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 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 ML 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 with the size of the training data, 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.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 206–215, 2012.

In this paper we describe an alternative approach that combines 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 refer to this as our PSBML framework, shorthand for Parallel Spatial Boosting Machine Learning. 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 local algorithm is running that has only local interactions with its immediate neighbors. In that sense our approach has much in common with cellular EAs and cellular automata.

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. The hypothesis we are exploring is that, by distributing the training data across the grid points as well, the emergent capability of an ensemble of ML classifiers that only have access to local subsets of training data will be comparable in classification performance to that of standard ML techniques and significantly more scalable via parallelization.

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 PSBML 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 system evolves.

3 The PSBML Framework

Our PSBML framework for parallelizing machine learning methods has at its core an 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. In our case the common algorithm is a replicator EA (i.e., no reproductive variation) that is manipulating a population of training examples. 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, however, experiment with different standard cellular neighborhood structures (Fig. 1) in order to study the effects of varying the overall selection pressure.

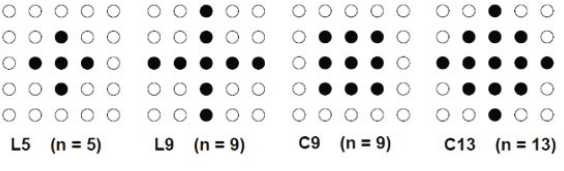

Fig. 1. 2D-grid with various neighborhood structures (Source: LNCS 1141, p 237) [3]

Each node in the grid has a local EA running maintaining a population of ML training examples that gets updated each generational cycle. The fitness of each training example in the population is assessed via is a local ML technique (e.g., a naive Bayesian classifier, a decision tree learner, etc.) in the following manner. 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 of the neighborhood test examples, allowing the local replicator EA to subsequently select (boost) the more difficult examples. Since each member of the overall set of training data is a member of the neighborhood of multiple local ML methods, the result is 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.

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

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). See section 4.3 for more details.

The overall pseudo-code of PSBML is as follows:

- Initialization: distribute the training dataset uniformly over all the nodes in the grid.
- For every EA generation:
  - On each node:
    - Use the local ML technique and the current local population of training examples to produce a candidate classifier.
    - Test this classifier on all the population members in the neighboring nodes, assigning confidence values to each population member.
    - Create a selection pool consisting of all population members of the node and its neighbor nodes.
    - For each member in the current node population, replace it with probability p_r with an individual from the selection pool using fitness-proportional selection.

4 Analysis of the PSBML Framework

As stated in the introduction, the hypothesis for this research is that the emergence behavior of ML techniques embedded locally in this spatially distributed framework will be comparable in classification performance to the corresponding monolithic ML versions with the significant additional benefit of significant improvements in scalability via parallelization. The two important emergent properties are the effects of local boosting and local classifier training and testing. We analyze both effects in this section.

Since local boosting is done by a replicator EA using fitness-proportional selection from a pool that includes neighboring populations, the formal analysis is identical to that of how the emergent selection pressure in SSEAs changes as a function of the neighborhood topology [3,7] as illustrated in Figure 2. In this case, increased selection pressure corresponds to increased boosting rates. As with standard boosting techniques, one must find a growth rate in PSBML that facilities the learning process by gradually propagating the more difficult training examples throughout the grid via evolutionary boosting.

The second emergent property is the overall classification accuracy of PSBML. Unless its local boosting and training elements result in an ensemble performance comparable to monolithic ML techniques, there is no virtue in PSBML's scalability.

