A Spatial EA Framework for Parallelizing Machine Learning Methods

Uday Kamath[1], Johan Kaers[2], Amarda Shehu[1], and Kenneth A. De Jong[1]

[1]George Mason University, Fairfax VA 22003, USA,
{ukamath,ashehu,kdejong@gmu.edu}
[2]Shaman Research, Heverlee 3001, Belgium,
{johankaers@telenet.be}

Abstract. The scalability of machine learning (ML) algorithms has become increasingly important due to the ever increasing size of datasets and increasing complexity of the models induced. Standard approaches for dealing with this issue generally involve developing parallel and distributed versions of the ML algorithms and/or reducing the dataset sizes via sampling techniques. In this paper we describe an alternative approach that combines the features of spatially-structured evolutionary algorithms (SSEAs) with the well-known machine learning techniques of ensemble learning and boosting. The result is a powerful and robust framework for parallelizing ML methods in a way that does not require changes to the methods. We first describe the framework and illustrate its behavior on a simple synthetic problem, and then evaluate its scalability and robustness using several different ML methods on a set of benchmark problems from the UC Irvine ML database.

Keywords: spatially-structured evolutionary algorithms, machine learning, ensemble learning, boosting.

1 Introduction

The most common applications of machine learning involve supervised learning in which a training set of labeled examples (or instances) is used to learn a model that can be subsequently used to make predictions about previously unseen examples. The scalability of such ML algorithms has become increasingly important as datasets become larger and the complexity of the induced models increases. Many current ML techniques scale poorly either because they require the entire training set to be in memory simultaneously, or because the running time of the model induction code grows non-linearly, or both [1]. Basic solutions like reducing the size of the training datasets via sampling can be used but can introduce sampling errors. ML boosting techniques are designed to deal with hard-to-classify examples, but do so by making multiple passes over the training data [2]. More complex approaches involve changing the basic structure of ML methods into parallel and distributed versions.

In this paper we describe an alternative approach that combines the features of spatially-structured evolutionary algorithms (SSEAs) with the well-known

machine learning techniques of ensemble learning and boosting. The result is a powerful and robust framework for parallelizing ML methods in a way that does not require changes to the underlying ML methods. We first describe our framework and illustrate its behavior on a simple synthetic problem. We then evaluate its scalability and robustness using several different standard ML methods on a selected subset of the benchmark problems in the UC Irvine ML database.

2 Related Work

As noted above, our framework combines features from both the evolutionary computing (EC) and ML communities. In this section we briefly summarize them.

Spatially-structured evolutionary algorithms (SSEAs) which use topologically distributed populations and local neighborhood selection have been well analyzed in the EC literature [3]. SSEAs have been shown to maintain a diverse set of better individuals longer, resulting in improved performance in many applications [4]. However, the key feature that we want to take advantage of is its "embarrassingly parallel" architecture in that at each topological grid point a micro-EA is running with only local interactions with its immediate neighbors.

One of the interesting ML developments is that a collection (ensemble) of simpler classifiers can often be more accurate than a single more complex classifier, and of course much easier to parallelize [5]. This maps nicely onto SSEAs in the sense that an ensemble of classifiers can be distributed across the topological grid points in an SSEA. However, standard ensemble techniques require each classifier to look at the entire (possibly sampled) set of training data. Imagine instead that the training data is also distributed across an SSEA's grid points, and each local ML technique only works with the training data in its immediate neighborhood. Such a framework has the potential for a method-independent approach to parallelizing ML techniques.

A second interesting ML development is the awareness of the important distinction between easy and hard training examples. Intuitively, the hard examples are those close to classification decision boundaries. A classic example of these are the "support vectors" on which support vector ML techniques are based. Since the decision boundaries are not known *a priori*, in general, multiple passes over the training data are required to identify these examples often via "boosting" techniques that increase the frequency and/or weights of such training instances [2]. Beyond just improving classification accuracy, the identification of these difficult training examples on the boundaries often leads to additional problem insights used for finding interesting features for classification [6].

