Is a Greedy Covering Strategy an Extreme Boosting?

Roberto Esposito[1] and Lorenza Saitta[2]

[1] Universitá degli Studi di Torino, Italy
esposito@di.unito.it
[2] Universitá del Piemonte Orientale, Alessandria, Italy
saitta@mfn.unipmn.it

Abstract. A new view of majority voting as a Monte Carlo stochastic algorithm is presented in this paper. Relation between the two approaches allows Adaboost's example weighting strategy to be compared with the greedy covering strategy used for a long time in Machine Learning. The greedy covering strategy does not clearly show overfitting, it runs in at least one order of magnitude less time, it reaches zero error on the training set in few trials, and the error on the test set is most of the time comparable to that exhibited by AdaBoost.

1 Introduction

Majority voting classification algorithms, such as boosting [10, 6] or bagging [3] are very popular nowadays because of the superior performances shown experimentally on a number of data sets (see, for example, [1, 8]). Majority voting methods increase the accuracy of classifiers acquired by weak learners combining their predictions.

An intriguing property of these algorithms is their robustness with respect to overfitting. In fact, their generalization error does not appear to increase, even when the number of voting classifiers ranges in the thousands. A rather convincing argument to explain this behavior is that boosting increases the number of learning examples that have a large classification margin [9].

In this paper we concentrate, for the sake of simplicity, on the case of binary classification.

In the effort to understand why and when boosting works, links with other approaches, such as logistic regression [7] and game theory [5], have been established. In this paper we offer a new perspective, relating majority voting with Monte Carlo stochastic algorithms [2]. In fact, the Monte Carlo approach offers a technique to increase the performance of a simple algorithm by repeatedly running it on the same problem instance. Monte Carlo algorithms have been studied for a long time, and they offer several results that can possibly be transferred to the majority voting framework. For instance, realistic bounds on the number of iterations necessary to reach a given level of performances were already available [2, p.265].

In addition, a subclass of Monte Carlo algorithms (i.e., biased, consistent ones [2]) shows particularly interesting properties with respect to the link between performance increase and number of iterations. Then, a natural question is whether they correspond to some class of machine learning algorithms, which these properties could be transferred to. As it turns out, these special Monte Carlo algorithms correspond to the well known greedy covering strategy, where covered examples are removed at each run, and majority voting becomes an "at least one" combination rule. Then, while Monte Carlo theory suggests that these algorithms are particularly good, past machine learning experience does not confirm this statement. Understanding where the relationship breaks down may help in deepening our knowledge of both majority voting and greedy covering. In order to clarify the above issue, we have taken an experimental approach, using several artificially generated learning problems.

2 Monte Carlo algorithms

Given a class Π of problems, a Monte Carlo algorithm is a stochastic algorithm that, applied to any instance $x \in \Pi$, always outputs an answer, but, occasionally, this answer is incorrect [2]. In order for an algorithm to be Monte Carlo, any problem instance must have the same probability of being incorrect. More precisely, let p be a real number such that $\frac{1}{2} < p < 1$. A Monte Carlo algorithm is *p-correct* if the probability that it returns a correct answer is at least p on any problem instance[1]. The difference $(p - \frac{1}{2})$ is the advantage of the algorithm. Moreover, a Monte Carlo algorithm is said to be *consistent* if it never outputs two different correct solutions to the same instance.

Given a p-correct, consistent Monte Carlo algorithm $\mathsf{MC}(x)$, its probability of success can be increased by running it several time on the same instance, and choosing the most frequent answer[2].

More precisely, let ϵ and η be two positive real numbers, such that $\epsilon + \eta < 1/2$. Let $\mathsf{MC}(x)$ be a consistent and $(\frac{1}{2}+\epsilon)$-correct Monte Carlo algorithm. If we define:

$$n(\epsilon) = -\frac{2}{\log_2(1 - 4\epsilon^2)} \tag{1}$$

it is sufficient to call $\mathsf{MC}(x)$ at least

$$T = \left\lceil n(\epsilon) \cdot \log_2 \frac{1}{\eta} \right\rceil \tag{2}$$

times on x, and to return the most frequent answer, to obtain an algorithm that is still consistent and also $(1 - \eta)$-correct ([2, p.263]). We have "amplified" the advantage of $\mathsf{MC}(x)$.

[1] This statement is different from saying that the algorithm is correct on most problem instances, being only incorrect on a small subset of them.

