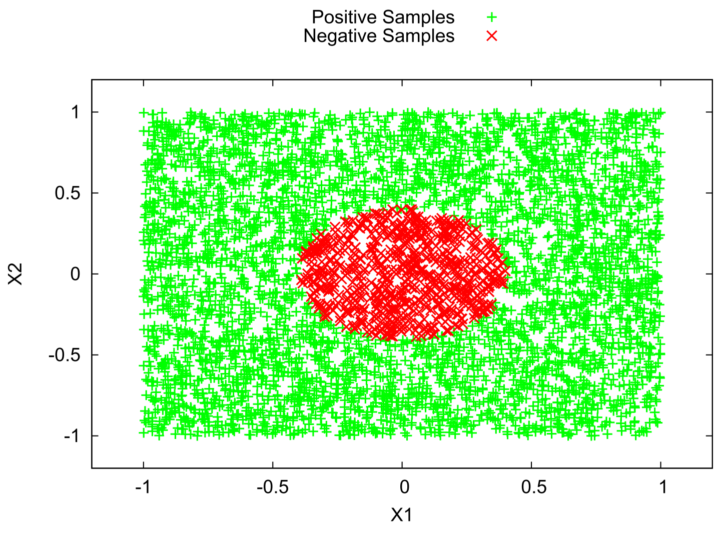

Figure 2: Circle dataset.

1. We ran the PSBML algorithm using a 5×5 spatial grid with the C9 neighborhood (see Fig. 1) and a large margin classifier, and observed the population distribution change over the training cycles.
2. We replaced each local classifier with a GMM with mean-shift, while keeping the grid structure and neighborhood interaction unchanged. Each data instance is weighted a priori using the Gaussian weighting function as defined in the theoretical analysis. We ran GMM with mean-shift on each node and performed sampling iteratively at every training cycle exactly as in PSBML. We again observed the changes in the population distribution over time.
3. We removed the grid and ran GMMs with mean-shift estimation on the whole dataset, with each instance weighted according to its distance from the known boundary as above. We observed the data distribution and final modes at convergence, and compared them with those obtained in the previous setting.

5.1 A Nonlinearly Separable Dataset

For the first dataset, instances were drawn at random within a square centered at the origin and with side of length two. Points with a distance smaller than 0.4 from the origin are labeled as negative, and those with a distance greater or equal than 0.4 are labeled as positive (see Figure 2).

We ran the three experiments described above on this data. For experiment 1, the large margin classifier used at each node fits a circle to its training set by setting its radius to the average distance of the origin from the smallest positive and the largest negative instances. For testing, the learner outputs "$-$" when the instance falls within the circle, and "$+$" otherwise. The confidence of the prediction is the distance of the instance from the circular boundary.

To compare the data distributions obtained in experiments 1 and 2, we recorded the number of points at various intervals of distances from the origin at training cycles 25 and 50. The resulting histograms are given in Figure 3. We can clearly observe

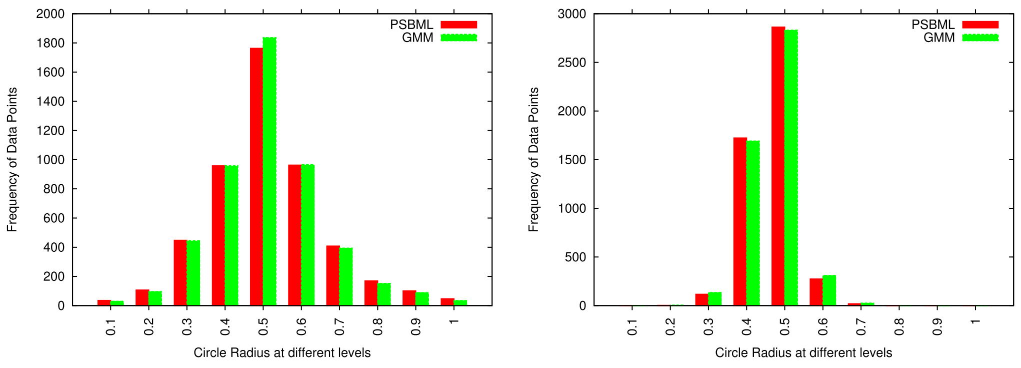

Figure 3: Circle dataset: data distribution at cycle 25 (Left) and 50 (Right) using PSBML and GMMs.

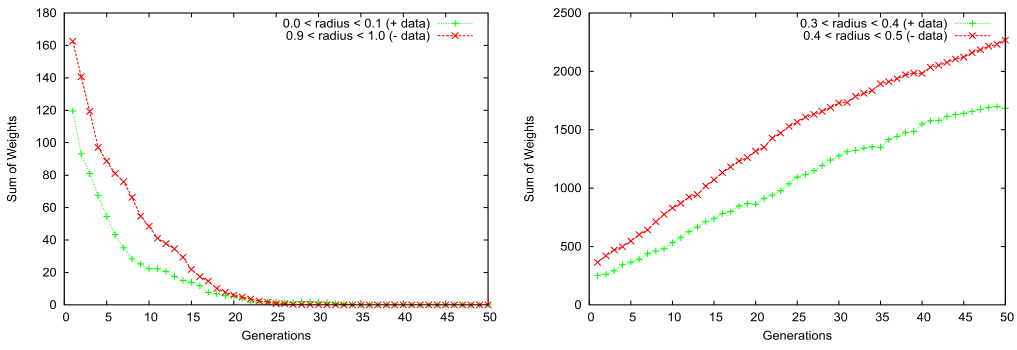

Figure 4: Changes in weight distribution as function of time: (Left) exponential decay; (Right) logistic increase.

that the two methodologies, PSBML and GMMs with mean-shift, provide a nearly identical distribution at both generations, and they converge to a distribution with modes centered on the points closest to the boundary.

For experiment 3, we ran GMMs with mean-shift estimation 30 times on the whole weighted data. The means of the modes at convergence were $(-0.01, 0.38)$ and $(0.01, -0.41)$, with a very small standard deviation of 0.03. The distribution at convergence was very close to those obtained in experiments 1 and 2. Interestingly, we observed that, when the weights were removed, the modes at convergence moved to $(-0.03, 0.51)$ and $(0.03, -0.49)$.

