An Analysis of a Spatial EA Parallel Boosting Algorithm

Uday Kamath
Dept. of Computer Science
George Mason University
Fairfax, USA
ukamath@gmu.edu

Carlotta Domeniconi
Dept. of Computer Science
George Mason University
Fairfax, USA
carlotta@cs.gmu.edu

Kenneth A. De Jong
Dept. of Computer Science
George Mason University
Fairfax, USA
kdejong@gmu.edu

ABSTRACT

The scalability of machine learning (ML) algorithms has become a key issue as the size of training datasets continues to increase. To address this issue in a reasonably general way, a parallel boosting algorithm has been developed that combines concepts from spatially structured evolutionary algorithms (SSEAs) and ML boosting techniques. To get more insight into the algorithm, a proper theoretical and empirical analysis is required. This paper is a first step in that direction. First, it establishes the connection between this algorithm and well known density estimation and mixture model approaches used by the machine learning community. The paper then analyzes the algorithm in terms of various theoretical and empirical properties such as convergence to large margins, scalability effects on accuracy and speed, robustness to noise, and connections to support vector machines in terms of instances converged to.

Track: Genetic Based Machine Learning

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*[Knowledge acquisition]

General Terms

Algorithms

Keywords

spatially structured evolutionary algorithms, machine learning, ensemble learning, boosting.

1. INTRODUCTION

The most common applications of machine learning (ML) involve supervised learning, where a learned model is induced from training data and is used for prediction of unseen future instances. The scalability of these ML algorithms has become increasingly important for many applications like real-time systems, bioinformatics, and web data mining, where the training dataset sizes are in millions or even larger. Many current ML techniques have scalability issues, either because they require the entire training set to be in memory for learning the model, or because the learning time during model induction grows exponentially with training set size, or both [6]. Basic solutions like reducing the size of the training datasets via sampling can be used, but they can introduce sampling errors [4]. Another approach is to employ complex changes specific to the desired ML algorithm for running on parallel and distributed architectures [9]. Boosting techniques are designed to boost training instances that are hard, but do so either by making multiple passes over the training data [13] or via incremental methods [10], affecting time and space complexity for large datasets.

To develop a solid general framework that tackles the scalability issue of supervised ML techniques, in this paper we analyze an alternative approach, that combines concepts from spatially structured evolutionary algorithms (SSEAs) [24, 29] for parallelization with the techniques of ensemble learning and boosting [19, 27]. This parallel spatial boosting algorithm (PSBML) has been shown to be a general approach for parallelizing ML methods [16]. It achieves this goal by partitioning the training data and by parallelizing the boosting step. The algorithm learns from the entire training data without changing the underlying learning techniques, and yet performs comparably to standard boosting and ensemble methods in terms of prediction accuracy.

In this paper, we show an equivalence between PSBML and well-established machine learning properties. Drawing this connection enables us to derive important properties of the PSBML method, and to achieve a better understanding of the extent of its applicability and its limitations. Theoretical models to study the distributions generated by SSEAs have been previously developed [24, 29]. Here we deal with an SSEA that runs an EA and a machine learner at each node of its grid. Each learner has access to only locally distributed subsets of training data (the current population of the local EA) and identifies the hardest instances to be classified. That information is used by the EA to define the fitness of training instances in terms of their classification difficulty, and then redistribute locally the most difficult instances via fitness proportionate selection. This parallel process causes an iterative change of the underlying training data distribution to increase the frequency of the difficult instances (as in boosting).

In this paper, we draw a clear connection between this procedure and statistical techniques that aim to find the

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

modes (i.e. local maxima) of Gaussian mixture models. By virtue of this connection, we can show that PSBML converges to a distribution whose modes are centered around the *margin*, i.e. around the hardest points as defined by the classification problem at hand. As a consequence, the PSBML algorithm preserves the main feature of boosting: convergence to the hard-to-classify instances.

In addition, we perform an empirical study to investigate the scalability of PSBML in terms of speed and accuracy, a crucial aspect of any parallel algorithm from a practitioner standpoint. Finally, we conduct an empirical analysis of the effects of noise on the parallel boosting framework as implemented in PSBML.

2. RELATED WORK

SSEAs have been theoretically analyzed using diffusion models [1, 14, 25]. 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 [24, 29]. Some of these findings constitute an important foundation for our theoretical and empirical analysis.

