Ensemble Learning for Free with Evolutionary Algorithms ?

Christian Gagné*
Informatique WGZ Inc.,
819 avenue Monk,
Québec (QC), G1S 3M9, Canada.
christian.gagne@wgz.ca

Marc Schoenauer
Équipe TAO – INRIA Futurs / CNRS UMR 8623,
LRI, Bat. 490, Université Paris Sud,
F-91405 Orsay Cedex, France.
marc.schoenauer@lri.fr

Michèle Sebag
Équipe TAO – CNRS UMR 8623 / INRIA Futurs,
LRI, Bat. 490, Université Paris Sud,
F-91405 Orsay Cedex, France.
michele.sebag@lri.fr

Marco Tomassini
Information Systems Institute,
Université de Lausanne,
CH-1015 Dorigny, Switzerland.
marco.tomassini@unil.ch

ABSTRACT

Evolutionary Learning proceeds by evolving a population of classifiers, from which it generally returns (with some notable exceptions) the single best-of-run classifier as final result. In the meanwhile, Ensemble Learning, one of the most efficient approaches in supervised Machine Learning for the last decade, proceeds by building a population of diverse classifiers. Ensemble Learning with Evolutionary Computation thus receives increasing attention. The Evolutionary Ensemble Learning (EEL) approach presented in this paper features two contributions. First, a new fitness function, inspired by co-evolution and enforcing the classifier diversity, is presented. Further, a new selection criterion based on the classification margin is proposed. This criterion is used to extract the classifier ensemble from the final population only (*Off*-EEL) or incrementally along evolution (*On*-EEL). Experiments on a set of benchmark problems show that *Off*-EEL outperforms single-hypothesis evolutionary learning and state-of-art Boosting and generates smaller classifier ensembles.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

*This work has been mainly realized during a postdoctoral fellowship of Christian Gagné at the University of Lausanne.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

Keywords

Ensemble Learning, Evolutionary Computation

1. INTRODUCTION

Ensemble Learning, one of the main advances in Supervised Machine Learning since the early 90's, relies on: i) a weak learner (extracting hypotheses, aka classifiers, with error probability less than $1/2 - \epsilon$, $\epsilon > 0$); ii) a diversification heuristics used to extract sufficiently diverse classifiers; iii) a voting mechanism, aggregating the diverse classifiers constructed [1, 8]. If the classifiers are sufficiently diverse and their errors are independent, then their majority vote will reach an arbitrarily low error rate on the training set as the number of classifiers increases [6]. Therefore, up to some restrictions on the classifier space [25], the generalization error will also be low[1].

The most innovative aspect of Ensemble Learning w.r.t. the Machine Learning literature concerns the diversity requirement, implemented through parallel or sequential heuristics. In Bagging, diversity is enforced by considering independent sub-samples of the training set, and/or using different learning parameters [1]. Boosting iteratively constructs a sequence of classifiers, where each classifier focuses on the examples misclassified by the previous ones [8].

Diversity is also a key feature of Evolutionary Computation (EC): in contrast with all other stochastic optimization approaches, evolutionary algorithms proceed by evolving a population of solutions, and the diversity thereof has been stressed as a key factor of success since the beginnings of EC. Deep similarities between Ensemble Learning and EC thus appear; in both cases, diversity is used to escape from local minima, where any single "best" solution is only too easily trapped. Despite this similarity, Evolutionary Learning has most often (with some notable exceptions, see [14, 16, 18] among others) focused on single-hypothesis learning, where some single best-of-run hypothesis is returned as the solution.

However, the evolutionary population itself could be used

[1]In practice, the generalization error is estimated from the error on a test set, disjoint from the training set. The reader is referred to [4] for a comprehensive discussion about the comparative evaluation of learning algorithms.

as a pool for recruiting the elements of an ensemble, enabling "Ensemble Learning for Free". Previous work along this line will be described in Section 2, mostly based on using an evolutionary algorithm as weak learner [17], or using evolutionary diversity-enforcing heuristics [16, 18].

In this paper, the "Evolutionary Ensemble Learning For Free" claim is empirically examined along two directions. The first direction is that of the classifier diversity; a new learning-oriented fitness function is proposed, inspired by the co-evolution framework [13] and generalizing the diversity-enforcing fitness proposed by [18]. The second direction is that of the selection of the ensemble classifiers within the evolutionary population(s). Selecting the best classifiers in a pool amounts to a feature selection problem, that is, a combinatorial optimization problem [12]. A greedy set-covering approach is used, build on a margin-based criterion inspired by Schapire *et al.* [23]. Finally, the paper presents two Evolutionary Ensemble Learning (EEL) approaches, called *Off*-EEL and *On*-EEL, respectively tackling the selection of the ensemble classifiers in the final population, or along evolution.

Paper structure is as follows. Section 2 reviews and discusses some work relevant to Evolutionary Ensemble Learning. Section 3 describes the two proposed approaches *Off*-EEL and *On*-EEL, introducing the specific fitness function and the ensemble classifier selection procedure. Experimental results based on benchmark problems from the UCI repository are reported in Section 4. The paper concludes with some perspectives for further research, discussing the priorities for a tight coupling of Ensemble Learning with Evolutionary Optimization in terms of dynamic systems [22].