Our SSEA framework incorporates this idea as well by introducing a local neighborhood notion of hardness, using it as a measure of fitness, and boosting the harder instances via fitness-proportional selection. The result is a parallel ML technique in which both classification accuracy and the identification of hard instances improves as the SSEA evolves.

3 The PSBML Framework

Our framework for parallelizing machine learning methods (Parallel Spatial Boosting Machine Learning) has at its core a standard SSEA [3] in which individuals and algorithms are distributed over a two-dimensional torroidal grid with a common algorithm running locally on each node in the grid and only local interactions with nearby grid points. Selection pressure in an SSEA is determined by two design choices: the selection method used by the local EAs running on each grid point, and the size and shape of the neighborhood structure. In the experiments described in this paper, the local EA selection method used was fitness-proportional selection (our boosting technique). We did experiment with different standard neighborhood structures (Fig. 1) in order to study the effects of varying the overall selection pressure.

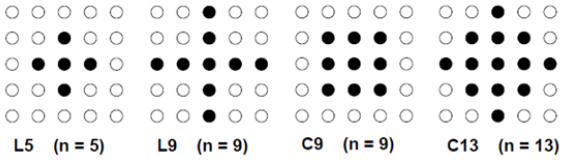


Fig. 1. 2D-grid with various neighborhood structures (Source: LNCS 1141, p 237)

Each node in the grid has a local EA running maintaining a population of ML training examples that gets updated each generational cycle. Also running on each node in the grid is a local ML technique (e.g., a naive Bayesian classifier, a decision tree learner, etc.). On each generational cycle, a node performs a standard ML train-test procedure using the training examples on its node for training, and using the training examples on neighboring nodes for testing. As is the case with standard ML boosting methods, in addition to classifying the test examples, the learners output a confidence value for each decision. The confidence values are used to assign a fitness to each instance, allowing the local EA to subsequently select (boost) the more difficult examples. Since each example is a member of the neighborhood of multiple local ML methods, we have an ensemble assessment of difficulty similar to the classification margin concept used in boosting [2]; namely, the smallest confidence from any node, for any class is taken as the fitness w of the instance.

$$w = \min_{i \in class} \left(\min_{n \in neighbor} c_{ni} \right)$$

Experimentally, we determined that a non-overlapping-generation model for the local EAs was much less effective than an overlapping one, in which only a fraction of the local population was replaced each generational cycle. We implemented this feature through a replacement probability parameter p_r, and found

that values around 0.20 were most effective (i.e., replacing about 20% each generation).

The overall pseudo-code of PSBML is as follows:

- Initialization: distribute example dataset over all the nodes in the grid.
- For every EA generation
  - Train all nodes.
  - Test all instances on their neighbor nodes, assigning confidence values.
  - For all nodes
    - Create a selection pool consisting of the instances on the node and its neighbor nodes.
    - For each instance in its current population, replace it with probability p_r with an instance from the selection pool using fitness-proportional selection.

4 Analysis of the PSBML Framework

In growth and diffusion studies and its application in spatial evolutionary learning, the migration of individuals through the nodes has been studied under various selection methods [3, 7]. For spatially-structured populations using fitness-proportional selection, the growth rate of the fittest individual in the population is described by a logistic curve:

$$p_{b,t} = \frac{1}{1 + (\frac{1}{p_{b,0}} - 1)e^{-at}}$$

where $p_{b,t}$ is the proportion of the fittest individual at time t, $p_{b,0}$ is the proportion of the fittest individual at time 0, and a is the growth coefficient which depends on the shape and size of the neighborhood relative to the entire grid. As described in [3], the neighborhood size and shape influences the growth coefficient a, with the larger, more dense neighborhoods producing the stronger selection pressures. Figure 2 illustrates how these growth curves differ, i.e., how the selection pressure changes with different neighborhood sizes and shapes.

In PSBML, fitness is a measure of classification difficulty. So, the underlying dynamic is one of propagating difficult instances throughout the grid, gradually replacing the easy ones in more and more node populations. The total set of unique instances taken over all nodes will therefore concentrate on the fewer but more relevant ones for the learning problem, i.e., evolutionary boosting.