In statistical learning theory, a formal relationship between the notion of *margin* and the generalization classification error has been established [30]. As a result, classifiers that converge to a large margin have better performance in terms of generalization error. One of the most popular examples of such classifiers is support vector machines (SVMs). A trained SVM satisfies the maximum margin property, i.e. the decision boundary it represents has the maximum minimum distance from the closest training point. In addition, a formal analysis of the AdaBoost technique, and many of its variants, has derived theoretical bounds on the margin distribution to which the approach converges [26].

The AdaBoost technique, and boosting in general, is an example of a learning methodology known as ensemble learning [19], in which multiple classifiers are generated and combined to make the final prediction. Ensemble learning has been shown to be effective with unstable classifiers, by improving upon the performance of the base learners. Semi-supervised ensemble methods have also been introduced that select the collection of learners that make consistent prediction on the unlabeled data. It has been shown that this approach is equivalent to maximizing the margin in function space of both the labeled and unlabeled data [5]

Estimation of Distribution Algorithms (EDAs) is a well known methodology in the EA community that maps evolutionary algorithms to statistical distribution models [17, 18]. Various previous studies have established relationships between evolutionary algorithms and multivariate distribution models [2, 23].

3. THE PSBML FRAMEWORK

The algorithm for parallelizing boosting (PSBML) has at its core a standard SSEA [24], where training instances are distributed over a two-dimensional toroidal grid with a common machine learning algorithm running at each node of the grid. A node in the grid, as in SSEA, interacts with only its neighbors as predefined by the neighborhood structure (see Figure 1). From Boosting perspective, the balance between maintaining the diversity of instances in the training set and eliminating easy instances is very important. Selection

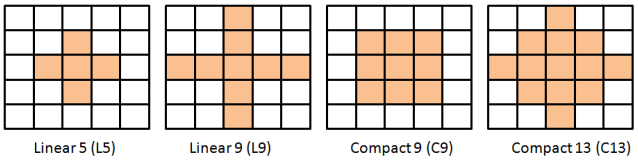

Figure 1: Two dimensional grid with various neighborhood structures.

pressure in SSEA performs this task and it depends on two elements: the survival selection method used by the EA in the node of the grid, and the size and shape of the neighborhood structure defined for the grid.

Each node in the grid starts with a uniformly stratified distribution of the training data. The local EA running in a node updates its training data during the generational cycle. Each node of the grid also has a pluggable ML classifier (e.g., a naive Bayes classifier, a decision tree learner, or support vector machine). During each generation, a node in the grid performs a standard training procedure using its own data, and performs testing using the combined training examples of neighboring nodes. In addition to classifying the test examples, the models output a confidence value for each prediction. These confidence values are used to assign a fitness to the instance, allowing the local EA in the node to subsequently select the most difficult instances for the next generation. Since each instance is a member of the neighborhood of multiple nodes, an ensemble assessment of difficulty is performed, similar to boosting the margin in AdaBoost [26]. Namely, the fitness v_i of an instance i is set equal to the smallest confidence value obtained from any node and for any class:

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

where N_i is a set of indices defined over the neighborhoods to which instance i belongs, and c_{ni} is the confidence credited to instance i by the learner corresponding to neighborhood n. A normalized weight v_i^{norm} with value in the interval [0,1] is obtained from v_i through linear re-scaling:

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

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

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

w_i is used in fitness proportionate 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 deterministically replacing the entire pool of local data at a given generation, a replacement probability P_r was introduced. Experimentally, it was determined that overlapping-generation model for the local EAs was more effective and the most effective value for P_r was found to be 0.20 [16] which we will be using in all the experiments.

The pseudo-code of PSBML is given in Algorithm 3.1. Various parameters for grid configuration, such as width/height, replacement probability, and maximum iterations (or epochs), are all included in *GridParam*.

Algorithm 3.1: PARBOOST($Train, Test, GridParam$)

```
INITIALIZEGRID(Train, GridParam)
comment: distribute the instances over the nodes in grid
currentMin ← 100
Pr ← GridParam.pr
comment: Probability of replacement
for i ← 0 to GridParam.iter
  do
hardData ← Train
comment: hardData initialized to all train
{
  TRAINNODES(GridParam)
  comment: Train all nodes
  TESTANDWEIGHNODES(GridParam);
  comment: Test using neighborhood and assign weight
  PrunedData ← {}
  for j ← 0 to GridParam.nodes
    do
    {
      NeighborData ← COLLECTNEIGHBORDATA(j);
      NodeData ← NodeData + NeighborData
      ReplaceData ← FITNESSPROPSEL(NodeData, Pr)
      PrunedData ← UNIQUE(PrunedData, ReplaceData)
      comment: Unique keeps 1 copy of instance in set
    }
  error ← TESTCLASSIFIER(PrunedData, Test)
  comment: Perform Validation step for the model
  if error < currentMin
    then {
      currentMin ← error
      bestClassifier ← SAVECLASSIF(PrunedData)
      hardData ← PrunedData
      comment: hardData set reduced
    }
}
return (bestClassifier, hardData)
```

4. ANALYSIS OF PSBML

We utilize Gaussian Mixture Models (GMMs) combined with mean-shift to show that the PSBML algorithm iteratively changes the underlying data distribution, and converges to a distribution whose modes are centered around the margin, i.e. around the hardest points as defined by the classification problem at hand.