2. RELATED WORK

Interestingly, some early approaches in Evolutionary Learning were rooted on Ensemble Learning ideas[2]. The Michigan approach [14] evolves a population made of rules, whereas the Pittsburgh approach evolves a population made of sets of rules. What is gained in flexibility and tractability in the Michigan approach is compensated by the difficulty of assessing a single rule, for the following reason. A rule usually only covers a part of the example space; gathering the best rules (e.g. the rules with highest accuracy) does not result in the best ruleset. Designing an efficient fitness function, such that a good quality ruleset could be extracted from the final population, was found a tricky task.

In the last decade, Ensemble Learning has been explored within Evolutionary Learning, chiefly in the context of Genetic Programming (GP). A first trend directly inspired from Bagging and Boosting aims at reducing the fitness computation cost [7, 16] and/or dealing with datasets which do not fit in memory [24]. For instance, Iba [16] divided the GP population into several sub-populations which are evaluated on subsets of the training set. Folino *et al.* [7] likewise sampled the training set in a Bagging-like mode in the context of parallel cellular GP. Song *et al.* [24] used Boosting-like heuristics to deal with training sets that do not fit in memory; the training set is divided into folds, one of which is loaded in memory and periodically replaced; at each generation, small subsets are selected from the current fold to compute the fitness function, where the selection is based on a mixture of uniform and Boosting-like distributions.

The use of Evolutionary Algorithms as weak learners within a standard Bagging or Boosting approach has also been investigated. Boosting approaches for GP have been applied for instance to classification [21] or symbolic regression [17]: each run delivers a GP tree minimizing the weighted sum of the training errors, and the weights were computed as in standard Boosting [8]. While such ensembles of GP trees result, as expected, in a much lower variance of the performance, they do not fully exploit the population-based nature of GP, as independent runs are launched to learn successive classifiers.

Liu *et al.* [18] proposed a tight coupling between Evolutionary Algorithms and Ensemble Learning. They constructed an ensemble of Neural Networks, using a modified back-propagation algorithm to enforce the diversity of the networks; specifically, the back-propagation aims at both minimizing the training error and maximizing the negative correlation of the current network with respect to the current population. Further, the fitness associated to each network is the sum of the weights of all examples it correctly classifies, where the weight of each example is inversely proportional to the number of classifiers that correctly classify this example. While this approach nicely suggests that ensemble learning is a Multiple Objective Optimization (MOO) problem (minimize the error rate and maximize the diversity), it classically handles the MOO problem as a fixed weighted sum of the objectives.

The MOO perspective was further investigated by Chandra and Yao in the DIVACE system, a highly sophisticated system for the multi-level evolution of ensemble of classifiers [2, 3]. In [3], the top-level evolution simultaneously minimizes the error rate (accuracy) and maximizes the negative correlation (diversity). In [2], the negative correlation-inspired criterion is replaced by a *pairwise failure crediting*; the difference concerns the misclassification of examples that are correctly classified by other classifiers. Finally, the ensemble is constructed either by keeping all classifiers in the final population, or by clustering the final population (after their phenotypic distance) and selecting a classifier in each cluster.

While the MOO perspective nicely captures the interplay of the accuracy and diversity goals within Ensemble Learning, the selection of the classifiers in the genetic pool as done in [2, 3] does not fully exploit the possibilities of evolutionary optimization, in two respects. On the one hand, it only considers the final population that usually involves up to a few hundred classifiers, while learning ensembles commonly involve some thousand classifiers. On the other hand, clustering-based selection proceeds on the basis of the phenotypic distance between classifiers, considering again that all examples are equally important, while the higher stress put on harder examples is considered the source of the better Boosting efficiency [5].

3. ENSEMBLE LEARNING FOR FREE

After the above discussion, Evolutionary Ensemble Learning (EEL) involves two critical issues: i) how to enforce both the predictive accuracy and the diversity of the classifiers in the population, and across generations; ii) how to best select the ensemble classifiers, from either the final population only or all along evolution.

[2]Learning Classifier Systems (LCS, [14, 15]) are mostly devoted to Reinforcement Learning, as opposed to Supervised Learning; therefore they will not be considered in the paper.

Two EEL frameworks have been designed to study these interdependent issues. The first one dubbed *Offline Evolutionary Ensemble Learning* (*Off*-EEL) constructs the ensemble from the final population only. The second one, called *Online Evolutionary Ensemble Learning* (*On*-EEL), gradually constructs the classifier ensemble as a selective archive of evolution, where some classifiers are added to the archive at each generation.

Both approaches combine a standard generational evolutionary algorithm with two interdependent components: a new diversity-enhancing fitness function, and a selection mechanism. The fitness function, presented in Section 3.1 and generalizing the fitness devised by Liu *et al.* [18], is inspired from co-evolution [13]. The selection process is used to extract a set of classifiers from either the final population (*Off*-EEL) or the current archive plus the current population (*On*-EEL), and proceeds by greedily maximizing the ensemble margin (Section 3.2).

Only binary or multi-class classification problems are considered in this paper. The decision of the classifier ensemble is the majority vote among the classifiers (ties being arbitrarily broken).

3.1 Diversity-enforcing Fitness

Traditionally, Evolutionary Learning maximizes the number of correctly classified training examples (or equivalently minimizes the error rate). However, examples are not equally informative; therefore a rule correctly classifying a hard example (e.g. close to the frontiers of the target concept) is more interesting and should be more rewarded than a rule correctly classifying an example which is correctly classified by almost all rules.