5.2 Weight Distribution Changes

One important property of boosting is to scale the weights of data as a function of its distance from the margin. To observe the effect of weight changes, in Figure 4 we plotted the weights of all points at different radii and for different generations for the circle dataset (see Figure 2). We can clearly see an exponential decay and a logistic increase based on the vicinity to the margin of the data. For positive points, when the radius is between 0.3 and 0.4, and for negative points, when the radius is between 0.4

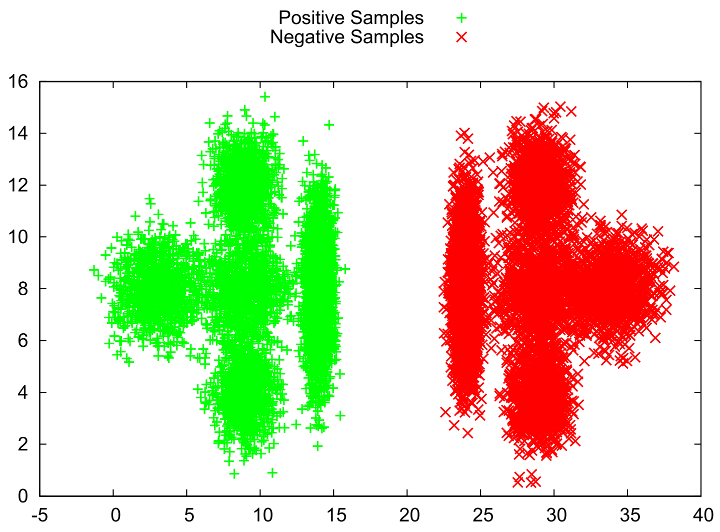

Figure 5: Bivariate Gaussian dataset.

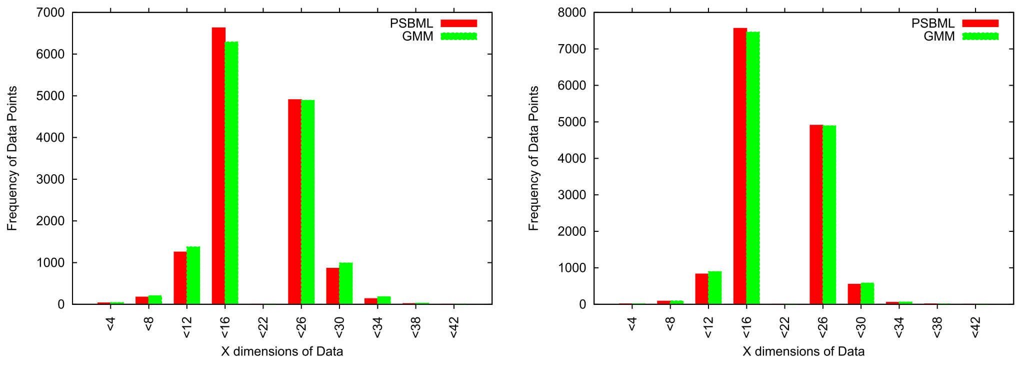

Figure 6: Linearly separable Gaussian dataset: data distribution at cycle 25 (Left) and 50 (Right) using PSBML and GMMs.

and 0.5, an increase is seen with time, and for the rest there is an exponential decay, confirming a behavior analogous to boosting.

5.3 Linearly Separable Bivariate Gaussians

The second dataset consists of 5 Gaussians for each class, with roughly the same density but different shapes (see Figure 5). The Gaussians with means (14, 8) and (24, 8) are the closest to the boundary, given by the line $x = 20$. They simulate the “global modes.” We again ran the three experiments described at the beginning of Section 5. The large margin classifier was simulated by estimating the average distance between the smallest positive and the largest negative instances.

Again we observed that the data distributions produced by PSBML and GMM with mean-shift and grid structure are very much alike, as illustrated in Figure 6. For experiment 3, with 30 runs on the weighted dataset, the modes of the data distribution

Table 1: Overlap percentage between support vectors and PSBML hard instances.

	2D Circle	2D Gaussians
SV overlap	90%	94%

converged to $(14.02, 7.89)$ and $(24.09, 7.88)$, with deviation of 0.002, matching exactly our results for experiments 1 and 2.

5.4 Hard Instances and Support Vectors

We also analyzed the data distribution at convergence by comparing the hard instances identified by PSBML with the support vectors of a trained SVM. Table 1 shows the percentage of overlap for the two simulated datasets. The support vectors of the trained SVMs with the highest α (i.e., weight) values correspond to the hard instances with the top 10% largest weights identified by the PSBML algorithm for both the datasets.

6 Empirical Analysis of the Performance of PSBML

We performed extensive experiments to investigate the feasibility of the proposed framework under different viewpoints. In particular, we studied the following crucial features: (i) sensitivity analysis against parameters; (ii) efficacy of PSBML as a meta-learning framework; (iii) scalability properties; and (iv) robustness against noise.

We ran all scalability experiments (where running times were measured) on a dual 3.33-GHz 6-core Intel Xeon 5670 processor with no hyperthreading. This means that we had a maximum of 12 hardware threads available. PSBML was implemented both as a single threaded Weka (Hall et al., 2009) classifier and as a multithreaded standalone Java program that could run on any JVM version above 1.5 (see Section 8). All experiments with PSBML were run using a maximum heap size of 8 GB and a number of threads equal to the number of nodes in the grid. All SVMs and boosting implementations, where running times were compared, used either the native Matlab or C++ code, except for AdaBoostM1, where Weka 3.7.1 was used. All statistical significance tests were performed using the Matlab paired t-test function with significance level set to 5%.

To measure accuracy we used the area under the receiver operating characteristic, or ROC, curve, widely used in machine learning and data mining. The curve is generated by plotting the true positive rate (TPR) on the y-axis against the false positive rate (FPR) on the x-axis at various threshold settings. The TPR measures the fraction of positive examples that are correctly classified. The FPR measures the fraction of negative examples that are misclassified as positive. An area of 1 represents a perfect test; an area of .5 represents a worthless test. For the unbalanced data we also measured the area under the Precision Recall Curve (PRC), as it has been shown to provide a better estimation of an algorithm's accuracy when data is highly skewed (Davis and Goadrich, 2006). The PRC is obtained by plotting recall on the x-axis and precision on the y-axis. Recall is the same as TPR, whereas precision measures the fraction of examples classified as positive that are truly positive.