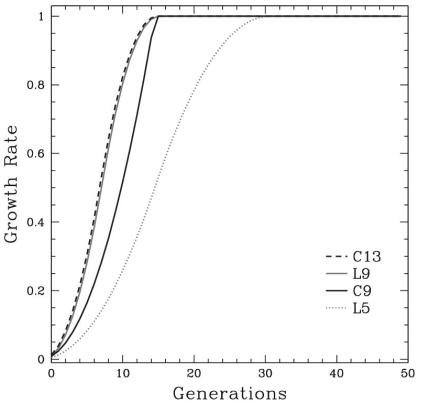

Fig. 2. Growth curves for C13, L9, C9 and L5 neighborhoods

In the following sections we describe an initial set of experiments to assess both of these emergent properties. We start with a series of tuning experiments to obtain a rough estimate of the more important PSBML design parameters that affect these emergent properties. Then, using this as the default PSBML configuration, we evaluate its performance on a set of standard ML benchmark problems and assess the robustness of the approach using a variety of standard ML methods.

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 space in which points inside a circle centered at the origin were designated as negative examples and the rest as positive 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 circle of radius 0.4. The underlying spatial topology was 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 hypothesized, the emergent 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 and comparable to single margin-based classifiers like SVMs.

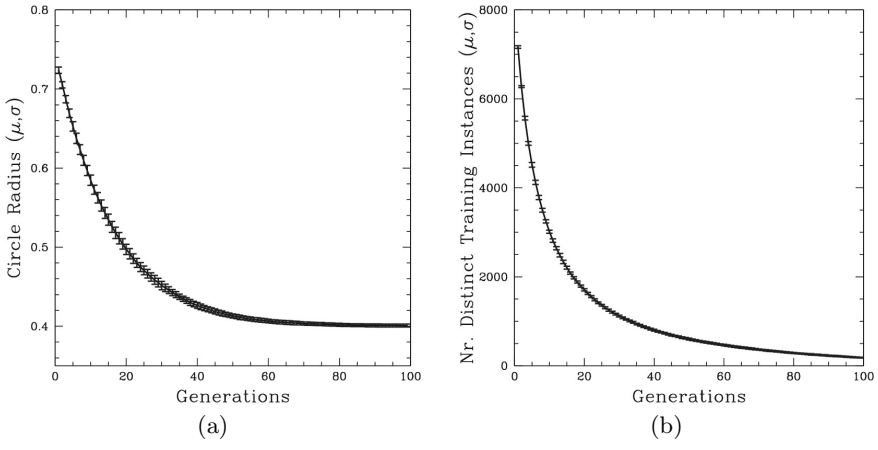

Fig. 3. (a) Mean values with 95% confidence intervals from 30 independent PSBML runs. (b) The number of distinct training distances decreases with the generations.

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. The ensemble classifier is evaluated by combining the reduced datasets from all the nodes, training a single classifier with these and comparing the test set predictions for classification accuracy or the error-rate. 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 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 C9.

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.