4.1 Background on Gaussian Mixture Models and Mean-shift

A Gaussian mixture model (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 fit the data [21]. Gaussian mixtures have been widely used to model clustering applications, to classifying complex data, and segmenting images [22].

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 [8]. 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 [7]. Mean-shift can be used as a nonparametric clustering approach, and has been successfully applied in many computer vision problems [8]. The key advantage of using the mean-shift algorithm on a given density for mode finding is two-fold: (1) The approach is deterministic and nonparametric, since based on kernel density estimation; and (2) It poses no a priori assumptions on the number of the modes [7, 8].

4.2 Connection between GMMs with Mean-shift and PSBML

The PSBML algorithm, through the fitness proportionate selection process driven by the replacement probabilities associated to instances, effectively performs, locally at each node, a change of the underlying data distribution. Such local changes are reflected in an overall shift of the distribution. We argue here that the iterative process of changing the underlying data distribution converges to a distribution whose modes are centered around the margin, i.e. around the hardest points as defined by the classification problem at hand. In virtue of this argument we'll be able to demonstrate interesting properties of the PSBML algorithm at convergence. Our argument makes use of GMMs combined with the mean-shift procedure.

The mean-shift procedure is used in Gaussian mixture models to find the modes of a given distribution, irrespective of the starting location [7]. As we'll see, this is a desirable property that maps nicely to the PSBML framework, thus providing convergence properties which are independent of the sampling process used to distribute the data through the nodes of the grid.

In a classification problem, 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. Each node of the grid, along with its neighborhood structure, can be visualized as a sample, where each point is weighed according to its hardness, and on which a distribution can be fit. The underlying classifier trained on a node, and tested on the whole neighborhood, plays the role of estimating the distance of the corresponding points from the boundary, and weighting them accordingly. If we fit a Gaussian mixture model using mean-shift on the data assigned to a node and on its neighbors, at a given instance of time, it can be argued that the modes closer to the boundary, corresponding to points with larger weights, will be the ones visited by mean-shift, irrespective of the starting point. Since each node in the wraparound grid behaves similarly, in this scenario every node, in parallel, fits a Gaussian mixture model on its neighborhood. At each successive iteration of learning, sampling is performed at each node according to the new distribution, and therefore it's biased towards the peaks previously found, and instances are re-weighted based on the current estimation of the boundary. Thus, inductively, aggregating the local behavior of each node, it can be argued that the whole data distribution would converge to a distribution with peaks around the boundary, since the points at this location hold larger weights. Our findings through an empirical analysis, presented in Section 5, corroborate this argument.

From the above discussion, it can be seen that the mean-shift procedure, which is an iterative search algorithm used to locate the modes of a distribution, is similar to a local EA running on a node of the grid. The local classifier assigns confidence to data based on the current estimated boundary. The confidence acts as an instance weight, interpreted as a fitness by the EA, where the harder instances are the most

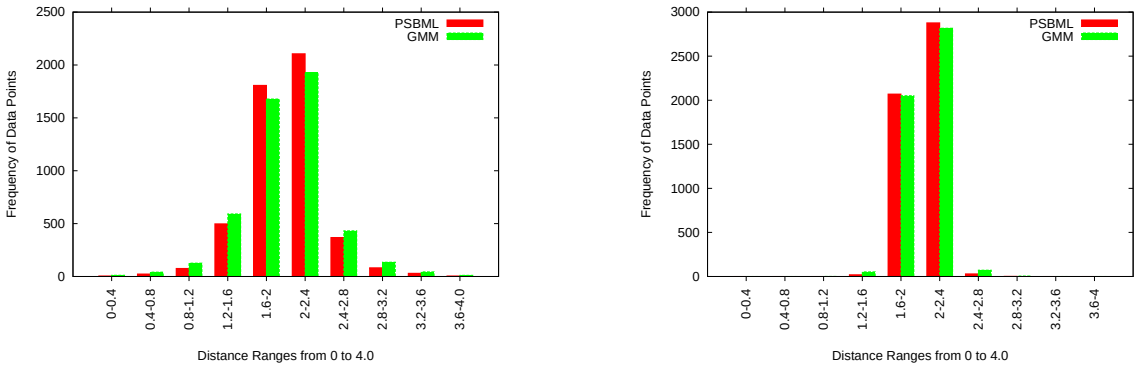

Figure 2: Linearly separable dataset: Data distribution at epochs 25 (Left) and 50 (Right) using PSBML and GMMs.

