Boosting as a Monte Carlo Algorithm

Roberto Esposito, Lorenza Saitta

Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, Torino, Italy
esposito@di.unito.it
DISTA, Università del Piemonte Orientale "Amedeo Avogadro",
Corso Borsalino 54, Alessandria, Italy
saitta@infn.unipmn.it

Abstract. A new view of majority voting as a Monte Carlo stochastic
algorithm is presented in this paper. The 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. Even though one may expect that the greedy
strategy is very much prone to overfitting, extensive experimental
results do not support this guess. The greedy 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, if not lower,
than that exhibited by Adaboost.

1 Introduction

Majority voting classification algorithms, such as boosting [Schapire,
1990] or bagging [Breiman, 1996] are very popular nowadays because of the
superior performances shown experimentally on a number of data sets (see, for example,
[Bauer & Kohavi, 1999; Quinlan, 1996]). 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, usually, to increase,
even when the number of voting classifiers ranges in the thousands. A rather
convincing argument to explain this behaviour is that boosting increases the number
of learning examples with a large classification margin [Schapire et al., 1998].

Given a class of hypotheses, and the empirical classification errors on a set of
examples of the single classifiers belonging to the class, it is possible to upper bound
the training error of the composite classification rule in terms of them, and to upper
bound, in probability, the generalization error, as well. However, these last bounds are
too loose to be useful in practice, even though they are consistent with the qualitative
behaviour of the classification error observed experimentally. Moreover, it is possible
to estimate the minimum number of iterations necessary to achieve a desired error on
the learning set, with high probability. 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 and game theory [Freund & Schapire, 1996b], have been established. In this paper we offer a new perspective, relating majority voting with Monte Carlo stochastic algorithms [Brassard & Bratley, 1988]. 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 [Brassard & Bratley, 1988, p. 265].

In addition, a subclass of Monte Carlo algorithms 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, the answer is yes, and 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, machine learning experience tells us that they are not. 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 and also some "natural" ones.

2 Monte Carlo Algorithms

Given a class \( \Pi \) 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 [Brassard & Bratley, 1988]. In order for an algorithm to be Monte Carlo, any problem instance must have the same probability of being incorrect \(^1\). More precisely, let \( p \) be a real number such that \( 1/2 < p < 1 \). A Monte Carlo algorithm is \( p \)-correct if the probability that it returns a correct answer is at least \( p \) on any problem instance. The difference \( (p - 1/2) \) 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.

The probability of success of a consistent Monte Carlo algorithm can be increased by running the algorithm several time on the same instance, and choosing the most frequent answer\(^2\). More precisely, let \( \epsilon \) and \( \eta \) be two positive real numbers, such that \( \epsilon \)

\(^{1}\) 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 MC and taking the most frequent answer leads to a 0.84-correct algorithm, whereas the resulting algorithm is only 0.71-correct, should MC be not consistent.
+ η < 1/2. Let MC(x) be a consistent and (1/2 + ε)-correct Monte Carlo algorithm. If we define

\[ n(ε) = \frac{2}{\lg_2(1 - 4 \varepsilon^2)} \]  

(1)

it is sufficient to call MC at least

\[ T = \left\lceil \frac{1}{\varepsilon} \right\rceil \lg_2(1/\eta) \]  

(2)

times on x, and to returns the most frequent answer\(^3\), to obtain an algorithm that is still consistent and also (1-η)-correct. We have *amplified* the advantage of MC(x). A more accurate bound than (2) can be obtained by running a consistent and (1/2 + ε)-correct algorithm a number \( T = (2m - 1) \) of times; the resulting algorithm is (1 - η)-correct, where:

\[ η = 1/2 - ε \sum_{i=0}^{m-1} \text{Bin}(2i, i) \left(\frac{1}{4} - ε^2\right)^i \cdot \frac{(1 - 4 ε^2)^m}{4ε√π m} \]  

(3)

### 2.1 Biased Monte Carlo Algorithms

Let us consider 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 \( p \leq 1/2 \), provided that \( p > 0 \). More formally:

**Definition 1** – Let \( \Pi \) be a class of problem and let \( s_0 \) be a possible output of a Monte Carlo algorithm MC(x). MC(x) is \( s_0 \)-*biased* if there exists a subset \( X \) of \( \Pi \) such that:

a. MC(x) is always correct on instance \( x \) whenever \( x \notin X \),

b. The correct solution to any \( x \in X \) is \( s_0 \), but MC(x) may not always return the correct answer on these instances.

It is easy to prove [Brassard & Bratley, 1988, p. 266] the following:

**Theorem 1** (Brassard & 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.

---

\(^3\) Taking the most frequent answer corresponds to an unweighted majority voting. It is possible to modify the decision procedure so that weighted voting results. However, this issue is not central to the paper, and will not be considered further.
Then, in order to achieve a correctness level of \((1 - \eta)\), it is sufficient to run the algorithm at least a number of times:

\[
T = \frac{\log_2 \eta}{\log_2 (1 - \eta)}
\]

(4)

Table 1 reports a comparison among the numbers of repetitions for unbiased and biased algorithms, according to (2), (3) and (4).