6.1 Parameter Sensitivity Analysis

In this section, we investigate the effect that different parameter settings have on the behavior of PSBML, in terms of accuracy and rate of convergence. Specifically we

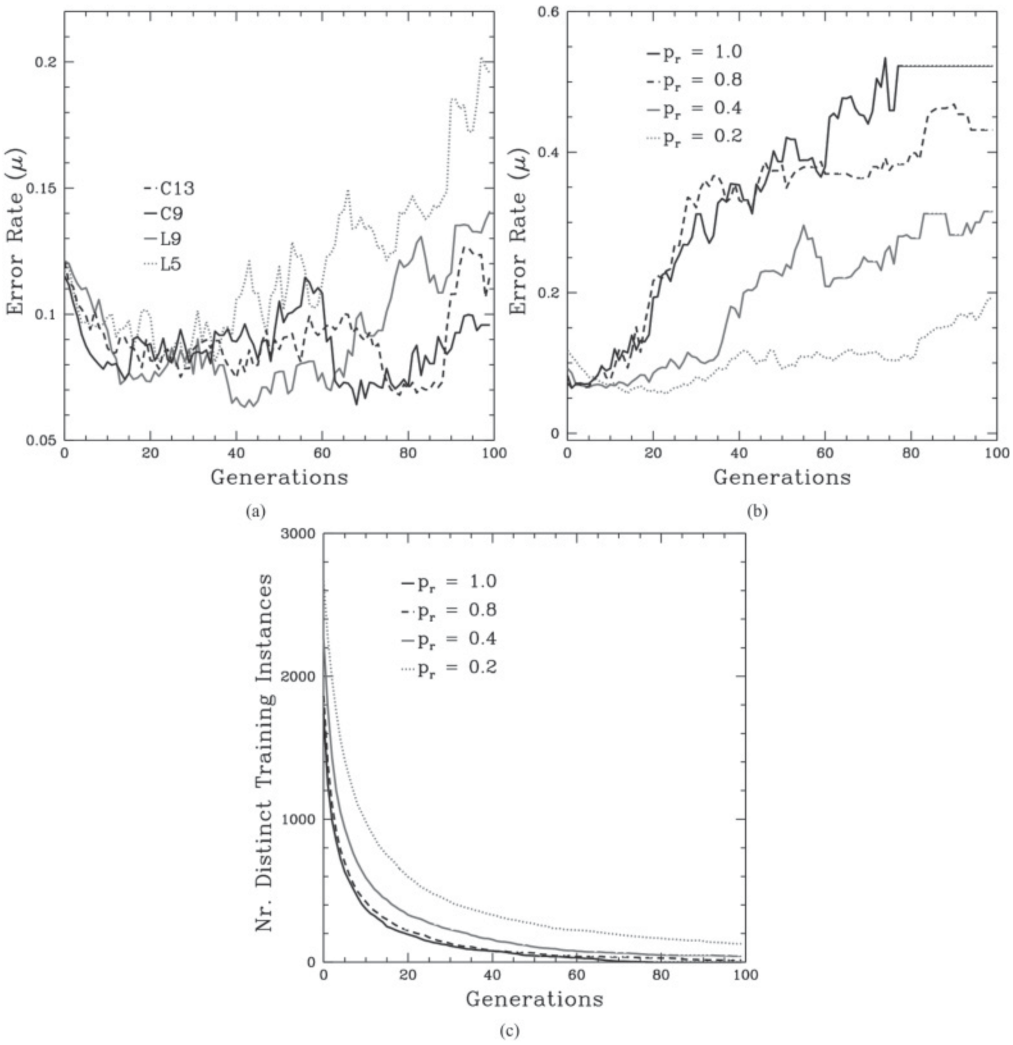

Figure 7: Sensitivity analysis results for the UCI chess dataset. (a) The error rate at successive generations is shown for different neighborhood structures. (b) The error rate at successive generations is shown for different P_r values. (c) The number of distinct instances sampled as successive generations is shown for different P_r values.

analyze the neighborhood structure, the rate of replacement P_r, and the grid size. To this end we use two datasets, UCI Chess and Magik. They offer different training data size scenarios: Chess is smaller with 3,196 instances, while Magik contains 17,116 instances.

To study the effect that the neighborhood structure of the grid has on the performance of PSBML, we ran experiments on the UCI Chess (King-Rook vs. King-Pawn) dataset; the dataset consists of 3196 instances, 36 attributes, and 2 classes. PSBML was run on this problem using various neighborhood structures, and the results are shown in Figure 7a. A 5 × 5 grid with a Naïve Bayes classifier with discretization for numeric features was used. PSBML was evaluated by combining the instances selected by all the

Table 2: ROC results for PSBML with different grid sizes.

Datasets	3×3	4×4	5×5	6×6	7×7
Chess	98.5	98.5	98.3	98.2	98.1
Magik	89.4	89.5	89.4	89.4	89.5

nodes at each generation; using this collection of instances, we trained a single classifier and tested its performance on the test set. Although the average size reduction of the training dataset was quite similar for all the neighborhoods, their classic "over fitting curves" were different (see Fig. 7a). The notion of *selection pressure* gives the degree to which only the highly weighted instances are selected at each generation. Since the sample selected at each node has a constant size, the selection pressure is driven by the size of the pool we choose the sample from. Furthermore, the more spread the neighborhood, the faster the highly weighted instances travel through the grid. As such, the neighborhoods L9 and C13 have a stronger selection pressure. They produced a more rapid initial decrease in test classification error rates, which subsequently increased more rapidly as the training data became too sparse. The simplest L5 neighborhood reduced classification error rates too slowly. The best results were obtained with the neighborhood structure C9.

We used the UCI Chess dataset to also investigate how the rate of replacement P_r affects the performance of PSBML. Figures 7b and 7c illustrate that increasing the value of P_r results in faster convergence rates, but also in less accurate models. The best results were obtained when $P_r = 0.2$.