fit. Fitness proportionate selection, which makes copies of highly fit points, is thus equivalent to performing a weighted sampling of the data. As a result, through the iterations, the peaks of the data density gradually move towards the boundary, in a way simulating successive estimations of the modes via mean-shift, with convergence at the margin.

5. EXPERIMENTS

All the experiments were run on an Intel Core2 Duo machine with 8GB RAM and 3.2GHz CPU. The PSBML algorithm implementation and the comparative classifier runs were all run using Java with max heap size of 4GB and and WEKA ML software [15]. LibSVM and LibLinear were run using native implementation in C++ for performance comparisons [11, 12].

5.1 Empirical Equivalence between GMMs with Mean-shift and PSBML

To empirically verify the above hypothesis and the relationship between PSBML and GMMs, a detailed step by step process was employed. First, synthetic datasets were crafted to have known boundaries/margins on which we ran the experiments. The experiments ranged from running PSBML with a machine learner in a grid, to eventually running GMMs with mean-shift on the whole dataset without any grid or learner. This change was done gradually in three steps, detailed below with justification of their intent. The focus of these experiments was to validate the comparison between GMMs and PSBML in terms of distribution of instances. At this stage classifier accuracy comparisons were not the focus.

1. Run the PSBML algorithm using a 5×5 spatial grid with the C9 neighborhood (see Fig. 1) and a large margin classifier on the synthetic dataset to observe the population distribution change over time.

2. Replace 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 according to its distance from the known boundary (the smaller the distance, the larger the weight). We run GMM with mean-shift on each node and perform sampling iteratively at every training epoch exactly as in PSBML. We observe the population distribution change over time. The assumption of the relationship between PSBML and GMMs with mean-shift is confirmed if the distributions generated in the two cases are similar.

3. Delete the grid structure and run GMMs with mean-shift estimation on the whole dataset, with each instance weighted according to its distance from the known boundary as above. We observe the data distribution and final modes at convergence, and compare them with those obtained in the previous setting.

5.1.1 *Experiment on a Linearly Separable Dataset*

A linearly separable one dimensional dataset of size 5000 is generated by uniformly sampling in the interval $[0, 4]$. The instances between 0 and 2 are labeled as negative, and the instances between 2 and 4 are labeled as positive. We run the three experiments as described in the previous section. For experiment 1, the large margin classifier used at each node fits a line to its training set by placing it at the location corresponding to the average distance between the smallest positive and the largest negative instances. The confidence associated to a point is its distance from the estimated edge.

To compare the data distributions obtained in experiments 1 and 2, we recorded the number of points at various intervals at generations 25 and 50. The resulting histograms are reported in Figure 2. We can clearly observe that the two methodologies, PSBML and GMMs with mean-shift, provide a practically identical distribution at both generations, and they converge to a distribution with modes centered on the points closest to the actual boundary ($x = 2$).

For experiment 3, we run GMMs with mean-shift estimation 30 times on the whole weighted data. The means of the modes at convergence were 1.973 and 2.012, with a very small standard deviation of 0.046. Again, the distribution at convergence was very close to those obtained in experiments 1 and 2. Interestingly, when the weights were removed, and therefore, implicitly the role performed by the classifier was also removed, the means of the modes at convergence moved to 1.03 and 2.98. This result corroborates the fact that the weights credited to points by PSBML proportionally increase their mass, so that the modes of the resulting distributions are centered on those points with the largest weights.