4.1 Experiment 1: A Simple Circle Classification Problem

As a first step in analyzing the behavior of PSBML, we designed a simple synthetic ML problem to illustrate its behavior. The underlying binary classification problem was a 2-dimensional Euclidean space in which points inside a circle centered at the origin were designated as negative examples and the rest as positive

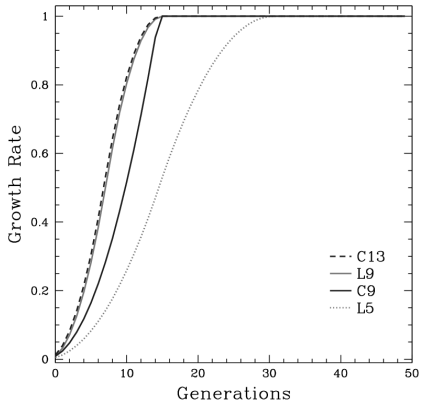


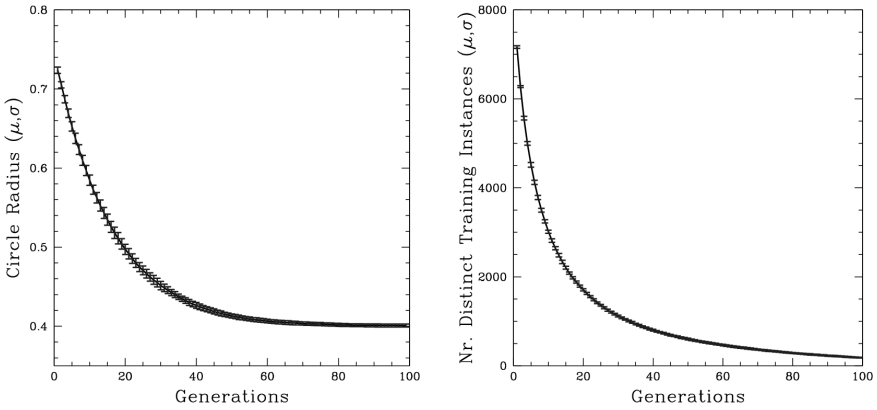
Fig. 2. Growth curves for C13, L9, C9 and L5 neighborhoods.

examples. In this case, a simple ML learner is trying to infer the radius of the circle from the training examples it is given by choosing a radius equal to the average distance from the origin of the largest negative example and the smallest positive example. Classification confidence is then based on the distance of an instance from the edge of the circle with the hypothesized radius.

Figure 3 illustrates the results involving a 5x5 torroidal grid in which 10,000 sample points are equally distributed over the 25 grid nodes. In these experiments, the C9 neighborhood structure was used. We ran the PSBML framework on this setup for 100 generations and collected two pieces of behavioral data: the average distance of the instances from the origin, and the number of distinct instances over all nodes. As expected, the global behavior of PSBML was to steadily reduce the number of distinct training instances to a subset that was "on the margin", i.e. close to the decision boundary of 0.4.

4.2 Experiment 2: Neighborhood Effects

The next step was to study the effects that SSEA neighborhood structure has on the performance of PSBML. We chose the UCI Chess (King-Rook vs. King-Pawn) dataset for these experiments. It has 3196 instances, 36 attributes and 2 classes. We ran PSBML on this problem using various neighborhood structures as shown in Fig. 4(a). We used a 5X5 grid with a naive Bayesian classifier as the ML method with discretization for numeric features. Although the average reduction in the training data was quite similar for all the neighborhoods, their classic "over fitting curves" were different. The stronger selection pressures of L9 and C13 produced the more rapid initial decrease in 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 C9.



(a) (b)

Fig. 3. (a) Mean (μ) and standard deviation (σ, error bars) are shown for the circle radius. (b) The number of distinct training distances decreases with the generations.