Finally, to investigate the impact of the grid size on accuracy, we used both the Chess and the Magik datasets, which offer different training data size scenarios. The Magik dataset is larger than Chess, with 17,116 instances, 10 attributes, and 2 classes. We used the C9 neighborhood configuration and fixed the value of the replacement rate P_r to 0.2. We tested various grid sizes ranging from 3×3 to 7×7. We measured the area under the ROC curve for PSBML over 30 runs. The Naïve Bayes classifier (with the same configuration) was used at each node of the grid. Table 2 summarizes the results, showing that there is no statistically significant difference in ROC values across the various grid sizes.

These results are not surprising. Given the wraparound nature of the grid, and the diffusion of hard instances through the fitness-proportional selection process, only the rate of convergence to the margin is affected by the grid size. For the Chess dataset, which has only 3196 instances, as the number of nodes increases we observe a slight degradation in performance. As the number of nodes increases, the training data available at each node reduces significantly, and as a result the classifier's VC bound comes into effect (Vapnik, 1995). Thus, for smaller datasets, the choice of grid configuration may depend on this lower bound. With a larger dataset like Magik (17,116 instances), no degradation is observed. This is an important insight for the practitioner dealing with massive data, as scaling based on the number of hardware cores available can be used to configure the grid size.

6.2 Meta-Learning Experiments

The goal of this experiment is two-fold: first to validate that PSBML provides a general framework for meta-learning, and therefore can be used in combination with a variety

Table 3: UCI datasets used in the experiments.

	Adult	W8A	ICJNN1	Cod	Cover
# *Train*	32560	49749	49990	331617	581012
# *Test*	16279	14951	91701	59535	58102
# *Features*	123	300	22	8	54
# *Labels*	2	2	2	2	7

Table 4: Meta-learning results (ROC) comparing the base classifiers and PSBML combined with the same.

	Adult	W8A	ICJNN1	Cod	Cover
NB	90.1	94.30	81.60	87.20	84.90
PSBML	**90.69**	**96.10**	**81.79**	**91.79**	**87.31**
C4.5	88.01	**87.80**	94.60	95.90	99.50
PSBML	88.78	84.80	**97.30**	**97.24**	97.44
Linear SVM	54.60	80.20	64.60	88.80	72.20
PSBML	**60.01**	80.70	64.80	**95.10**	**79.10**

of learners; second, to verify that it's an effective parallel algorithm; that is, it provides accuracy results comparable to a sequential counterpart, while achieving a speedup. To illustrate this, we performed experiments using three base classifiers: Naive Bayes, Decision Trees (C4.5), and Linear SVMs (LibLinear v1.8) (the corresponding Weka implementations were used). We used five medium to large UCI datasets (Frank and Asuncion, 2010), commonly used for performance comparisons. Table 3 provides a description of the data. For each dataset, we normalized the features in the range [0,1], and converted multi-class problems to binary, using the one-vs-all strategy optimized for the LibSVM system, as described in (Fan et al., 2005). The PSBML algorithm was run with the C9 neighborhood, a 3×3 grid, a replacement probability of 0.2, 20 training cycles, and a validation set size of 10%. We first optimized the base classifiers for performance, and then used the optimized settings in PSBML. Naïve Bayes was used with the option of kernel estimation instead of using the default normal estimation; C4.5 was used with the default settings; and LibLinear was used with the L_2 loss function in both experiments. Each run, with the exception of Cover and C4.5, was repeated 30 times, and paired t-tests were used for statistical significance computation using the Area Under the Curve (AUC) (Bradley, 1997) as the metric. The experiments involving Cover and C4.5 were run only 10 times due to the long processing time. Hence, significance is not recorded in this case. Results are reported in Table 4. All statistically significant results are marked in boldface.

We observe that PSBML, combined with the Naïve Bayes classifier, performs statistically significantly better than the Naïve Bayes classifier itself on all the datasets. Similar results were observed, and theoretical insights were provided, with regular boosting and Naïve Bayes (Elkan, 1997). Another important result to note is that the ensemble effect of PSBML makes the accuracy of a linear SVM significantly better (in three cases), while parallelizing the LibLinear SVM, which was already optimized for speed.

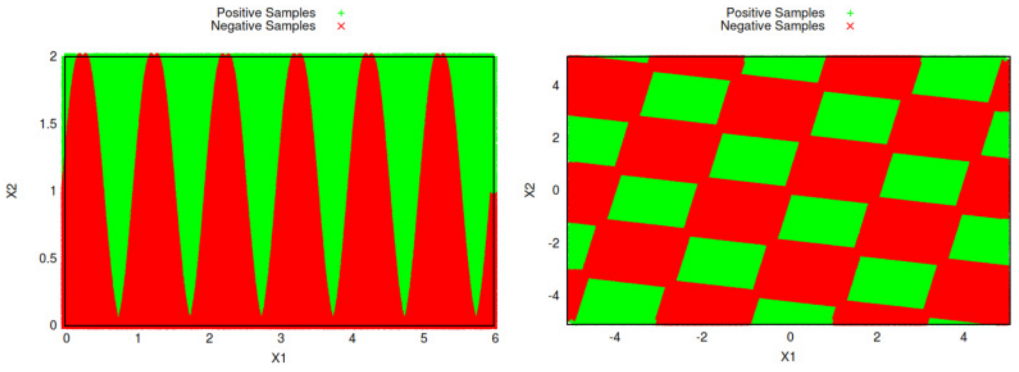

Figure 8: Synthetic datasets: (Left) Sine wave; (Right) Checkerboard.

6.3 Scalability Experiments

The goal of this experiment is to validate whether PSBML performs competitively against custom optimized learning algorithms, in terms of training time, as a measure of speed, and in terms of accuracy, as a measure of performance. PSBML shares an important feature with SVMs: it reduces the training data to the points which are close to the boundary. Thus, we compared PSBML with a number of SVM implementations: a fast Newton-based method, LP-SVM (Fung and Mangasarian, 2002), a structural optimization-based technique, SVM-PERF (Joachims, 1999) (linear because with an RBF kernel it crashed), the most commonly used LibSVM (Fan et al., 2005), a fast optimized LibLinear (Fan et al., 2008), a stochastic gradient-based approximation method, SGDT (Bottou and Bousquet, 2008), and fast ball enclosure-based BVM (Tsang et al., 2007). We also compared PSBML against a parallel AdaBoost algorithm (Favre et al., 2007) and the standard AdaBoostM1. All of the above-mentioned implementations of SVMs incorporate some form of custom changes to boost the speed, such as incremental sampling of the dataset, or simplifying the quadratic optimization, or assuming linearly separable data.