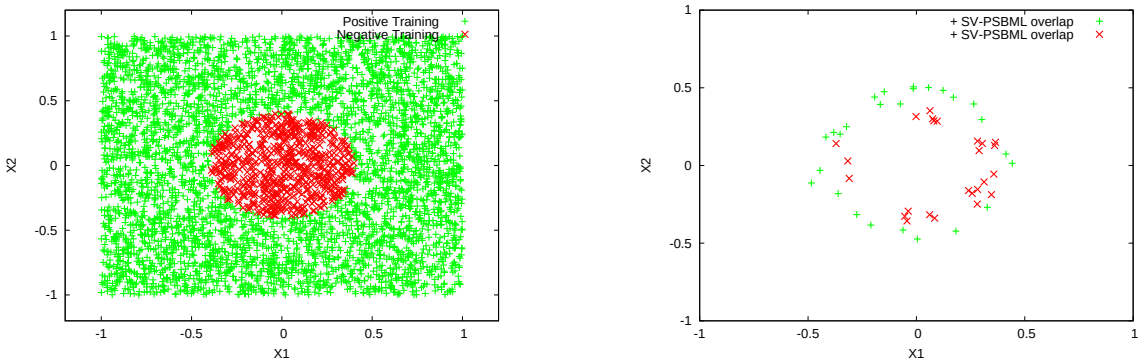

Figure 3: (Left) Circle dataset; (Right) Points that are both support vectors and hard instances selected by PSBML.

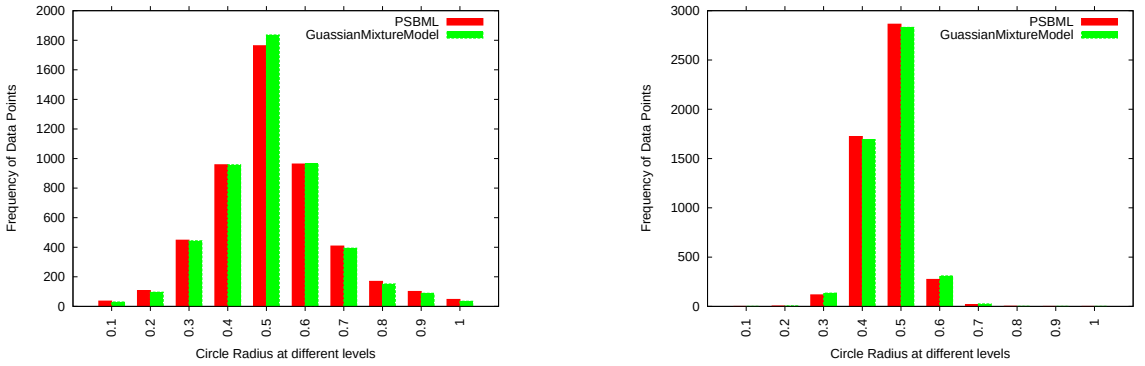

Figure 4: Circle dataset: Data distribution at epochs 25 (Left) and 50 (Right) using PSBML and GMMs.

5.1.2 *Experiment on a Non-linearly Separable 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 3). Again, we run the three experiments as described in Section 5.1. 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 circled 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 generations 25 and 50. The resulting histograms are reported in Figure 4. We can observe again that the two methodologies, PSBML and GMMs with mean-shift, provide a practically 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 run 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. Again, the distribution at convergence was very close to those obtained in experiments 1 and 2. Similarly to the one dimensional dataset, removing the weights from the instances during the runs with GMMs results in a change of means of modes at convergence; they are $(-0.03, 0.51)$ and $(0.03, -0.49)$.

5.1.3 *Support Vectors and Hard Instances in PSBML*

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. The points that are identified both as hard instances and as support vectors for the circle dataset are shown in Figure 3.

Datasets	1D Linear	2D Circle
SV overlap	92%	90%

Table 1: Support vectors and PSBML hard instances overlap.

5.1.4 *Weight Distribution Changes*

One important property of boosting is to scale the weights of data as a function of its distance from the margin. SSEAs have similar behavior where the *takeover* curves exhibit a logistic function with time [1, 14, 25]. To observe the effect

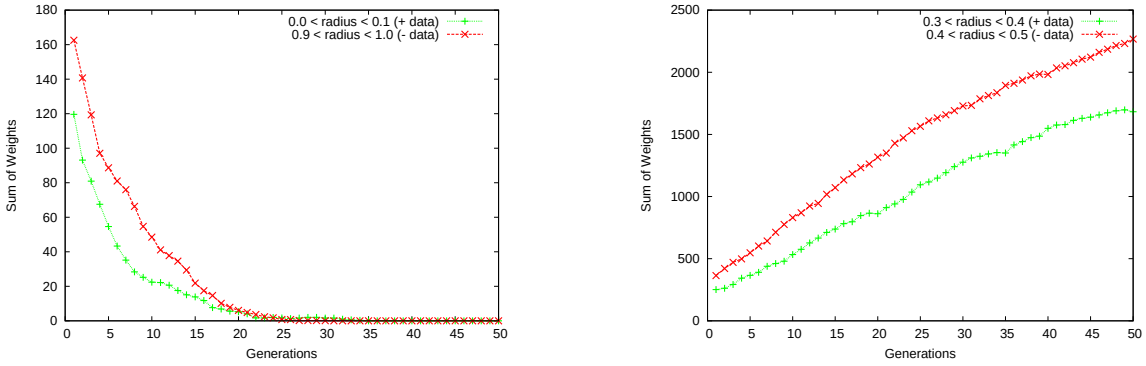

Figure 5: Changes in weight distribution as function of time: (Left) exponential decay; (Right) logistic increase.