[2] The consistency of the algorithm is fundamental for the amplification. For instance, running three times a consistent 0.75-correct Monte Carlo algorithm $\mathsf{MC}(x)$ and taking the most frequent answer leads to a 0.84-correct algorithm, whereas the resulting algorithm is only 0.71-correct, should $\mathsf{MC}(x)$ be not consistent.

2.1 Biased Monte Carlo algorithms

Let us consider now a Monte Carlo algorithm solving a decision problem, with only two answers: *true* and *false*. Suppose moreover that the algorithm is always correct when it outputs true, errors being only possible on the answer "false". Such an algorithm is said to be a *true-biased* Monte Carlo. With a true-biased Monte Carlo algorithm, majority voting on a sequence of runs is superfluous, because it is sufficient that the answer true be output a single time. More importantly, amplification occurs also for biased p-correct algorithms with $p < \frac{1}{2}$, provided that $p > 0$. More formally [2, p. 265]:

Definition 1. *Let Π be a class of problems and let s_0 be a possible output of a Monte Carlo algorithm $\mathsf{MC}(x)$. $\mathsf{MC}(x)$ is* s_0-biased *if there exists a subset X of Π such that:*

1. *$\mathsf{MC}(x)$ is always correct on instance x whenever $x \notin X$;*
2. *The correct solution to any $x \in X$ is s_0, but $\mathsf{MC}(x)$ may not always return the correct answer on these instances.*

Theorem 1 (Brassard and Bratley). *Running k times a consistent, s_0-biased, p-correct Monte Carlo algorithm (with $0 < p < 1$) yields a consistent, s_0-biased, $[1 - (1 - p)^k]$-correct algorithm.*

Then, in order to achieve a correctness level of $(1 - \eta)$, it is sufficient to run the algorithm at least a number of times:

$$T = \left\lceil \frac{\log_2 \eta}{\log_2(1 - p)} \right\rceil \tag{3}$$

2.2 Relations between Ensemble learning and Monte Carlo Algorithms

Let us consider now a learning context in which a weak learner A acquires decision rules $h(x) : X \to Y$, belonging to a set H. X is a set of instances and $Y = \{+1, -1\}$ is a binary set of classes. We will consider positive and negative instances labelled respectively +1 and -1.

If we would like to build a Monte Carlo algorithm out of a learning algorithm we would have to find a way to force the weak learner to behave in a stochastic way. Furthermore we shall want to let two calls to the learner be as independent as possible. We have at least two ways to face the problem: we can rewrite the weak learner from the scratch, or we can manipulate the learner input as a way to randomize the algorithm. The only input we can randomize in a learning context is the training set, then a reasonable choice is to iterate the weak learner providing it with different randomly chosen subsamples of the original training set. As long as this technique is successful, i.e. as long as independence between different calls is achieved, and provided that the weak algorithm outperforms random guessing, we will be able to improve its performances in the way we explained in Sect. 2. Bagging does exactly what we have just explained. It iterates

the weak algorithm several times providing it with different, randomly chosen, subsamples of the original training set and then it combines the hypotheses with a majority vote.

Since our goal is to increase the difference between two iterations of the same algorithm, we could think of better ways of perturbing the input. In particular it is very likely that the greatest difference between the hypotheses could be obtained by providing the algorithm with the examples that it has misclassified in previous iterations. In fact, since the goal of the weak algorithm is to classify as much examples as possible, it is very likely that a very different hypothesis would be built. AdaBoost reweighting scheme exploits this idea as a way to force the weak learner to induce very different hypotheses.

If we now consider biased Monte Carlo algorithms, we may wonder about what kind of combined classifiers they might correspond to. If a correspondence can be established, it would be reasonable to expect that the learning counterpart shows at least two advantages over more generic boosting methods: first of all, comparable error rates with a much smaller numbers of individual classifiers, and, second, the possibility of using very rough weak learners, because their error rate only needs to be greater than zero. Actually, it turns out that the learning counterpart of a consistent, true-biased and p-correct Monte Carlo algorithm is a greedy covering algorithm (GCA), with the set of positive examples as the X set in Definition 1. In fact, let us consider as weak learner A an algorithm that covers some positive examples and no negative ones. Then, at each repetition of A, we eliminate the already covered positive examples. At the end, when no positive example is left, the majority voting rule becomes an "at least one" rule. In fact, it is sufficient that one among the $h_t(x)$'s says $+1$ to classify the example as positive, due to the bias.