For this experiment, we used two datasets that present complex and highly nonlinear patterns, and are widely used in the literature to test the bias induced by data sampling (Elwell and Polikar, 2011; Cervantes et al., 2011). The first dataset was a two-dimensional decision boundary based on a sine wave generated by the function $f(\mathbf{x}) = 2sin(2\pi x_1)$ (see Figure 8). The dimension x_1 was sampled from the interval $[0, 6.28]$ and the $y = f(\mathbf{x})$ dimension was randomly sampled from the interval $[0, 2]$. The second dataset is a 4×4 rotated checkerboard data with alternate positive and negative classes as shown in Figure 8. Each dataset has one million instances, and all the experiments were repeated 30 times. We measured training time for each of the runs, and the average training time is reported. Ten-fold cross-validation was performed for accuracy and the average accuracy is reported. Each algorithm was tuned to some level of optimality for comparisons; that is, the soft margin parameter and the radius of the RBF kernel for SVMs were optimized using a grid search in the intervals $[-5,15]$ and $[3,-15]$, respectively.

The PSBML algorithm was run with the C9 neighborhood, a 3×3 grid, replacement probability of 0.2, 10 training cycles, and a validation set size of 10% for each training fold. The C4.5 classifier with default parameters was used as it had an intermediate

Table 5: Training speed (in seconds) and accuracy for the Checkerboard and the Sine Wave datasets.

	Checkerboard		Sine Wave	
Algorithm	Speed	Acc	Speed	Acc
SVM				
LP-SVM (Linear)	44.20	50.23	33.20	68.80
LP-SVM (RBF)	33.20	57.11	105.56	70.11
LibLinear	133.20	50.08	203.12	68.60
SGDT (10 iterations)	4.20	54.49	4.20	54.89
SVM-PERF (Linear)	1.10	51.01	2.01	61.90
BVM (RBF)	1.80	50.03	1.20	49.03
LibSVM (RBF, 0.1%)	136.20	98.20	423.23	70.80
Boosting				
AdaBoostM1	38.21	51.25	30.71	74.25
ParallelAdaBoost (9 threads, 10 iterations)	17.90	51.22	13.90	78.30
PSBML				
PSBML (C4.5)	123.10	**99.49**	193.10	**99.56**

training speed between the fast LibLinear and the kernel estimated Naïve Bayes. Results are shown in Table 5.

We emphasize that some of the difference in running times may be explained by the language choice, library choice, programming quality, etc. But looking at the numbers in Table 5, it is clear that there is more going on here. In fact, the differences in speed shown by the methods implemented in C++ (SGDT, SVM-PERF, BVM, and ParallelAdaBoost) vs. those implemented in Java (AdaBoostM1, LibLinear, LibSVM, and PSBML) are well beyond the difference just due to the native language used. For example, SGDT (C++) is 9 to 31 times faster than any of the methods implemented in Java.

For both the synthetic datasets, PSBML gives the most accurate results with respect to the methods that have comparable training speed (i.e., LibLinear and LibSVM). Most of the techniques customized for high speed give poor accuracy results. The synthetic datasets, being highly nonlinear, exaggerate the trade-offs implemented by the algorithms.

6.3.1 Asynchronous PSBML

We have implemented an asynchronous version and a GPU-based (also asynchronous) version of PSBML. PSBML in its standard form is synchronous: nodes wait for neighbors to complete before continuing. We were interested in the impact on performance and speedup due to asychrony. To do this, we implemented an asynchronous version of PSBML (called PSBML Async) where each node doesn't wait for its neighbors, but rather uses their current state (i.e., current choice of local data). We were also interested in the speedups made possible by GPUs, which have been used extensively to implement parallel models (Steinkraus et al., 2005; Srinivasan et al., 2010). To this end, we implemented a simplified version of PSBML (called PSBML GPU) which used APARAPI[1] to convert the Java bytecode into OpenCL (Stone et al., 2010) ready to execute on a GPU.

[1]http://developer.amd.com/tools-and-sdks/opencl-zone/aparapi/

Table 6: ROC results for PSBML Async and PSBML GPU.

	Adult	W8A	ICJNN1	Cod	Cover
PSBML Sync	90.69	96.1	81.79	91.79	87.31
PSBML Async	90.23	96.7	81.1	90.9	87.82
PSBML GPU	89.91	96.03	82.2	90.21	87.72
Async Speedup	0.17	0.19	0.42	0.15	0.18
GPU Speedup	3.3	4.3	8.2	4.1	3.7

We used the same UCI datasets as before (see Table 3) and ran PSBML Async and PSBML GPU using Naïve Bayes as the underlying classifier. We implemented a Naïve Bayes classifier for the GPU as the WEKA version was poorly optimized for a GPU environment. The GPU implementation was run on a Radeon 240 Graphics card AMD machine. We ran the experiments with the same distribution for training and testing and performed 30 runs. Statistical significance was measured using a paired *t*-test with p-value of 0.05. Table 6 shows that PSBML Async and PSBML GPU had similar accuracy to synchronous PSBML, and there was no statistically significant difference in ROC areas across all three methodologies. We note that the speedup achieved by PSBML Async was 15% to 42% over the synchronous runs. The GPU speedup was much higher, ranging from 3.3 to 8.2 times faster. These results indeed show that by implementing PSBML in an asynchronous mode, and by using base classifiers optimized for GPUs, a huge performance gain in speedup can be achieved without sacrificing accuracy.