Co-evolutionary learning, first pioneered by Hillis [13], nicely takes advantage of the above remark, gradually forging more and more difficult examples to enforce the discovery of high-quality solutions. Boosting proceeds along the same lines, gradually putting the stress on the examples which have not been successfully predicted so far.

A main difference between both frameworks is that Boosting exploits a finite set of labelled examples, while co-evolutionary learning has an infinite supply of labelled examples (since it embeds the oracle). A second difference is that the difficulty of an example depends on the whole sequence of classifiers in Boosting, whereas it only depends on the current classifier population in co-evolution. In other words, Boosting is a memory-based process, while co-evolutionary learning is a memoryless one. Both approaches thus suffer from opposite weaknesses. Being a memory-based process, Boosting can be misled by noisy examples; consistently misclassified, these examples eventually get heavy weights and thus destabilize the Boosting learning process. Quite the contrary, co-evolution can forget what has been learned during early stages and specific heuristics, e.g. the so-called Hall-of-Fame, archive of best-so-far individuals, are required to prevent co-evolution from cycling in the learning landscape [20].

Based on these ideas, the fitness of classifiers is defined in this work from a set of reference classifiers noted $\mathcal{Q}$. The hardness of every training example x is measured after the number of classifiers in $\mathcal{Q}$ which misclassify x. The fitness of every classifier h is then measured by the cumulated hardness of the examples that are correctly classified by h.

Three remarks can be made concerning this fitness function. Firstly, contrasting with standard co-evolution, there is no way classifiers can "unlearn" to classify the training examples, since the training set is fixed. Secondly, as in Boosting, the fitness of a classifier reflects its diversity with respect to the reference set. Lastly, the classifier fitness function is highly multi-modal compared to the simple error rate: good classifiers might correctly classify many easy examples, or sufficiently many hard enough examples, or a few very hard examples.

Formally, let $\mathcal{E} = \{(\mathbf{x}_i, y_i),\ \mathbf{x}_i \in \mathcal{X},\ y_i \in Y,\ i = 1 \ldots n\}$ denote the training set (referred to as set of fitness cases in the GP context); each fitness case or example $(\mathbf{x}_i, y_i)$ is composed of an instance $\mathbf{x}_i$ belonging to the instance space $\mathcal{X}$ and the associated label y_i belonging to a finite set Y. Any classifier h is a function mapping the instance space $\mathcal{X}$ onto Y. The loss function ℓ is defined as $\ell : Y \times Y \mapsto \mathbb{R}$, where $\ell(y, y')$ is the (real valued) error cost of predicting label y instead of the true label y'.

The hardness or weight of every training example $(\mathbf{x}_i, y_i)$, noted $w_i^{\mathcal{Q}}$, or w_i when the reference set $\mathcal{Q}$ is clear from the context, is the average loss incurred by the reference classifiers on $(\mathbf{x}_i, y_i)$:

$$w_i = \frac{1}{|\mathcal{Q}|} \sum_{h \in \mathcal{Q}} \ell(h(\mathbf{x}_i), y_i). \tag{1}$$

The cumulated hardness fitness $\mathcal{F}$ is finally defined as follows: $\mathcal{F}(h)$ is the sum over all training examples that are correctly classified by h, of their weight w_i raised to power γ. Parameter γ governs the importance of the weights w_i (the cumulated hardness boils down to the number of correctly classified examples for $\gamma = 0$) and thus the diversity pressure.

$$\mathcal{F}(h) = \sum_{\substack{i=1\ldots n \\ h(\mathbf{x}_i)=y_i}} w_i^{\gamma} \tag{2}$$

Parameter γ can also be adjusted depending on the level of noise in the dataset. As noisy examples typically reach high weights, increasing the value of γ might lead to retain spurious hypotheses, which happen to correctly classify a few noisy examples. When ℓ is set to the step loss function ($\ell(y, y') = 0$ if $y = y'$, 1 otherwise) and γ is set to 1, the above fitness function is the same as the one used by Liu *et al.* [18]. The value of γ is set to 2 in the experiments (Section 4).

3.2 Ensemble Selection

As noted earlier on, the selection of classifiers in a pool $\mathcal{H} = \{h_1, \ldots, h_T\}$ in order to form an efficient ensemble is formally equivalent to a feature selection problem. The equivalence is seen by replacing the initial instance space $\mathcal{X}$ with the one defined from the classifier pool, where each instance $\mathbf{x}_i$ is redescribed as the vector $(h_1(\mathbf{x}_i), \ldots, h_T(\mathbf{x}_i))$. Feature selection algorithms [12] could thus be used for ensemble selection; unfortunately, feature selection is one of the most difficult Machine Learning problems.

Therefore, a simple greedy selection process is used in this paper to select the classifiers in the diverse pools considered by the *Off*-EEL (Section 3.3) and *On*-EEL (Section 3.4) algorithms. The novelty is the selection criterion, generalizing the notion of margin [11, 23] to an ensemble of examples as follows.

Formally, let $\mathcal{L}$ denote the current ensemble, initialized to the classifier h^* with minimum error rate in $\mathcal{H}$. For each

Figure 1: Pseudo-code of `Ensemble-Selection`(Classifier pool $\mathcal{H}$, training set $\mathcal{E}$, initial classifier ensemble $\mathcal{L}_0$).