of weight changes, in Figure 5 we plotted the weights of all points at different radii for the circle dataset at different generations. We can clearly see the exponential decay and the logistic increase based on the data points closeness to the margin. For positive points, when the radius is between 0.3 and 0.4, and for negative points, when the radius is between 0.4 and 0.5, an increase is seen with time, and for the rest there is exponential decay, confirming the behavior analogous to boosting.

5.2 Scalability and Accuracy Experiments

To evaluate how PSBML would behave in terms of scalability of data, we created three different non-linear and complex decision boundary datasets with functions normally used for optimization experiments. In each case, the points above the surface defined by the function were designated as positive examples, and the points below were negative examples.

The first dataset was a two dimensional decision boundary based on a sine wave generated by the function

$$f(\mathbf{x}) = 2sin(2\pi x_1) \quad (1)$$

The dimension x_1 was sampled from $[0, 6.28]$ and the $y = f(\mathbf{x})$ dimension was randomly sampled from $[0, 2]$.

A three dimensional decision boundary was generated using the Rastrigin function

$$f(\mathbf{x}) = 20 + (x_1^2 - 10cos(2\pi x_1)) + (x_2^2 - 10cos(2\pi x_2)) \quad (2)$$

The two dimensions x_1 and x_2 were sampled between $[-1, 1]$, and the $y = f(\mathbf{x})$ dimension was sampled between $[0, 40]$.

Another three dimensional decision boundary was generated using the Rosenbrock function

$$f(\mathbf{x}) = (1 - x_1^2)^2 + 100(x_2 - x_1^2)^2 \quad (3)$$

The two dimensions x_1 and x_2 were sampled between $[-3, 3]$ and the $y = f(\mathbf{x})$ dimension was sampled between [0,2500].

In each case we generated a training dataset with a million instances. However, this was too large for an SVM (LIBSVM) with a non-linear kernel (RBF kernel). We kept reducing the data and found that at 10% (i.e. around $100K$ sample points) the non-linear SVM could run in reasonable time. We thus performed 25×10 times a bootstrap approach of sampling with replacement from the whole dataset, and computed the best performance from the runs. For comparison, we also ran the liblinear SVM, which has been optimized to scale on large datasets but constrained to linear models only. The trade-off of speed for model simplification results in reduced performance, exaggerated here because of the highly non-linear datasets.

Finally, we ran PSBML with decision trees (J48) [20] as the base classifier with a 3×3 grid for 25 generations to approximate the budget in terms of data sampled. A decision tree classifier was used as it had an intermediate training speed relative to the other classifiers we tested initially (fast LibLinear and the kernel estimated Naive Bayes) [16]. PSBML takes advantage of multi-threading and parallelizing the node learning on a multi-processor machine. The whole setup was run 10 times to get the average auROC of all the runs and paired t-test for significance comparisons. Table 2 summarizes the findings. We also measured the speed and show comparative runs in Table 3. We clearly see that, while maintaining good accuracy, the parallelization using SSEA helped reducing the training time significantly.

To further evaluate the scalability of PSBML on training times with different sizes of the training set, we used the real-world KDDCup-99 intrusion detection dataset with close to 4.9 million training data, 42 features, and 24 classes. The training data was sampled in various sizes from $50K$, $100K$, $500K$ and 1 Million, and 10 runs were performed with standard PSBML with decision trees, a 3×3 grid, and the C9 neighborhood. The training times are shown in Figure 6, which clearly shows almost linear scalability with training data sizes.

Datasets	Sinewave	Rossenbrock	Rastrigin
SVM (Sampled 111k)	70.3	97.2	85.1
LibLinear (1M)	68.8	51.1	69.3
PSBML (1M)	**84.2**	**99.6**	**92.2**

Table 2: Accuracy (auROC) comparisons with SVMs.