6.3.2 A Real-World Dataset

The KDD Cup 1999 intrusion detection dataset (del Rio et al., 2014) was used to compare the performance of the algorithms. The dataset contains 4,898,431 training instances. The problem was converted into a binary classification problem because many SVM implementations did not support multiclass labels. The feature set was also scaled within the range [0,1], which improved the performance of many SVMs almost 10 times. The PSBML algorithm was run with the C9 neighborhood, a 3×3 grid, replacement probability of 0.2, 10 training cycles, and a validation size of 0.1% of the training data. C4.5 was used with default parameters again for the same reasons mentioned earlier.

In previous work, it was noted that many algorithms have a very similar error rate on this dataset. Hence, the number of mis-classifications was suggested and used as comparison metric (Yu et al., 2003). We do the same here. In addition, we measure the areas under the ROC and under the Precision Recall Curve (PRC), since the dataset is unbalanced. Each of the experiments was run 30 times, except the AdaBoostM1 (only 10 times) due to large training time. The mean training times and the mean mis-classification averages are reported in Table 7. Some of the algorithms, for example, LP-SVM, couldn't run with a 12-GB RAM machine, because the loading of the data matrix itself failed. Also, for SGDT and BVM we couldn't compute the output probabilities to measure ROC and PRC due to the kernel choice. We observe that most algorithms that were optimized for speed had to trade off accuracy. Also, the training time of LibSVM increased considerably when the sampled data went from 1% to 10%, with a small change in classification rate. The ROC value for PSBML was statistically significantly better; the value of the PRC area was comparable to that of SVM-PERF. In

Table 7: Training speed in secs, misclassification, area under ROC, and PRC for the KDD Cup 1999 dataset.

Algorithm	Speed	MisClass	ROC	PRC
SVM				
LibLinear	80.20	25447.3	94.4	6.3
LibSVM (RBF, 1%)	90.20	25517.8	94.1	76.9
LibSVM (RBF, 10%)	1495.20	25366.1	94.1	13.1
SGDT (10 iterations)	211.10	121301	-	-
SVM-PERF (linear)	4.90	25877.1	93.1	90.3
BVM (RBF)	**3.20**	25451.3	-	-
Boosting				
AdaBoostM1	13296.42	190103.3	88.4	17.2
ParallelAdaBoost (9 threads, 10 iterations)	202.30	26170.2	36.2	70.2
PSBML				
PSBML(C4.5)	2913.10	**20898.8**	**95.6**	91.2

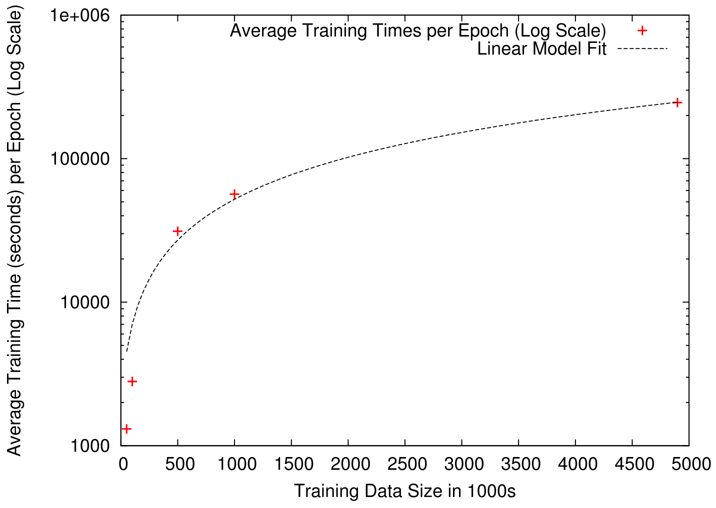

Figure 9: Mean training times per cycle with varying dataset sizes.

conclusion, PSBML, while working on the entire dataset, finds a good classification rate at a considerable performance speed.

To see the impact of data sizes on PSBML, we also selected training samples of various sizes from $50K$, $100K$, $500K$, to one million. Ten runs were performed with standard PSBML with decision trees, a 3×3 grid, and the C9 neighborhood. Nine threads were used in this experiment. Training time (log scale) is plotted against data size in Figure 9. The graph clearly shows a steady linear scaling with data size. To see the impact of the multicore processor described above on scalability, we changed the number of threads and computed the corresponding average training times. The result

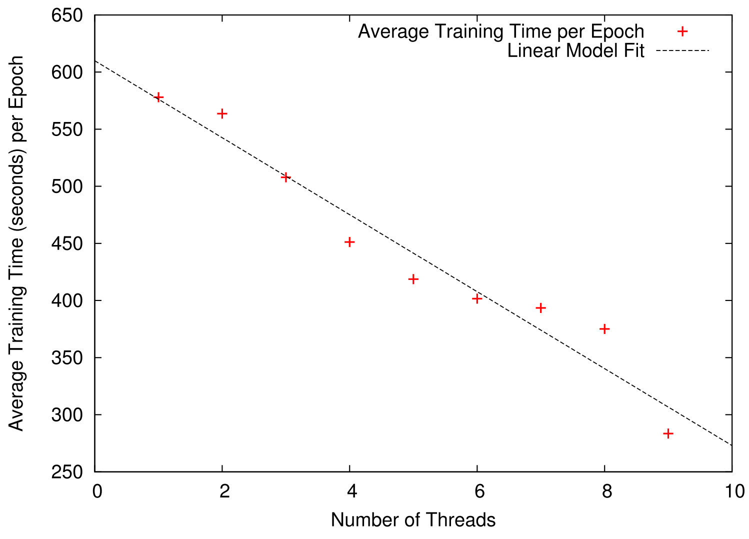

Figure 10: Mean training times per cycle with varying threads.