<table>
<thead>
<tr>
<th>(1 - \varepsilon)</th>
<th>(1 - \eta)</th>
<th>(T) Bound (2)</th>
<th>(T) Bound (3)</th>
<th>(T) Bound (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.55</td>
<td>0.95</td>
<td>596</td>
<td>269</td>
<td>4</td>
</tr>
<tr>
<td>0.55</td>
<td>0.99</td>
<td>918</td>
<td>387</td>
<td>6</td>
</tr>
<tr>
<td>0.10</td>
<td>0.99</td>
<td>—</td>
<td>—</td>
<td>11</td>
</tr>
</tbody>
</table>

From Table 1 biased algorithms appear to be much more effective and highly desirable.

3 Relations between Majority Voting and Monte Carlo Algorithms

Let us consider now a learning context in which a weak learner \(A\) acquires decision rules \(h(x) : X \rightarrow Y\), belonging to a set \(H\). \(X\) is a set of instances and \(Y = \{+1, -1\}\) is a binary set of classes. Let us call positive the instances labelled +1. Let \(E\), be a subset of \(X\), such that \(|E| = N\), and \(E = POS \cup NEG\). Let \(MC(x)\) be a Monte Carlo algorithm that takes as input a random instance \(x\) and outputs a class label \(h_t(x) \in Y\). The instance belongs to \(E\), but \(MC(x)\) does not know. \(MC(x)\) can be defined as in Figure 1.

\(MC(x)\) is a Monte Carlo algorithm, because the probability \(\varepsilon_t\) of misclassifying \(x\) (i.e., the generalization error of \(h_t(x)\)) is uniform over all instances \(x\). Moreover, \(MC\) is consistent, because the correct answer (class label) is unique.

If we run \(MC(x)\) \(T\) times on instance \(x\) and take as answer the most frequent one, it is immediate to see that \(MC(x)\) corresponds to a majority voting classification algorithm. Different combination schemes, such as boosting or bagging, correspond to different ways of choosing the set at each run. Then, for the error \(\varepsilon\) of the combined decision, formulas (2) and (3) hold. As we may notice, formula (2) is the same as the one derived by Shapire [1990], whereas formula (3) is an improvement thereupon, which is more close to experimental results [Quinlan, 1996; Bauer & Kohavi, 1999].

If we now consider biased Monte Carlo algorithms, we may wonder to 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 of the AQ-type, with the set of positive examples as the S 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 MC(x), 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 look at Adaboost, this algorithm partially satisfies the biasedness requirement. In fact, even though successive runs are not independent, the weighting process simulates the elimination of already classified examples: actually, correctly classified examples are not really eliminated, but only considered less important than not yet correctly classified ones. The process of deleting covered examples is a limiting process of this weighting procedure.

The idea of obtaining much smaller classifiers 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 recognition rate (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 different behaviour of Adaboost and GCA with respect to generalization is reflected in the evolution of margin distributions, if it is true that large margins are the reason underlying Adaboost's robustness.

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

We now describe the experimental setting used to test Adaboost and GCA algorithms. The following three subsections will describe the learning task, the two tested algorithms and the datasets used for experiments.

4.1 Weak Learner

Both Adaboost and the GCA use a very simple weak learner, called MCK, that was designed on purpose. This learner induces spherical stumps, i.e. the basic classifiers \( h_i(x) \) are spheres in the example space: if \( x \) falls inside the sphere, then \( h_i(x) = 1 \), otherwise \( h_i(x) = -1 \). The characteristic of MCK is that individual classifiers are local, and hence they differ from decision stumps in the same way as Radial Basis Function neural net differ from a multilayer perceptron. Adaboost's implementation is the same as in [Freund & Schapire, 1996a].

In order to explore the behaviour of the generalization error for high redundancy level, Adaboost has been configured to stop after 20,000 iterations on "difficult" datasets, after "10,000" iterations on the other artificial ones, and after 6,000 iterations on the "Splice Junction" dataset. For MCK, we have used two strategies: first, the iteration of MCK stops when a zero error is reached, which happens when no positive example is left uncovered. But we have also continued to add classifiers, as in Adaboost, in the hope to show that this added redundancy would let the test error increase. Then, in this case, the GCA has been iterated for up to 1,000 runs.

4.2 Datasets

We created nine artificial datasets. Eight of them contain examples of concepts that lie in a two-dimensional Euclidean space. Figure 2 shows these concepts. In addition, a 5-dimensional, sphere-like concept (D8.Hyperspheres, \( N = 5000 \)), and the "Splice Junction" dataset (D9.Splice, \( N = 3190 \)) from Irvine repository have been considered. \( N \) is the total number of examples in the dataset.

4.3 Experimental Results

Even though the same analysis has been performed on all the datasets, detailed results are described only for D4.Asia, because they all showed similar patterns of behaviour. In particular, the results have been summarized into Figure 3.

The graphs reported in Figure 3 are typical of the two dimensional cases. An interesting feature of these graphs is the fact that even though Adaboost may be better in the long run, for small number of trials the simpler GCA strategy is often better with respect to not only the training error (which reaches zero in few runs), but also with respect on the test set.
5 Conclusions

Relating majority voting procedures with similar approaches can shed light on the reasons and conditions under which it may work. According to the experimentation provided, there is still a need for investigation on at least two issues: the influence of the very methods of combination, and the type of example distribution modification (reweighting). However, further experiments are necessary to draw more precise conclusions.
Fig. 3. Results for the dataset D4.Asia. Empirical error on the training and test sets for Adaboost and GCA.

References