If we now try to make AdaBoost to fit into the Monte Carlo framework, it turns out that AdaBoost seems to lie between GCA and Bagging. In fact both its example weighting scheme and its weighted majority voting scheme are a compromise between the two algorithms. The elimination process performed by the biased Monte Carlo algorithm is a limit process of AdaBoost weighting scheme; besides, the at-least-one vote can be thought of as a hard way of weighting the hypotheses.

Given that the GCA is a biased Monte Carlo algorithm, we expect to obtain much smaller classifiers. This idea is appealing, as one of the drawback of boosting is the generation of incomprehensible classification rules. However, the machine learning field has dealt a long time with greedy covering algorithms, which did not prove to be very robust with respect to generalization error. Then, a pertinent question would be: why? The previous observation suggested us to test the following hypotheses:

1. a GCA allows very simple learners to be boosted in few runs, without bothering about their accuracy (provided that it is greater than zero);
2. GCA's should be prone to overfitting, whereas (most of the times) AdaBoost is not. Then, increasing the number of basic classifiers should let the test error of a GCA increase, contrarily to what happens to AdaBoost;

3. the simple politics adopted by GCA should perform poorly in difficult situations such as noisy datasets.

In order to test the previous hypotheses, we performed experiments on a set of artificial datasets, reported in the following section.

3 Experiments

In this section a description of the experiments we carried out to test the performances of AdaBoost, Bagging, and GCA under different conditions is reported. In particular we were interested in exploring under which conditions the former two algorithms could outperform the latter. Since we planned to use really weak hypotheses, we allowed the algorithms to run for a huge number of iterations (namely five thousands)[3] on twelve different artificial datasets.

The datasets were built on purpose and differ from each other in many aspects. There are three main datasets: *asia*, *triangles* and *fiveD*, which differ in shape of the target concept and in dimensionality. In particular, *asia* contains two fairly large clusters of points, *triangles* contains five small disjunctions, while *fiveD* contains five large balls in a five dimensional space.

The remaining nine datasets have been generated by adding three types of noise in the three main ones (see Subsection 3.1 for further details).

For each dataset the following parameters have been computed:

- η_t, i.e. the training error at step t;
- ω_t, i.e the test error at step t;
- ϵ_t, i.e. the error committed by the weak learner on the set used for its training

These values have been computed at each iteration for $1 \leq t \leq 100$, and every fifty iterations for $101 \leq t \leq 5000$. Five-fold cross-validation has been used to increase the confidence in the output.

3.1 Random Noises

In addition, we created nine noisy datasets by introducing three kinds of random noise in the three original noise-free datasets.

The first noise scheme (labels) randomly perturbs the labels given in the original dataset, no matter where the perturbed instance lies. Ten percent of the labels have been changed in this way.

The other two noise schemes take into account the place where the instances lie; in fact, only the instances that are near the borders of the concept have been changed with a probability that depends upon the dataset itself. The difference between these last two noise schemes is that in one of them (marginNoBayes) the label of the perturbed instance is changed, while in the other (marginBayes)

[3] We forced GCA to run for such a long time even if, as expected, this has lowered its performances.

the whole instance is reinserted in the dataset, once its label has been changed. This corresponds to deal with a target concept that may return two different labels if called twice (in other words its Bayes error is greater than zero)[4].

3.2 Weak Learners

The three algorithms have been tested using two different kinds of weak learners.

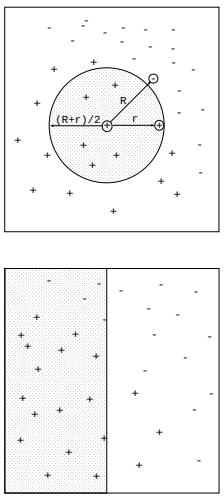

Fig. 1. The sphere and the stump inducers

Both algorithms search a very small hypothesis space and only slightly outperform random guess when used alone with a complex data set. Once again the algorithms have been chosen to be very different, in order to test the three ensemble learners in very different situations.

The first of the two algorithms is a stochastic, positively biased learner[5]. It induces a sphere in the example space, whose center is chosen at random among the positive examples. The radius of the sphere is optimized so that it maximize the coverage of positive examples while preserving its positive bias. A simple example of a hypothesis induced by this learner is reported in the top-most of the figures on the left. The radius is chosen in such a way as to lie half the way between the nearest negative example and the farthest positive one. The second learner is a decision stump inducer, i.e., it induces axis-parallel half planes in the example space. The induced half plane is optimal, as the best attribute and the best value are chosen for the testing attribute (see Fig. 1).