1. Let $t = 1$, and $\mathcal{H}_1$ be the set $\mathcal{H}$ with duplicate individuals removed
2. While $\mathcal{H}_t$ is not empty:
   (a) Let $h_t^* = \text{argmax}_{h \in \mathcal{H}_t} (\mathcal{L}_{t-1} \cup \{h\})$ after the margin-based order relation of Equation 5
   (b) Let $\mathcal{H}_{t+1} = \mathcal{H}_t \backslash \{h_t^*\}$ (remove h_t^* from $\mathcal{H}_t$)
   (c) Let $\mathcal{L}_t = \mathcal{L}_{t-1} \cup \{h_t^*\}$ (and add it to $\mathcal{L}_t$)
   (d) $t = t + 1$
3. Return $\mathcal{L}^*$, the classifier ensemble in $\{\mathcal{L}_0 \dots \mathcal{L}_{t-1}\}$ that achieves the lowest error rate on $\mathcal{E}$, selecting the smallest ensemble in case of ties.

example $(\mathbf{x}_i, y_i)$, let its margin m_i be defined as follows. Let y_i' be the class most frequently associated to $\mathbf{x}_i$ by the classifiers in $\mathcal{L}$, such that y_i' is different from the true class y_i. Let c_i (respectively c_i') denote the number of classifiers in $\mathcal{L}$ associating class y_i (resp. y_i') to $\mathbf{x}_i$. Then margin m_i is defined as $c_i - c_i'$. A positive margin thus denotes the fact that the example is correctly classified by the majority vote; the higher the margin, the more confident the ensemble prediction. Conversely, a negative margin denotes an error; the ensemble misclassifies the example as belonging to class y_i'; the more negative the margin, the more classifiers need to be added to the ensemble in order to correctly classify $\mathbf{x}_i$.

Let K denote the number of classes of the problem and $|A|$ the size of a set A. The above definitions then read:

$$y_i' = \underset{\substack{k=1\dots K \\ k \neq y_i}}{\text{argmax}} |\{h_j(\mathbf{x}_i) = k,\ h_j \in \mathcal{L}\}|, \quad (3)$$

$$m_i = |\{h_j(\mathbf{x}_i) = y_i,\ h_j \in \mathcal{L}\}| - |\{h_j(\mathbf{x}_i) = y_i',\ h_j \in \mathcal{L}\}|. \quad (4)$$

Initially, the quality of ensemble $\mathcal{L}$ was measured after its minimum margin when $(\mathbf{x}_i, y_i)$ ranges over the training set, and the selection process aimed at maximizing the minimum margin likewise Boosting [22]. However, it turned out experimentally that the minimum margin alone is too coarse a criterion, leading to many ties. Thus, a finer grained criterion, based on the margin histogram, has finally been defined.

Let $c(\mathcal{L}, m)$ denote the number of training examples with margin m after $\mathcal{L}$. An order relation on classifier ensembles $\mathcal{L}$ and $\mathcal{L}'$ can then be defined by comparing $c(\mathcal{L}, m)$ and $c(\mathcal{L}', m)$ for increasing values of m; the best ensemble is the one with lesser number of examples with the smallest margin.

$$\mathcal{L} < \mathcal{L}' \text{ iff } \exists\ m_0 \text{ s.t. } \begin{cases} \forall m < m_0,\ c(\mathcal{L}, m) = c(\mathcal{L}', m) \\ c(\mathcal{L}, m_0) > c(\mathcal{L}', m_0) \end{cases} \quad (5)$$

The pseudo-code of the ensemble selection algorithm is displayed in Figure 1. It starts with a classifier pool $\mathcal{H}$, a set of training examples $\mathcal{E}$ and an initial set of classifiers $\mathcal{L}_0$. It then iteratively moves all classifiers from $\mathcal{H}$ into $\mathcal{L}$, based on the above order on ensembles. Ultimately, the ensemble with lowest error rate on $\mathcal{E}$ in the ensemble sequence $\mathcal{L}_0 \dots \mathcal{L}_{t-1}$ is selected.

3.3 Offline Evolutionary Ensemble Learning

Off-EEL is a two-step process. It firstly runs a standard evolutionary learning algorithm. The approach does not make any requirement on the genetic search space, that is the classifier space; the designer can run *Off*-EEL on the top of her favorite evolutionary learning algorithm, searching for linear classifiers, neural nets, rule systems, or genetic programs. The only required modification concerns the fitness function, which is set to the diversity-enhancing fitness described in Section 3.1, taking the whole current population as set of reference classifiers. In contrast with Boosting, the process does not maintain any memory about the examples; their weights are recomputed from scratch at each generation. While Boosting might result in exponentially increasing the weight of hard or possibly noisy examples, *Off*-EEL thus keeps the weight of each training example bounded, and thereby avoids the instability due to the data noise.

The second step achieves the ensemble selection based on the margin-based criterion (Section 3.2 and Figure 1). It uses the final population as pool of classifiers $\mathcal{H}$, and initializes the classifier ensemble to the classifier h^* that has the smallest error rate on the training set in the population ($\mathcal{L}_0 = \{h^*\}$).

3.4 Online Evolutionary Ensemble Learning

In contrast with *Off*-EEL, *On*-EEL interleaves evolutionary learning and ensemble selection; at each generation the classifier ensemble is updated using the current population.