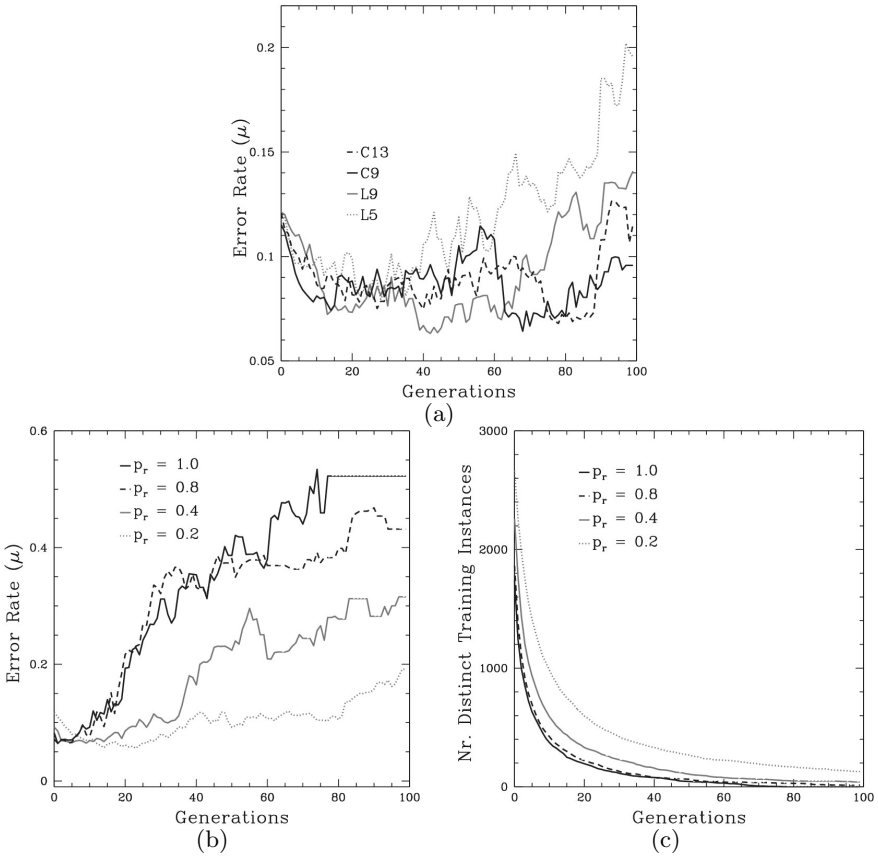

Fig. 4. Results are shown on the UCI chess dataset. The error rate is shown as a function of the neighborhood structure in (a) and probability p_r in (b). The number of distinct training instances is shown as a function of p_r in (c).

4.4 Experimental Analyses on Benchmark Datasets

Using the rough tuning parameters of the previous sections, we evaluated the performance of PSBML on nine classification problems with medium-to-large datasets from the UCI ML repository [8]. 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.

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 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

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

"#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 improvements are highlighted in bold.

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

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. Again we see that PSBML produces comparable or better results.

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 of the previous section support the hypothesis that PSBML achieves comparable classification in comparison with other single classifiers, while providing a significant scalability potential via parallel local learning on local subsets of training data. These experiments were run on single machines using single computation threads. The next obvious step is a systematic study of the scalability of PSBML on a variety of parallel and distributed computational architectures. In general, although cellular models 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.

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 as a function of the number of threads employed, suggesting there is indeed significant potential for speedup and hence scalability. These particular results were obtained running under Linux OS on a 2GHz 2x4 core Intel machine.

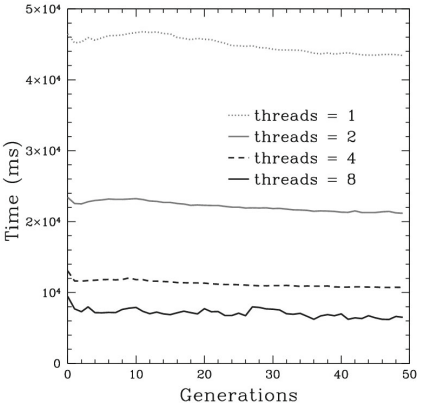

Fig. 5. Training time when using 1, 2, 4, and 8 threads

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.

References

1. Bordes, A., Bottou, L., Gallinari, P.: Sgd-qn: Careful quasi-newton stochastic gradient descent. Journal of Machine Learning Research 10, 1737–1754 (2009)
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: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN IV. LNCS, vol. 1141, pp. 236–244. Springer, Heidelberg (1996)
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, 169–198 (1999)
6. Pamuk, B., Can, T.: Coevolution based prediction of protein-protein interactions with reduced training data. In: 2010 5th International Symposium on Health Informatics and Bioinformatics (HIBIT), pp. 187–193 (April 2010)
7. Banks, R.B.: Growth and Diffusion Phenomena: Mathematical Frameworks and Applications. Springer (1993)
8. Asuncion, A., Newman, D.J.: 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), 10–18 (2009)
10. Yu, C., Skillicorn, D.B.: Parallelizing boosting and bagging (2001)
11. Favre, B., Hakkani-Tür, D., Cuendet, S.: Icsiboost (2007), `http://code.google.come/p/icsiboost`