Three main differences exist between the two algorithms:

1. the sphere inducer contains a stochastic step, while the decision stump inducer is totally deterministic;
2. the sphere inducer is positively biased, while the other one is not;
3. the region labelled as positive by the first learner is bounded, the other one extends to infinity (we say that the first hypothesis is "local" whereas the other one is "non-local").

4 Results

Due to space constraints not all the results will be presented: the whole set of tables and pictures can be found in [4]. In the following we will use subscripts to denote which weak learner a given algorithm is coupled with. Furthermore we will use the following abbreviations: Boost for Boosting, GCA for the greedy covering strategy, Bag for Bagging, SP for the sphere inducer and DS for the decision stump inducer.

As already stated in Sect. 2, Bagging is exactly a non-biased Monte Carlo algorithm. Actually the performances shown by bagging when used along with

[4] Noisy datasets have been named joining the name of the original dataset and the name given to the particular kind of noise used for perturbing it, for instance in the following we will denote with *asia.labels* the dataset obtained perturbing *asia* with the kind of noise we explained first in this section.

[5] Here biased is used with the meaning explained in section 2.1

the sphere inducer matches exactly the ones of the hypothesis that labels each point as negative (we will refer to it as $\mathsf{False(x)}$ or as the default hypothesis). At first this appears to be a bit counter-intuitive, since one may expect that such an algorithm should amplify the advantage of the weak learner as predicted by the theory. The key point is that Bagging is not suited at all to be used with local biased hypotheses, since this kind of hypotheses violates the independence assumption over the weak algorithm. The learned hypotheses will agree over all the negative instances, but for any dataset in which it is unlikely to learn very large spheres, very few of them will agree over positive examples. Therefore the default hypothesis will be learned or, in other words, nothing will be learned at all.

When coupled with decision stumps something similar happens but due to different reasons. The deterministic nature of decision stumps violates once again Monte Carlo assumption of independence between hypotheses. While this is not catastrophic in AdaBoost, which heavily reweights the training set, the small differences in the training sets introduced by Bagging are not sufficient to force the deterministic learner to induce very different hypotheses. This, again, does not allow Bagging to amplify the advantage of the weak learner.

The above considerations derive from Fig. 2 and 3; as we can see, $\mathsf{Bag_{SP}}$ quickly converges to the default hypothesis; indeed, the training and test error increase from the initial error made by the weak learner to the error made by $\mathsf{False(x)}$. On the other hand, $\mathsf{Bag_{DS}}$ does nothing but returning the weak hypothesis.

$\mathsf{Boost_{SP}}$ seems to work quite well; it monotonically amplifies the performances of the weak learner and there is no sign of overfitting in any of the experiments. The error rate drops exponentially fast on most datasets (note the log scale on the x-axis). It seems interesting that this does not hold for datasets with noise on labels. The experiments with the other datasets seem to confirm this behavior.

$\mathsf{Boost_{DS}}$ learns quite bad hypotheses. It seems that even if the hypothesis spaces searched by the decision stumps inducer and by the sphere inducer share the same VC-dimension (namely $\mathsf{VCdim(DS)} = \mathsf{VCdim(SP)} = 3$), the space of their linear combinations is quite different. In particular, it seems that combining spheres leads to much more expressive hypotheses than combining decision stumps. The graphs show that $\mathsf{Boost_{DS}}$ slightly improves the performances of the weak learner, converging quickly to a final hypothesis. Plots of ϵ_t (not included here due to space constraints) clearly shows that AdaBoost takes few rounds to drive the error of the stump inducer to 0.5: after this point no further learning occurs.

Most astonishing is that, against our initial guess, GCA outperforms AdaBoost (and all the other algorithms) on almost all datasets. It reaches zero error on the training data in few rounds, and even when it is forced to learn for a long period (while its stopping criterion says that it should stop as soon as the training data are covered), it shows a really small overfitting.

Interestingly enough, even though AdaBoost does not overfit at all, it never reaches GCA's performances.