At generation 1, the classifier ensemble is initialized to the classifier that minimizes the error rate on the training set. In further generations, the current population is evolved using the diversity-enhancing fitness function with the current ensemble as reference set (Section 3.1), and the ensemble selection algorithm (Figure 1) is launched, using the current population as classifier pool $\mathcal{H}$, and the current classifier ensemble as $\mathcal{L}_0$. The pseudo-code of *On*-EEL is given in Figure 2.

Notably, *Off*-EEL and *On*-EEL achieve different Exploration *vs* Exploitation trade-offs. In *Off*-EEL, the set of reference classifiers is the current population; the fitness function thus favors both accurate and diverse classifiers in each generation. The ensemble selection algorithm is launched only once, on a high quality and diversified pool of classifiers.

In *On*-EEL, the set of reference classifiers is the current

Figure 2: Pseudo-code of `On-EEL`(training set $\mathcal{E}$).

1. Let $\mathcal{P}_1$ be the first evolutionary population, and h^* the classifier with minimal error rate on $\mathcal{E}$.
2. $\mathcal{L}_1 =$ `Ensemble-Selection`$(\mathcal{P}_1, \mathcal{E}, \{h^*\})$
3. For $t = 2 \dots T$:
   (a) Evolve $\mathcal{P}_{t-1} \rightarrow \mathcal{P}_t$, using $\mathcal{L}_{t-1}$ as reference set.
   (b) $\mathcal{L}_t =$ `Ensemble-Selection`$(\mathcal{P}_t, \mathcal{E}, \mathcal{L}_{t-1})$
4. Return $\mathcal{L}_T$.

classifier ensemble; like in Boosting, the goal is to find classifiers which overcome the errors of the past classifiers. While the ensemble selection algorithm is launched at every generation, it uses the biased current population as classifier pool. In fact, *On*-EEL addresses a dynamic optimization problem; if the classifier ensemble significantly changes between one generation and the next, the fitness landscape will change accordingly and several evolutionary generations might be needed to accommodate this change. On the other hand, as long as the current population does not perform well, the ensemble selection algorithm is unlikely to select further classifiers in the current ensemble; the fitness landscape thus remains stable. The population diversity does not directly result from the fitness function as in the *Off*-EEL case; rather, it relates with the dynamic aspects of the fitness function.

4. EXPERIMENTAL SETTING

This section describes the experimental setting used to assess the EEL framework.

4.1 Datasets

Experiments are conducted on the six UCI datasets [19] presented in Table 1. The performance of each algorithm is measured after a standard stratified 10-fold cross-validation procedure. The dataset is partitioned into 10 folds with same class distribution. Iteratively, all folds but the i-th one are used to train a classifier, and the error rate of this classifier on the remaining i-th fold is recorded. The performance of the algorithm is averaged over 10 runs for each fold, and over the 10 folds.

4.2 Classifier Search Space

As mentioned earlier on, evolutionary ensemble learning can accommodate any type of classifier; *Off*-EEL and *On*-EEL could consider neural nets, genetic programs or decision lists as genotypic search space. Our experiments will consider the most straightforward classifiers, namely separating hyperplanes, as these can easily be inspected and compared. Formally, let $\mathcal{X} = \mathbb{R}^d$ be the instance space, a separating hyperplane classifier h is characterized as $(\mathbf{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ with $h(\mathbf{x}) = < \mathbf{w}, \mathbf{x} > - b$ ($< \mathbf{w}, \mathbf{x} >$ denotes the scalar product of $\mathbf{w}$ and $\mathbf{x}$). The search for a separating hyperplane is amenable to quadratic optimization, with:

$$\mathcal{F}(h) = \sum_{i=1 \dots n} (h(\mathbf{x}_i) - y_i)^2. \quad (6)$$

As the above optimization problem can be tackled using standard optimization algorithms, it provides a well-founded baseline for comparison. Specifically, the first goal of the experiments is thus to assess the merits of evolutionary ensemble learning against three other approaches.

The first baseline algorithm referred to as Least Mean Square (LMS) uses a stochastic gradient algorithm to determine the optimal separating hyperplane in the sense of criterion given by Equation 6 (see pseudo-code in Figure 3).

The second baseline algorithm is an elementary evolutionary algorithm, producing the best-of-run separating hyperplane such that it minimizes the (training) error rate[3].

The third reference algorithm is the prototypical ensemble learning algorithm, namely AdaBoost with its default parameters [8]. AdaBoost uses simple decision stumps [23] baseline algorithm as weak learner (more on this below).

The learning error is classically viewed as composed from a variance term and a bias term [1]. The bias term measures how far the target concept tc is from the classifier search space $\mathcal{H}$, that is, from the best classifier h^* in this search space. The variance term measures how far away one can wander from h^*, wrongly selecting other classifiers in $\mathcal{H}$ (overfitting).

The comparison of the first and second baseline algorithms gives some insight into the intrinsic difficulty of the problem. Stochastic gradient (LMS) will find the global optimum for criterion given by Equation 6, but this solution optimizes at best the training error. The comparison between the solutions respectively found by LMS and the simple evolutionary algorithm will thus reflect the learning variance term.

Similarly, the comparison of the first baseline algorithm and AdaBoost gives some insight into how the ensemble improves on the base weak learner; this improvement can be interpreted in terms of variance as well as in terms of bias (since the majority vote of decision stumps allows for describing more complex regions than simple separating hyperplanes alone).