4.3 Experiment 3: Impact of p_r

In this set of experiments, we ran PSBML on the UCI Chess dataset with different p_r values to observe the effect that different rates of replacement have on the performance of PSBML. Figure 4(b)-(c) illustrates that increasing p_r results in faster convergence but a less accurate learner, with the best results obtained when p_r is about 0.2.

4.4 Experimental Analyses on Benchmark Datasets

Finally, we selected nine ML problems with medium-to-large datasets from the UCI ML repository [8] to test the robustness of PSBML. The datasets are shown in Table 1 in terms of the number of training instances, number of testing instances, number of features, and number of classes.

Table 1. UCI Benchmark datasets

Dataset	Chess	Spam	Digit	Magick	Adult	W8A	Cod	Cover	KDD99
#Train	3196	4600	10992	19020	32560	49749	271617	581012	4000000
#Test	319	460	1099	1902	16279	14951	59535	58102	311000
#Feat	36	57	256	10	14	300	8	54	42
#Class	2	2	10	2	2	2	2	7	24

We employed a 5x5 grid with C9 neighborhood configuration and p_r of 0.2. To evaluate the robustness and meta learning capability, we tested PSBML with 3 standard ML methods: a naive Bayesian (NB) classifier, a decision tree (DT) learner and a support vector machine (SVM), each employed with the standard

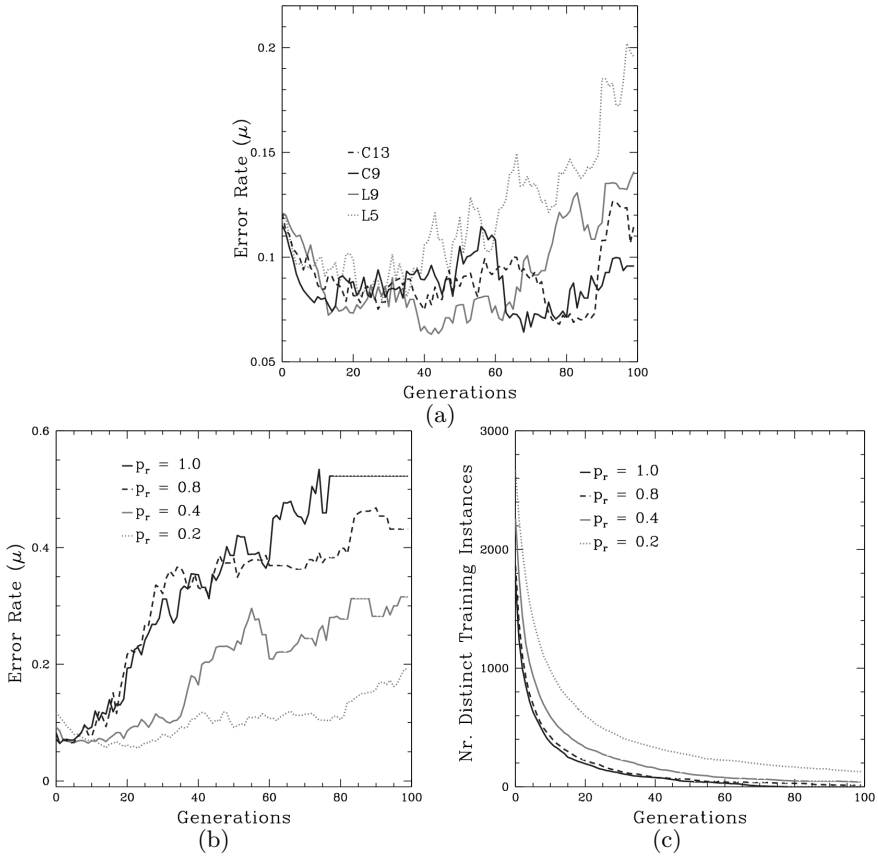


Fig. 4. Results are shown on the UCI chess dataset. The error rate is shown as a function of the neighborhood structure in (a) and probability p_r in (b). The number of distinct training instances is shown as a function of p_r in (c).