Datasets	Sinewave	Rossenbrock	Rastrigin
LibSVM 111K	356.23	243.2	367.8
LibSVM 1M	3611.23	2512.2	2713.4
PSBML-Node 111K	25.8	28.3	22.3
PSBML-Grid 1M	26.2	28.6	22.6

Table 3: Average speed (secs) comparisons with SVMs.

5.3 Accuracy Comparison on a Large Real-world Dataset

The goal of this experiment is to perform a statistical analysis comparing PSBML to known classifiers and meta-classifiers on a sufficiently large real-world dataset with non-overlapping train/test distributions. We used the NSL-KDD dataset with approximately $120K$ training data and $22K$ test data [28]. The NSL-KDD dataset retained all the important statistical characteristics of KDDCup-99, but it's smaller so that performance can be evaluated on a variety of algorithms. We ran the PSBML algorithm with a 3×3 grid, the C9 neighborhood, and Naive Bayes as the base classifier. We obtained similar performance using other classifiers like decision trees, but to reduce experimental computation time we used Naive Bayes. We used the top ranking competitive methods (as shown in [28]), performed 30 runs of 10-fold cross validation, and compared the results using a paired t-test for significance. The results are shown in Table 4. Boosting and PSBML are very close in performance. However, in this case, PSBML is statistically significantly better, illustrating the ability of PSBML to perform well on a large dataset, while maintaining the advantage of parallelization.

Algorithms	Accuracy (auROC)
DecisionTee (J48)	81.02
NaiveBayes	80.67
NB-Tree	82.8
NeuralNetwork	77.42
SVM (RBF)	60.12
RandomForest	80.67
RandomTree	81.5
AdaBoost	90.77
PSBML (NaiveBayes)	**93.5**

Table 4: Performance on NSL-KDD.

5.4 Noise Impact Analysis

Classification problems with medium-to-large datasets from the UCI ML repository [3] were chosen to perform an additional empirical analysis of PSBML in presence of noise. The purpose of this experiment was to compare the impact of noise on the behavior of boosting on a whole dataset and PSBML in parallel mode. The datasets are described in Table 5 in terms of numbers of training and testing instances, number of features, and number of classes.

We first performed a comparison between the PSBML algorithm and standard AdaBoost to show their accuracy measured as area under ROC which is a reliable measure when data is imbalanced.

Adding noise to the class values in the training set and validating it on the hold-out test set gives a good empirical robustness measure of an algorithm to noise. We performed an analysis with noise level at 10% and ran the train/test 30 times, using 90-10 split for train-test in WEKA-experimenter. We ran AdaBoost with the same underlying classifier (Naive Bayes) and with the same settings so that we can draw conclusive results in the comparisons. We used a paired t-test for significance comparison. The results are shown in Table 6, where statistically significant accuracy values are shown in boldface. PSBML achieved statistically significant

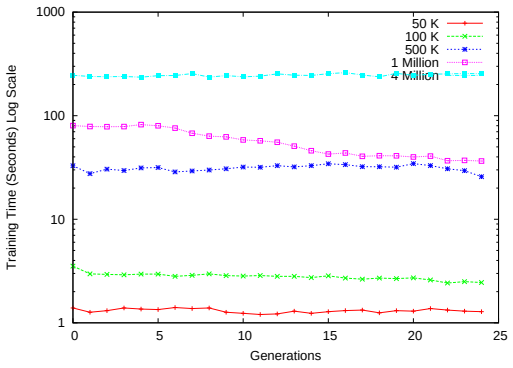

Figure 6: Mean training times with varying dataset sizes.

improvements compared to AdaBoost in most datasets. Similar results were obtained for 10% level of noise, showing advantages of PSBML.

6. CONCLUSION AND FUTURE WORK

In this paper we take an important first step in analyzing PSBML, an SSEA-based parallel boosting framework for improving the scalability of ML techniques. By establishing an equivalence between PSBML and GMMs using mean-shift analysis, important properties regarding convergence to large margins are obtained and verified empirically. In addition, it was also shown that the scalability achieved by PSBML is obtained without compromising on accuracy or performance, an important feature from a practitioner's perspective. Finally, the spatial topology of PSBML was shown to provide a level of robustness to noise that is lacking in standard ML techniques.

The experimental studies in this paper were done on a single multi-threaded machine. An interesting extension of this work would be to analyze it for parallel and distributed computational architectures like the Beowulf-style clusters. Also, the focus of this paper was on analyzing ML properties. However, the GMM approach is more general than that and could be a useful analysis tool for other more optimization-oriented EC areas such as SSEAs and EDAs.