4.3 Experimental Setting

The parameters for the LMS algorithm (see Figure 3) are as follows: the training rate, set to $\eta(t) = 1/(n\sqrt{t})$, decreases over the training epochs; the maximum number of epochs allowed is $T = 10000$; the stopping criterion is when the difference in the error rates over two consecutive epochs, is less that some threshold ϵ ($\epsilon = 10^{-7}$). Importantly, LMS requires a preliminary normalization of the dataset, (e.g. $\forall i = 1 \dots n,\ \mathbf{x}_i \in [-1, 1]^d$). The final result is the error on the test set, averaged over 10 runs for each fold (because of the stochastic reordering of the training set) and averaged over 10 folds.

The classical AdaBoost algorithm [8] uses simple decision stumps [23], and the number of Boosting iterations is limited to 2000. Decision stumps are simple binary classifiers that

[3]For 3-classes problems, e.g. `bos` or `cmc`, the classifier is characterized as two hyperplanes, respectively separating class 0 (resp. class 1) from the other two classes. In case of conflict (the example is simultaneously classified in class 0 by the first classifier and in class 1 by the second classifier), the tie is broken arbitrarily.

Table 1: UCI datasets used for the experimentations.

Dataset	Size	# features	# classes	Application domain
`bcw`	683	9	2	Wisconsin's breast cancer, 65 % benign and 35 % malignant.
`bld`	345	6	2	BUPA liver disorders, 58 % with disorders and 42 % without disorder.
`bos`	508	13	3	Boston housing, 34 % with median value $v < 18.77$ K\$, 33 % with $v \in]18.77, 23.74]$, and 33 % with $v > 23.74$.
`cmc`	1473	9	3	Contraceptive method choice, 43 % not using contraception, 35 % using short-term contraception, and 23 % using long-term contraception.
`pid`	768	8	2	Pima indians diabetes, 65 % tested negative and 35 % tested positive for diabetes.
`spa`	4601	57	2	Junk e-mail classification, 61 % tested non-junk and 39 % tested junk.

Figure 3: Least-mean square training algorithm.

1. Initialize $\mathbf{w} = 0$ and $b = 0$
2. For $t = 1 \dots T$:
   (a) Shuffle the dataset $\mathcal{E} = \{(\mathbf{x}_i, y_i),\ i = 1 \dots n\}$
   (b) For $i = 1 \dots n$:

$$\begin{aligned} a_i &= < \mathbf{w}, \mathbf{x}_i > - b \\ \Delta_i &= 2\eta(t)(a_i - y_i) \\ \mathbf{w} &= \mathbf{w} + \Delta_i \mathbf{x}_i \\ b &= b - \Delta_i \end{aligned}$$

   (c) $Err_t = \sqrt{\frac{1}{n} \sum_{i=1\dots n} (a_i - y_i)^2}$ (RMS error)
   (d) If $|Err_t - Err_{t-1}| < \epsilon$, stop

Table 2: Parameters for the real-valued GA.

Parameter	Value
Population size	500
Termination criteria	100000 fitness evaluations
Tournament size	2
Initialization range	[-1,1]
SBX crossover prob.	0.3
SBX crossover n-value	$n = 2$
Gaussian mutation prob.	0.1
Gaussian mutation std. dev.	$\sigma = 0.05$

classify data according to a threshold value on one of the features of the data set. If the feature value of a given data is less (or greater) than the threshold, the data is assigned to a given class, otherwise it is assigned to another class. Decision stumps are trained deterministically, by looping over all features and all features threshold for a given training dataset, selecting the feature, threshold, and comparison operation on the threshold ($>$ or $<$) that maximize the classification accuracy on the training data set. Decision stumps are the simplest possible linear classifiers, but generate good results in combination with AdaBoost.

The elementary evolutionary algorithm is a real-valued generational GA using SBX crossover, Gaussian mutations, and tournament selection. The search space is $\mathbb{R}^{d+1}$ for binary classification problems, and $\mathbb{R}^{2d+2}$ for ternary classification problem, where d is the number of attributes in the problem domain. The evolutionary parameters are detailed in Table 2. All experiments with the real-valued GA rely on the C++ framework Open BEAGLE [9, 10].

5. RESULTS

This section reports on the experimental results obtained by *Off*-EEL and *On*-EEL, compared to the three baseline methods respectively noted LMS (optimal linear classifier), GA (genetically evolved linear classifier) and Boosting (ensemble of decision stumps), on the six UCI data sets described in Table 1. For each method and problem, the average test error (over 100 independent runs as described in Section 4) and the associated standard deviation are displayed in Table 3. The average computational effort of *Off*-EEL for a run ranges from 30 seconds (on problem `bld`) to 20 minutes (on problem `spa`), on AMD Athlon 1800+ computers with 1G of memory. For *On*-EEL, the average computational effort for a run ranges from 2 hours (on problem `pid`) to 24 hours (on problem `spa`), on the same computers.

With respect to the baseline algorithms, a first remark is that the LMS-based classifier is significantly outperformed by all other methods, on all problems but one (`pid`). This is explained as the criterion given by Equation 6 uselessly overconstrains the learning problem, replacing a set of linear inequalities with the minimization of the sum of quadratic terms. Similarly, the single-hypothesis evolutionary learning is dominated by all other methods on all problems but one (`bcw`). Boosting shows its acknowledged efficiency as it is the best algorithm on two out of six problems (*Off*-EEL and Boosting are both best performers for the `cmc` problem).