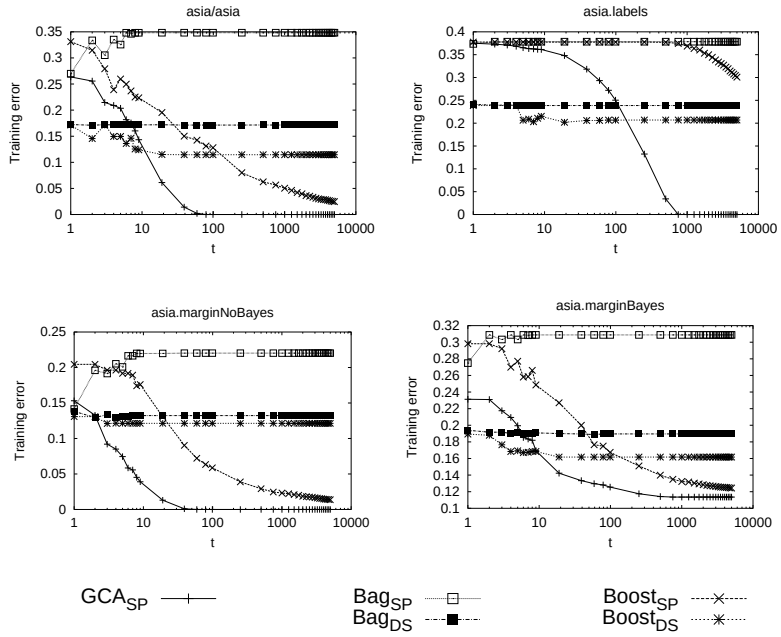

Fig. 2. Training error of all the classifiers over *asia* dataset family

As a last observation we may note that even if performances of $\mathsf{Boost}_{\mathsf{DS}}$ are worse than the others, it seems to be quite invariant with respect to different kind of noises. However, definite conclusions cannot be drawn without further investigations.

5 Conclusion

Relating Ensemble Learning to Monte Carlo theory can shed light over the reasons of the observed performances of ensemble learners. After introducing basic Monte Carlo theory and terminology, the article takes an experimental approach as a way to investigate where the predictions given by the theory break down. On the contrary, experimental results seems to suggest what immediately follows from the theory, and the theory allows us to explain both the unexpected good results of the GCA and the poor performances of Bagging.

References

1. Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105, 1999.
2. G. Brassard and P. Bratley. Algorithmics: theory and practice, 1988.

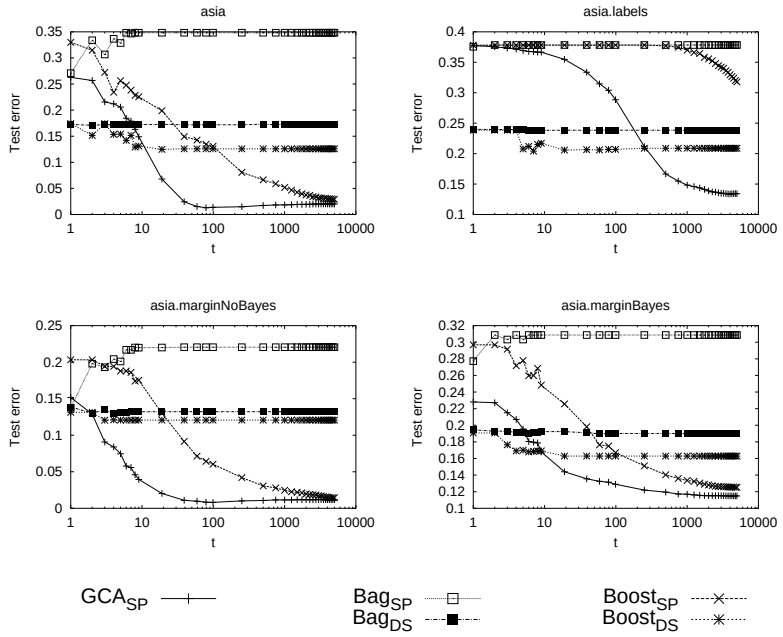

Fig. 3. Test error of all the classifiers over *asia* dataset family

3. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
4. R. Esposito and L. Saitta. "Is a Greedy Covering Strategy an Extreme Boosting?": table of experiments. `http://www.di.unito.it/~esposito/mcandboost`.
5. Y. Freund and R. Schapire. Game theory, on-line prediction and boosting. In *Proceedings, 9th Annual Conference on Computational Learning Theory*, pages 325–332, 1996.
6. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.
7. J. Friedman, J. Stochastic, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Department of Statistics, Stanford University, 1999.
8. J. R. Quinlan. Bagging, boosting and c4.5. In *Proc. AAAI'96 National Conference on Artificial Intelligence*, 1996.
9. R. Schapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistic*, 26(5):1651–1686, 1998.
10. Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197, 1990.