implementations available in Weka [9]. The classification accuracy obtained by each ML method is compared to the classification accuracy obtained when embedding that method within PSBML. Results are shown in Table 2. Rows labeled "#Hard" show the reduced size of the data set resulting from the evolutionary boosting in PSBML. All reported results are averages over 30 runs (ceiling values reported for "#Hard"). The standard deviation for most runs was below 0.1 and so is not shown. Fields labeled NA correspond to experiments that could not be performed due to algorithmic constraints or very long training times required by the base classifier. Comparisons between methods are done by performing 30 runs and using t-tests for statistical significance with 95% confidence intervals. Runs that show statistically significantly improvements are highlighted in bold.

Comparison of PSBML with NB: Rows 2-4 in Table 2 summarize the results of running PSBML on the nine ML datasets using a naive Bayesian (NB) classifier.

Table 2. Comparison of PSBML with NB, DT, and SVM on UCI Benchmark datasets

Dataset	Chess	Spam	Digit	Magick	Adult	W8A	Cod	Cover	KDD99
NB	88.32	79.52	84.41	78.21	83.19	96.7	78.11	79.15	98.89
PNB	**93**	**94**	**90**	**83.1**	**89.01**	**98.1**	**90.01**	**85.1**	**99.65**
#Hard	191	752	484	4544	5625	7234	9157	47234	45034
DT	99.65	97.17	79.51	85.49	85.83	NA	95.12	NA	NA
PDT	99.64	96.1	**80.12**	**86.1**	85.61	NA	**96.34**	NA	NA
#Hard	2678	2667	302	7699	9163	NA	45001	NA	NA
SVM	96.24	90.76	87.97	79.33	85.26	NA	93.9	NA	NA
PSVM	**97.1**	78	**88.45**	**80.12**	**86.1**	NA	84.1	NA	NA
#Hard	2001	3078	1297	3715	23409	NA	47234	NA	NA

Row 2 labeled NB shows the classification accuracies obtained by NB, and row 3 labeled PNB shows the the classification accuracies obtained when embedding NB in PSBML framework. For all nine of the benchmarks, PSBML with NB produces statistically significant improvements in classification accuracy.

Comparison of PSBML with DT: Rows 5-6 in Table 2 summarize the results of running PSBML on the nine ML datasets using a decision tree (DT) classifier. Row 5 labeled DT shows the classification accuracies obtained by DT, and row 6 labeled PDT shows the the classification accuracies obtained when embedding DT in PSBML. In this case, we observed statistically significant improvements on only a subset of the benchmarks.

Comparison of PSBML with SVM: Rows 8-9 in Table 2 summarize the results of running PSBML on the nine ML datasets using an SVM classifier. Row 8 labeled "SVM" shows the classification accuracies obtained by SVM, and row 9 labeled "PSVM" shows the the classification accuracies obtained when embedding SVM in PSBML. In this case, we observed statistically significant improvements on only a subset of the benchmarks.

Recent research has shown that parallelizing boosting algorithms results in efficient learning [10, 11]. So we also compared the performance of PSBML to the ensemble-based meta-learners AdaBoost and ParallelBoost. Table 3 summarizes the result using NB as the base classifier.

The column labeled PNB shows the classification accuracies obtained by PSBML running an NB classifier, the column labeled AB-NB shows the classification accuracies obtained when employing AdaBoost with NB, and the column labeled PB-NB shows the classification accuracies obtained when employing ParallelBoost with NB. Table 3 shows that the spatial boosting in PSBML outperforms the normal or parallelized boosting on most datasets and overcomes the problem of large datasets required for boosting in memory.