Off-EEL is the best method for three out of six problems tested. Compared to AdaBoost, it generates ensemble with lower test error rate on four problems, with a tie for the `cmc` problem, and AdaBoost being the best on `spa` problem. In all cases, the number of classifiers is lower, with an average between 235 and 335 classifiers for *Off*-EEL compared with more than 750 on all problems but `bcw` for Boosting. This is understandable given that the ensembles are built with *Off*-EEL starting from a population of 500 individuals. This raises the question on whether the evolutionary learning accuracy could be improved by considering larger population sizes. But it should not be forgotten that the decision stumps classifier making the AdaBoost ensembles are significantly simpler than the evolved linear discriminants of *Off*-EEL. No clear conclusion can thus be made on the relative complexity of the ensembles generated by *Off*-EEL compared to Boosting.

Table 3: Results on the UCI datasets based on 10-folds cross-validation, using 10 independent runs over each fold. Values are averages (standard deviations) over the 100 runs. Statistical tests are p-values of paired t-tests on the test error rate compared to that of the best method on the dataset (in bold).

Measure	LMS	GA	Boosting	*Off*-EEL	*On*-EEL
			`bcw`		
Train error	3.9% (0.2%)	1.8% (0.2%)	0.0% (0.0%)	1.4% (0.2%)	0.4% (0.4%)
Test error	4.0% (1.6%)	**3.2**% (1.7%)	5.3% (2.0%)	3.4% (1.7%)	3.5% (2.0%)
Test error p-value	0.00	–	0.00	0.09	0.04
Ensemble size	–	–	291.6 (68.2)	235.6 (66.8)	116.3 (278.2)
			`bld`		
Train error	29.8% (0.9%)	25.4% (1.2%)	0.0% (0.0%)	20.9% (1.5%)	18.9% (2.0%)
Test error	30.4% (6.6%)	32.7% (6.6%)	30.4% (5.4%)	**29.2**% (7.4%)	29.5% (8.4%)
Test error p-value	0.04	0.00	0.14	–	0.64
Ensemble size	–	–	1081.4 (166.1)	301.0 (37.9)	294.1 (154.2)
			`bos`		
Train error	32.2% (1.3%)	23.4% (4.1%)	0.0% (0.0%)	16.7% (1.9%)	20.9% (2.3%)
Test error	34.0% (6.7%)	30.7% (7.5%)	26.9% (4.2%)	**22.7**% (5.7%)	26.2% (7.2%)
Test error p-value	0.00	0.00	0.00	–	0.00
Ensemble size	–	–	761.1 (40.8)	303.8 (41.4)	2960.9 (2109.3)
			`cmc`		
Train error	51.6% (0.4%)	45.7% (1.4%)	43.3% (0.7%)	42.9% (1.2%)	43.9% (1.4%)
Test error	51.8% (2.5%)	50.4% (3.9%)	**46.8**% (2.9%)	**46.8**% (3.9%)	47.7% (3.9%)
Test error p-value	0.00	0.00	0.99	–	0.04
Ensemble size	–	–	4000.0 (0.0)	326.4 (35.7)	2707.7 (1696.1)
			`pid`		
Train error	22.0% (0.6%)	20.2% (0.7%)	0.6% (0.5%)	19.8% (0.7%)	20.0% (0.8%)
Test error	**22.8**% (3.5%)	24.2% (3.9%)	28.1% (5.0%)	24.0% (4.0%)	24.0% (3.9%)
Test error p-value	–	0.00	0.00	0.00	0.00
Ensemble size	–	–	1978.1 (43.0)	309.5 (37.6)	1196.3 (765.7)
			`spa`		
Train error	11.1% (0.4%)	7.9% (0.5%)	1.4% (0.1%)	6.1% (0.2%)	7.6% (0.8%)
Test error	11.3% (1.2%)	9.0% (1.3%)	**5.7**% (0.8%)	6.7% (1.2%)	8.3% (1.4%)
Test error p-value	0.00	0.00	–	0.00	0.00
Ensemble size	–	–	2000.0 (0.0)	331.1 (28.4)	6890.0 (2938.1)

Despite its larger ensemble size, *On*-EEL is dominated by *Off*-EEL on all problems but `pid`, where both approaches generate identical test error rates. A tentative explanation stems from the nature of the two approaches, with *Off*-EEL having a clear algorithm organized in two stages, classifiers evolution with diversity-enhancing fitness followed by ensemble construction, while *On*-EEL is more complex, with a succession of ensemble construction and classifiers evolution with diversity-enforcing measure taken relatively to the current ensemble. The dynamics of *On*-EEL is hard to understand, but it can be speculated that the iterative construction of the ensemble (without individual removal) is prone to be stuck in local optima. Indeed, the "construction path" taken to build the ensemble begins with a selection of some (supposed poor) individuals at the beginning of the evolution. As these individuals cannot be removed from the ensemble, they significantly influence the choice of other individuals, biasing and possibly misleading the whole process.

6. DISCUSSION AND PERSPECTIVES

This paper has examined the "Evolutionary Ensemble Learning for Free" claim, based on the fact that, since Evolutionary Algorithms maintain a population of solutions, it comes naturally to use these populations as a pool for building classifier ensembles.

Two main issues have been studied, respectively concerned with enforcing the diversity of the population of classifiers, and with selecting the classifiers either in the final population or along evolution.