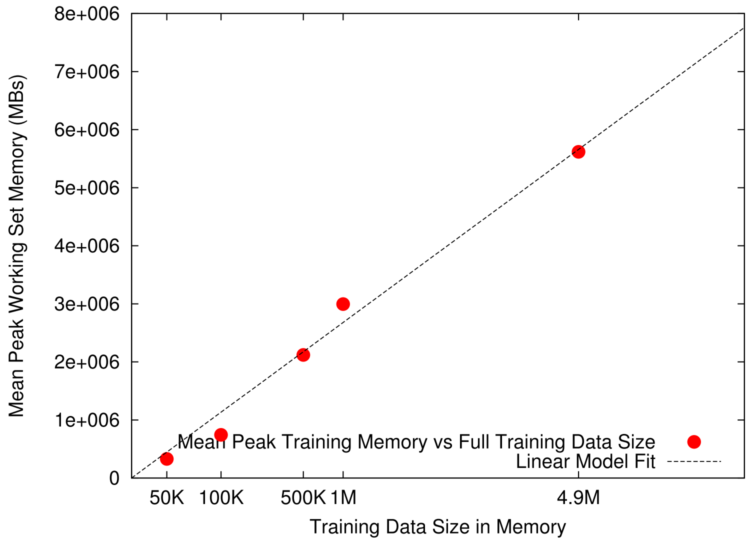

Figure 11: Mean peak working set memory with varying dataset sizes.

is given in Figure 10, which shows again a consistent linear improvement with the number of threads.

Another important aspect of a large-scale learning algorithm is memory requirements. To evaluate this impact, we measured the memory usage with varying data sizes. We used the same data sizes and configuration as in the previous experiments. Figure 11 shows the mean peak working memory during training as a function of different training data sizes. This again shows a linear increase with the training data size. Thus, the memory space complexity of PSBML appears to be $O(n)$, where n is the size

Table 8: Performance of AdaBoostM1 (DS: Decision Stump), AdaBoostM1 (NB: Naïve Bayes) and PSBML (NB: Naïve Bayes) with no, 10%, and 20% noise.

	Adult	W8A	ICJNN1	Cod	Cover
No Noise					
AdaBoostM1/DS	87.10	77.80	**93.40**	92.80	75.70
AdaBoostM1/NB	87.20	93.30	84.30	**95.70**	85.30
PSBML/NB	**90.69**	**96.10**	81.79	91.79	**87.31**
10% Noise					
AdaBoostM1/DS	85.70	58.90	**92.82**	92.20	75.10
AdaBoostM1/NB	85.80	83.40	79.80	**95.10**	85.10
PSBML/NB	**90.46**	**96.01**	77.46	88.06	**87.14**
20% Noise					
AdaBoostM1/DS	85.10	57.10	**92.30**	92.10	75.10
AdaBoostM1/NB	84.88	79.01	79.70	**94.90**	84.20
PSBML/NB	**90.10**	**95.97**	77.42	86.98	**87.11**

of the training set. In comparison, SVMs are $O(n^2)$ (Tsang et al., 2005). This result shows a key advantage.

6.4 Comparison against AdaBoost and Impact of Noise

Here we compare PSBML against AdaBoost and test the robustness in presence of noise. Previous work found that boosting is more susceptible to noise as compared to other ensemble methods like bagging and stacking (Melville et al., 2004; Long and Servedio, 2008). We added noise to the class labels by randomly changing different percentages of labels. We used AdaBoostM1 both with decision stumps and with Naïve Bayes (optimized using kernel estimators), and compared it against PSBML combined with the same underlying Naïve Bayes classifier. PSBML was used with the default C9 neighborhood, replacement probability of 0.2, and validation set of 10%.

We used the same datasets used for the meta-learning experiments, and did the same preprocessing. We performed 30 runs to compare the three algorithms without noise, and in presence of 10% and 20% of noise. The results are shown in Table 8. Statistically significant results are highlighted in boldface.

In absence of noise, PSBML with Naïve Bayes performs significantly better than AdaBoostM1 with decision stumps or with the same optimized Naïve Bayes in three of the five datasets. To measure how robust a method is across all the datasets, we compute the following quantity: $impact = \frac{1}{N}\sum_{i=i}^{N}(\overline{auc}^{i}{}_{\textit{no-noise}} - \overline{auc}^{i}{}_{noise})$, where N is the number of datasets. The smaller the value of the impact is for an algorithm, the more robust that method is on average.

The impact values of AdaBoostM1 (DecisionStump), AdaBoostM1 (Naïve Bayes), and PSBML (Naïve Bayes) with 10% noise are 4.41, 3.32, and 1.71, respectively. Similarly, with 20% noise the impact values for these algorithms are 5.02, 4.62, and 2.02, respectively. This shows that the PSBML algorithm is more robust to noise as compared to standard boosting. This is likely due to two reasons. First, in PSBML, the weighted sampling procedure is driven by the confidence of predictions only (prediction errors are not used), while AdaBoost credits larger weights to instances which are erroneously predicted. Second, PSBML makes use of a validation set to estimate the best classifier to be used for prediction of test instances, thus preventing overfitting.

7 Summary and Conclusions

The PSBML algorithm described and analyzed in this article directly addresses the difficulties that machine learning algorithms have in scaling up to increasingly larger data sets. Rather than requiring subsampling of the data or algorithmic modifications to support some form of parallelism, PSBML provides a meta-learning framework that uses existing learning algorithms and scales effectively with increasing dataset size. This is achieved by combining concepts from spatially structured evolutionary algorithms (SSEAs) with concepts from ensemble and boosting methodologies.

We have presented both theoretical and empirical analyses which show that PSBML preserves a critical property of boosting, specifically, convergence to a distribution centered around the margin. We then presented additional empirical analyses showing that this meta-level algorithm provides a general and effective framework that can be used in combination with a variety of learning classifiers. We performed extensive experiments to investigate the trade-off achieved between scalability and accuracy, and robustness to noise on both synthetic and real-world data. These empirical results corroborate our theoretical analysis, and demonstrate the potential of PSBML in achieving scalability without sacrificing accuracy.

The meta-learning experiments have shown that PSBML exhibits characteristics similar to that of AdaBoost in the sense that adding ensemble boosting to a standard ML classifier produces at least comparable and often better results. Scalability experiments confirm that while maintaining good running times for training, the accuracy is not compromised. We have also shown a steady linear improvement in speed with an increasing number of threads, as well as linear training time and linear memory use as a function of data size. In addition, the spatially structured aspects of PSBML provide a resilience to noise, an important feature for real-world applications.

There are several immediate extensions to this work. We are now adapting the algorithm to semi-supervised learning and unsupervised learning. In addition, we are exploring the possibility of mapping PSBML onto distributed architectures like the Beowulf-style clusters in combination with map-reduce algorithms.