Table 3. Comparison of PSBML with AdaBoost and ParallelBoost

Dataset	Chess	Spam	Digit	Magick	Adult	W8A	Cod	Cover	KDD99
AB-NB	92.83	93.89	86.9	83.22	85.13	97.45	92.43	78.72	99.14
PB-NB	92.9	93.8	77.21	83.9	85.7	97.9	93.21	82.1	99.18
PNB	**93**	**94**	**90**	83.1	**89.01**	**98.1**	90.01	**85.1**	**99.65**

4.5 Scalability Experiments

The experiments above on the nine chosen datasets illustrate that PSBML achieves similar or better accuracies in most cases in comparison with other single classifiers. These experiments were run on single machines using single computation threads. An important motivation for PSBML was to take advantage of the embarrassingly parallel nature of SSEAs. Although SSEAs can map onto loosely-coupled Beowulf-style clusters, a better fit is a multi-threaded shared memory architecture with each local EA running on a separate thread.

Although a principled port of PSBML to our GPU server environment is in progress, to date our multi-threading experiments consist of measuring PSBML speedup on a single machine with multiple cores and multi-threading support. As an example, Fig. 5 shows the running time in milliseconds of PSBML using a 4x4 grid as a function of the number of threads employed. These particular results were obtained running under Linux OS on a 2GHz 2x4 core Intel machine.

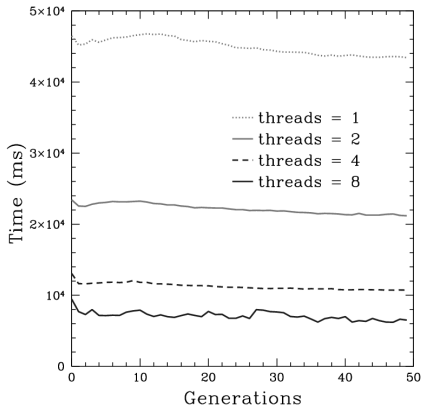


Fig. 5. Training time when using 1, 2, 4, and 8 threads.

5 Conclusion and Future Work

We have described a novel approach for parallelizing machine learning methods that combines the features of spatially-structured evolutionary algorithms with the well-known machine learning techniques of ensemble learning and boosting.

It does so in a way that does not require changes to the underlying machine learning methods, maintains or improves classification accuracy, and can achieve significant speedup in running times via a straightforward mapping to multi-threaded shared-memory architectures.

Although our experiments to date have been on machines that have significantly fewer parallel threads than the number of grid points of the underlying SSEA, we plan to continue our evaluation of PSBML in the context of a GPU server environment in which this is not the case.

We are also exploring the use of PSBML as the first stage of multi-stage experiments in which subsequent stages take advantage of the reduction of the dataset to both a more manageable size and containing the most critical exemplars.

6 Implementation and Code Availability

Implementation of this framework with source code for the meta classifier integrated with weka, synthetic circle classifier, experiments performed, datasets used are available for academic use on request.

References

1. Bordes, A., Bottou, L., Gallinari, P.: Sgd-qn: Careful quasi-newton stochastic gradient descent. Journal of Machine Learning Research **10** (2009) 1737–1754
2. Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S.: Boosting the margin: A new explanation for the effectiveness of voting methods (1997)
3. Sarma, J., De Jong, K.: An analysis of the effects of neighborhood size and shape on local selection algorithms. In: LNCS 1141: PPSN IV, Springer-Verlag (1996) 236–244
4. Tomassini, M.: Spatially structured evolutionary algorithms: artificial evolution in space and time. Natural computing series. Springer (2005)
5. Opitz, D., Maclin, R.: Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research **11** (1999) 169–198
6. Pamuk, B., Can, T.: Coevolution based prediction of protein-protein interactions with reduced training data. In: Health Informatics and Bioinformatics (HIBIT), 2010 5th International Symposium on. (april 2010) 187 –193
7. B.Banks, R.: Growth and Diffusion Phenomena: Mathematical Frameworks and Applications. Springer (1993)
8. A. Asuncion, D.N.: UCI machine learning repository (2007)
9. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. **11**(1) (2009) 10–18
10. Yu, C., Yu, C., Skillicorn, D.B., Skillicorn, D.B.: Parallelizing boosting and bagging (2001)
11. Favre, B., Hakkani-Tür, D., Cuendet, S.: Icsiboost. http://code.google.come/p/icsiboost (2007)