7. REFERENCES

[1] E. Alba and G. Luque. Growth curves and takeover time in distributed evolutionary algorithms. In *Proc. of GECCO-2004*, pages 864–876, 2004.

[2] S. J. Arnold. Multivariate inheritance and evolution: a review of concepts. *Quantitative Genetic Studies of Behavioral Evolution*, pages 17–48, 1994.

[3] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[4] J. Bacardit and X. Llorà. Large scale data mining using genetics-based machine learning. In *Proc. of GECCO-2012*, pages 1171–1196, 2012.

[5] K. P. Bennett, A. Demiriz, and R. Maclin. Exploiting unlabeled data in ensemble methods. In *In Proc. ACM Int. Conf. Knowledge Discovery and Data Mining*, pages 289–296. ACM Press, 2002.

[6] A. Bordes, L. Bottou, and P. Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *J. of Machine Learning Research*, 10:1737–1754, 2009.

Dataset	Chess	Spam	Digit	Magick	Adult	W8A	Cod	Cover	KDD99
#Train	3196	4600	10992	19020	32560	49749	271617	581012	4898431
#Test	319	460	1099	1902	16279	14951	59535	58102	311027
#Feat	36	57	256	10	14	300	8	54	42
#Class	2	2	10	2	2	2	2	7	24

Table 5: UCI datasets used in the experiments.

No Noise	Chess	Spam	Digit	Magick	Adult	W8A	Cod	Cover	KDD99
AdaBoost	92.83	93.89	86.9	83.22	85.13	97.45	92.43	78.72	99.14
PSBML	**93**	**94**	**90**	83.1	**89.01**	**98.1**	90.01	**85.1**	**99.65**
10%Noise									
AdaBoost	81.80	76.6	81.9	80.22	82.13	96.45	**92.13**	76.72	98.14
PSBML	**84.38**	**81.8**	79.21	**81.9**	**83.7**	96.9	90.20	**78.1**	99.18

Table 6: Comparison with AdaBoost with and without Noise (10%).

[7] M. Á. Carreira-Perpiñán. Gaussian mean shift is an em algorithm. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29:2007, 2005.

[8] M. Á. Carreira-Perpiñán and C. K. I. Williams. On the number of modes of a gaussian mixture. In *Proc. of Scale-Space 2003*, pages 625–640, 2003.

[9] P. Domingos, G. Hulten, P. C. W. Edu, and C. H. G. W. Edu. A general method for scaling up machine learning algorithms and its application to clustering. In *In Proceedings of the Eighteenth Int'l Conference on Machine Learning*, pages 106–113, 2001.

[10] N. U. Edakunni, G. Brown, and T. Kovacs. Boosting as a product of experts. In *Proc. of UAI-2011*, pages 187–194, 2011.

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

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

[13] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. of ICML-1996*, pages 148–156, 1996.

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

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[16] U. Kamath, J. Kaers, A. Shehu, and K. A. De Jong. A spatial EA framework for parallelizing machine learning methods. In *Parallel Problem Solving from Nature — PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 206–215. Springer, 2012.

[17] H. Mühlenbein, J. Bendisch, and H.-M. Voigt. From recombination of genes to the estimation of distributions: II. continuous parameters. In *Parallel Problem Solving from Nature — PPSN IV*, pages 188–197, Berlin, 1996. Springer.

[18] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions: I. Binary parameters. In *Parallel Problem Solving from Nature — PPSN IV*, pages 178–187, Berlin, 1996. Springer.

[19] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[20] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[21] C. E. Rasmussen. The infinite gaussian mixture model. In *In Advances in Neural Information Processing Systems 12*, pages 554–560. MIT Press, 2000.

[22] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn. Speaker verification using adapted gaussian mixture models. *Digital Signal Processing*, 10:19–41, 2000.

[23] S. H. Rice. *Evolutionary Theory: Mathematical and Conceptual Foundations*. Sinauer Associates, Inc., 2004.

[24] J. Sarma and K. De Jong. An analysis of the effects of neighborhood size and shape on local selection algorithms. In *Parallel Problem Solving from Nature — PPSN IV*, pages 236–244. Springer-Verlag, 1996.

[25] J. Sarma and K. De Jong. An analysis of local selection algorithms in a spatially structured evolutionary algorithm. In *Proc. ICGA-1997*, pages 181–187, 1997.

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

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

[28] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, pages 53–58, Piscataway, NJ, USA, 2009. IEEE Press.

[29] M. Tomassini. *Spatially structured evolutionary algorithms: artificial evolution in space and time*. Natural Computing series. Springer, 2005.

[30] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.