The use of a co-evolution-inspired fitness function, generalizing [18], was found sufficient to generate diverse classifiers. As already noted, there is a great similarity between the co-evolution of programs and fitness cases [13] and the Boosting principles [8]; the common idea is that good classifiers are learned from good examples, while good examples are generalized by good classifiers. The difference between Boosting and co-evolution is that in Boosting, the training examples are not evolved; instead, their weights are updated. However, the uncontrolled growth of some weights, typically in the case of noisy examples, actually appears as the Achilles' heel of Boosting compared to Bagging. Basically, AdaBoost can be viewed as a dynamic system [22]; the possible instability or periodicity of this dynamic system has undesired consequences on the ensemble learning performance. The use of co-evolutionary ideas, even though the set of ensemble does not evolve, seems to increase the stability of the learning process.

The two EEL frameworks investigated in this paper can be considered as promising. *Off*-EEL constructs ensembles with best performances while needing little modifications over a traditional evolutionary algorithm, with a diversity-enhancing fitness and the construction of an ensemble from the final population. But the size of the ensembles generated suggests that bigger population would lead to bigger and possibly better ensembles. For the sake of scalability, this suggests that the ensemble should be gradually constructed along evolution, instead of considering only the final population. This has been explored with *On*-EEL, with lesser performance comparing to *Off*-EEL. It is suggested that ensemble construction with *On*-EEL is prone to be stuck in local minima, so some capability of removing individuals can be beneficial, at the risk of inducing an highly dynamic algorithm. Ultimately, the momentum and dynamics of EEL should be controlled by evolution itself, enforcing some trade-off between exploring new regions and preserving efficient optimization. This will be the subject of future researches.

Acknowledgments

This work was supported by postdoctoral fellowships from the ERCIM-SARIT (Europe), the Swiss National Science Foundation (Switzerland), and the FQRNT (Québec) to C. Gagné. The second and third authors gratefully acknowledge the support of the `Pascal` Network of Excellence IST-2002-506 778.

7. REFERENCES

[1] L. Breiman. Arcing classifiers. *Annals of Statistics*, 26(3):801–845, 1998.

[2] A. Chandra and X. Yao. Ensemble learning using multi-objective evolutionary algorithms. *J. of Mathematical Modelling and Algorithms*, 5(4):417–425, 2006.

[3] A. Chandra and X. Yao. Evolving hybrid ensembles of learning machines for better generalisation. *Neurocomputing*, 69:686–700, 2006.

[4] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923, 1998.

[5] T. G. Dietterich. Ensemble methods in machine learning. In *First Int. Workshop on Multiple Classifier Systems*, pages 1–15, 2000.

[6] R. Esposito and L. Saitta. Monte Carlo theory as an explanation of Bagging and Boosting. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI'03)*, pages 499–504, 2003.

[7] G. Folino, C. Pizzuti, and G. Spezzano. Ensemble techniques for parallel genetic programming based classifiers. In *Proc. of the European Conf. on Genetic Programming (EuroGP'03)*, pages 59–69, 2003.

[8] Y. Freund and R. Schapire. Experiments with a new Boosting algorithm. In *Proc. of the Int. Conf. on Machine Learning (ICML'96)*, pages 148–156, 1996.

[9] C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case-study. *Int. J. on Artificial Intelligence Tools*, 15(2):173–194, 2006.

[10] C. Gagné and M. Parizeau. Open BEAGLE: An evolutionary computation framework in C++. `http://beagle.gel.ulaval.ca`, 2006.

[11] R. Gilad-Bachrach, A. Navot, and N. Tishby. Margin based feature selection - theory and algorithms. In *Proc. of the Int. Conf. on Machine Learning (ICML'04)*, pages 43–50, 2004.

[12] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature Extraction: Foundations And Applications*. Springer-Verlag, 2006.

[13] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[14] J. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning, An Artificial Intelligence Approach*, volume 2, pages 593–623. Morgan Kaufmann, 1986.

[15] J. Holmes, P. Lanzi, W. Stolzmann, and S. Wilson. Learning classifier systems: New models, successful applications. *Information Processing Letters*, 82(1):23–30, 2002.

[16] H. Iba. Bagging, Boosting, and bloating in genetic programming. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 1053–1060, 1999.

[17] M. Keijzer and V. Babovic. Genetic programming, ensemble methods, and the bias/variance tradeoff – introductory investigations. In *Proc. of the European Conf. on Genetic Programming (EuroGP'00)*, pages 76–90, 2000.

[18] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Trans. on Evolutionary Computation*, 4(4):380–387, 2000.

[19] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1998.

[20] J. Paredis. Coevolving cellular automata: Be aware of the Red Queen! In *Proc. of the Int. Conf. on Genetic Algorithms (ICGA'97)*, pages 393–400, 1997.

[21] G. Paris, D. Robilliard, and C. Fonlupt. Applying Boosting techniques to genetic programming. In *Artificial Evolution 2001*, volume 2310 of *LNCS*, pages 267–278. Springer Verlag, 2001.

[22] C. Rudin, I. Daubechies, and R. E. Schapire. The dynamics of AdaBoost: Cyclic behavior and convergence of margins. *J. of Machine Learning Research*, 5:1557–1595, 2004.

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

[24] D. Song, M. I. Heywood, and A. N. Zincir-Heywood. Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Trans. on Evolutionary Computation*, 9(3):225–239, 2005.

[25] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New York, NY (USA), 1998.