8 Software and Data

Software, data, and parameters used to perform the experiments in this article are available at https://sites.google.com/site/psbml2013/ and at https://sites.google.com/site/psbml2016/ under an academic license.

References

Alba, E., and Luque, G. (2004). Growth curves and takeover time in distributed evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 864–876.

Bennett, K. P., Demiriz, A., and Maclin, R. (2002). Exploiting unlabeled data in ensemble methods. In *Proceedings of Knowledge Discovery and Data Mining*, pp. 289–296.

Bottou, L., and Bousquet, O. (2008). The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), *Advances in neural information processing systems*, pp. 161–168. NIPS Foundation. Retrieved from http://books.nips.cc.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159.

Carreira-Perpiñán, M. (2005). Gaussian mean shift is an EM algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:2007.

Carreira-Perpiñán, M., and Williams, C. K. I. (2003). On the number of modes of a Gaussian mixture. In *Proceedings of Scale-Space 2003*, pp. 625–640.

Cervantes, J., Lopez, A., Garcõa, F., and Trueba, A. (2011). A fast SVM training algorithm based on a decision tree data filter. In *Mexican International Conference on Artificial Intelligence*, pp. 187 –197.

Chang, E. Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., and Cui, H. (2007). Parallelizing support vector machines on distributed computers. *Neural Information Processing Systems*, 17:521–528.

Chu, C. T., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. (2007). Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*, pp. 281–288. Cambridge, MA: MIT Press.

Davis, J., and Goadrich, M. (2006). The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 233–240.

del Rio, S., Lopez, V., Benitez, J. M., and Herrera, F. (2014). On the use of mapreduce for imbalanced big data using random forest. *Information Sciences*, 285:112–137.

Drost, I., Dunning, T., Eastman, J., Gospodnetic, O., Ingersoll, G., Mannix, J., Owen, S., and Wettin, K. (2010). Apache Mahout. Apache Software Foundation. Retrieved from http://mloss.org/software/view/144/.

Elkan, C. (1997). Boosting and naive Bayesian learning. In *Proceedings of Knowledge Discovery and Data Mining*, pp. 125–135.

Elwell, R., and Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005). Working set selection using the second order information for training SVM. *Journal of Machine Learning Research*, 6(1532-4435):1889–1918.

Favre, B., Hakkani-Tür, D., and Cuendet, S. (2007). icsiboost. Retrieved from http://code.google .come/p/icsiboost.

Frank, A., and Asuncion, A. (2010). UCI machine learning repository. Retrieved from http://archive.ics.uci.edu/ml.

Freund, Y., and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the International Conference on Machine Learning*, pp. 148–156.

Fung, G., and Mangasarian, O. L. (2002). A feature selection Newton method for support vector machine classification. Technical Report 02-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin.

Gallant, S. (1990). Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191.

Giacobini, M., Tomassini, M., Tettamanzi, A., and Alba, E. (2005). Selection intensity in cellular evolutionary algorithms for regular lattices. *IEEE Transactions on Evolutionary Computation*, 9(5):489–505.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18.

Joachims, T. (1999). Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola (Eds.), *Advances in kernel methods*, pp. 169–184. Cambridge, MA: MIT Press.

Kamath, U., Domeniconi, C., and Jong, K. A. D. (2013). An analysis of a spatial ea parallel boosting algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1053–1060.

Kamath, U., Kaers, J., Shehu, A., and De Jong, K. A. (2012). A spatial EA framework for parallelizing machine learning methods. In *Parallel Problem Solving from Nature*, pp. 206–215.

Long, P. M., and Servedio, R. A. (2008). Random classification noise defeats all convex potential boosters. In *International Conference on Machine Learning*, Vol. 307, pp. 608–615. ACM.

Melville, P., Shah, N., Mihalkova, L., and Mooney, R. J. (2004). Experiments on ensembles with missing and noisy data. In *Proceedings of the Workshop on Multi Classifier Systems*, pp. 293–302.

Rasmussen, C. E. (2000). The infinite Gaussian mixture model. In *Advances in neural information processing systems*, pp. 554–560. Cambridge, MA: MIT Press.

Sarma, J., and De Jong, K. (1996). An analysis of the effects of neighborhood size and shape on local selection algorithms. In *Parallel Problem Solving from Nature*, pp. 236–244.

Sarma, J., and De Jong, K. (1997). An analysis of local selection algorithms in a spatially structured evolutionary algorithm. In *Proceedings of the International Computer Games Association*, pp. 181–187.

Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686.

Schapire, R. E., and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336.

Shafer, J., Agrawal, R., and Mehta, M. (1996). Sprint: A scalable parallel classifier for data mining. In *Proceedings of the 22nd International Conference on Very Large Databases*, pp. 544–555.

Srinivasan, B. V., Hu, Q., and Duraiswami, R. (2010). GPUML: Graphical processors for speeding up kernel machines. In *Workshop on High Performance Analytics—Algorithms, Implementations, and Applications, Siam Conference on Data Mining*.

Steinkraus, D., Buck, I., and Simard, P. Y. (2005). Using GPUs for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition*, pp. 1115–1120.

Stone, J. E., Gohara, D., and Shi, G. (2010). Opencl: A parallel programming standard for heterogeneous computing systems. *IEEE Design & Test of Computers*, 12(3):66–73.

Svore, K., and Burges, C. (2011). *Large-scale learning to rank using boosted decision trees*. Cambridge: Cambridge University Press.

Tomassini, M. (2005). *Spatially structured evolutionary algorithms: Artificial evolution in space and time*. Natural Computing Series. Berlin: Springer.

Tsang, I. W., Kocsor, A., and Kwok, J. T. (2007). Simpler core vector machines with enclosing balls. In *Proceedings of International Conference on Machine Learning*, pp. 911–918.

Tsang, I. W., Kwok, J. T., and Cheung, P. (2005). Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392.

Vapnik, V. (1995). *The nature of statistical learning theory*. Berlin: Springer.

Woodsend, K., and Gondzio, J. (2009). Hybrid MPI/OpenMP parallel linear support vector machine training. *Journal of Machine Learning Research*, 10:1937–1953.

Yu, H., Yang, J., and Han, J. (2003). Classifying large data sets using SVMs with hierarchical clusters. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 306–315.