Chapter 14 in "Online Algorithms: the state of the art", Fiat and Woeginger eds., LNCS #1442, 1998.

On-Line Algorithms in Machine Learning

Avrim Blum

Carnegie Mellon University, Pittsburgh PA 15213. Email: avrim@cs.cmu.edu

Abstract. The areas of On-Line Algorithms and Machine Learning are both concerned with problems of making decisions about the present based only on knowledge of the past. Although these areas differ in terms of their emphasis and the problems typically studied, there are a collection of results in Computational Learning Theory that fit nicely into the "on-line algorithms" framework. This survey article discusses some of the results, models, and open problems from Computational Learning Theory that seem particularly interesting from the point of view of on-line algorithms.
The emphasis in this article is on describing some of the simpler, more intuitive results, whose proofs can be given in their entirity. Pointers to the literature are given for more sophisticated versions of these algorithms.

1 Introduction

The areas of On-Line Algorithms and Machine Learning are both concerned with problems of making decisions from limited information. Although they differ in terms of their emphasis and the problems typically studied, there are a collection of results in Computational Learning Theory that fit nicely into the "on-line algorithms" framework. This survey article discusses some of the results, models, and open problems from Computational Learning Theory that seem particularly interesting from the point of view of on-line algorithms. This article is not meant to be comprehensive. Its goal is to give the reader a sense of some of the interesting ideas and problems in this area that have an "on-line algorithms" feel to them.

We begin by describing the problem of "predicting from expert advice," which has been studied extensively in the theoretical machine learning literature. We present some of the algorithms that have been developed and that achieve quite tight bounds in terms of a competitive ratio type of measure. Next we broaden our discussion to consider several standard models of on-line learning from examples, and examine some of the key issues involved. We describe several interesting algorithms for on-line learning, including the Winnow algorithm and an algorithm for learning decision lists, and discuss issues such as attribute-efficient learning and the infinite attribute model, and learning target functions that change over time. Finally, we end with a list of important open problems in the area and a discussion of how ideas from Computational Learning Theory and On-Line Algorithms might be fruitfully combined.

To aid in the flow of the text, most of the references and discussions of history are placed in special "history" subsections within the article.

Chapter 14 in "Online Algorithms: the state of the art", Fiat and Woeginger eds., LNCS #1442, 1998.

2 Predicting from Expert Advice

We begin with a simple, intuitive problem. A learning algorithm is given the task each day of predicting whether or not it will rain that day. In order to make this prediction, the algorithm is given as input the advice of n "experts". Each day, each expert predicts yes or no, and then the learning algorithm must use this information in order to make its own prediction (the algorithm is given no other input besides the yes/no bits produced by the experts). After making its prediction, the algorithm is then told whether or not, in fact, it rained that day. Suppose we make no assumptions about the quality or independence of the experts, so we cannot hope to achieve any absolute level of quality in our predictions. In that case, a natural goal instead is to perform nearly as well as the best expert so far: that is, to guarantee that at any time, our algorithm has not performed much worse than whichever expert has made the fewest mistakes to date. In the language of competitive analysis, this is the goal of being competitive with respect to the best single expert.

We will call the sequence of events in which the algorithm (1) receives the predictions of the experts, (2) makes its own prediction, and then (3) is told the correct answer, a *trial*. For most of this discussion we will assume that predictions belong to the set $\{0, 1\}$, though we will later consider more general sorts of predictions (e.g., many-valued and real-valued).

2.1 A Simple Algorithm

The problem described above is a basic version of the problem of "predicting from expert advice" (extensions, such as when predictions are probabilities, or when they are more general sorts of suggestions, are described in Section 2.3 below). We now describe a simple algorithm called the Weighted Majority algorithm. This algorithm maintains a list of weights $w_1, \ldots w_n$, one for each expert, and predicts based on a weighted majority vote of the expert opinions.

The Weighted Majority Algorithm (simple version)

1. Initialize the weights $w_1, \ldots, w_n$ of all the experts to 1.
2. Given a set of predictions $\{x_1, \ldots, x_n\}$ by the experts, output the prediction with the highest total weight. That is, output 1 if
$$\sum_{i:x_i=1} w_i \geq \sum_{i:x_i=0} w_i$$
and output 0 otherwise.
3. When the correct answer ℓ is received, penalize each mistaken expert by multiplying its weight by 1/2. That is, if $x_i \neq \ell$, then $w_i \leftarrow w_i/2$; if $x_i = \ell$ then w_i is not modified.
Goto 2.

Theorem 1. *The number of mistakes M made by the Weighted Majority algorithm described above is never more than $2.41(m + \lg n)$, where m is the number of mistakes made by the best expert so far.*

Proof. Let W denote the total weight of all the experts, so initially $W = n$. If the algorithm makes a mistake, this means that at least half of the total weight of experts predicted incorrectly, and therefore in Step 3, the total weight is reduced by at least a factor of 1/4. Thus, if the algorithm makes M mistakes, we have:

$$W \leq n(3/4)^M. \qquad (1)$$

On the other hand, if the best expert has made m mistakes, then its weight is $1/2^m$ and so clearly:

$$W \geq 1/2^m. \qquad (2)$$

Combining (1) and (2) yields $1/2^m \leq n(3/4)^M$ and therefore:

$$\begin{aligned} M &\leq \tfrac{1}{\lg(4/3)}(m + \lg n) \\ &\leq 2.41(m + \lg n)\Box \end{aligned}$$

2.2 A Better Algorithm

We can achieve a better bound than that described above by modifying the algorithm in two ways. The first is by randomizing. Instead of predicting the outcome with the highest total weight, we instead view the weights as probabilities, and predict each outcome with probability proportional to its weight. The second change is to replace "multiply by 1/2" with "multiply by β" for a value β to be determined later.

Intuitively, the advantage of the randomized approach is that it dilutes the worst case. Previously, the worst case was that slightly more than half of the total weight predicted incorrectly, causing the algorithm to make a mistake and yet only reduce the total weight by 1/4. Now, there is roughly a 50/50 chance that the algorithm will predict correctly in this case, and more generally, the probability that the algorithm makes a mistake is tied to the amount that the weight is reduced.

A second advantage of the randomized approach is that it can be viewed as selecting an expert with probability proportional to its weight. Therefore, the algorithm can be naturally applied when predictions are "strategies" or other sorts of things that cannot easily be combined together. Moreover, if the "experts" are programs to be run or functions to be evaluated, then this view speeds up prediction since only one expert needs to be examined in order to produce the algorithm's prediction (although all experts must be examined in order to make an update of the weights). We now formally describe the algorithm and its analysis.

The Weighted Majority Algorithm (randomized version)

1. Initialize the weights $w_1, \ldots, w_n$ of all the experts to 1.
2. Given a set of predictions $\{x_1, \ldots, x_n\}$ by the experts, output x_i with probability w_i/W, where $W = \sum_i w_i$.

3. When the correct answer ℓ is received, penalize each mistaken expert by multiplying its weight by β.
   Goto 2.

Theorem 2. *On any sequence of trials, the expected number of mistakes M made by the Randomized Weighted Majority algorithm described above satisfies:*

$$M \leq \frac{m \ln(1/\beta) + \ln n}{1 - \beta}$$

where m is the number of mistakes made by the best expert so far.

For instance, for $\beta = 1/2$, we get an expected number of mistakes less than $1.39m + 2\ln n$, and for $\beta = 3/4$ we get an expected number of mistakes less than $1.15m + 4\ln n$. That is, by adjusting β, we can make the "competitive ratio" of the algorithm as close to 1 as desired, at the expense of an increase in the additive constant. In fact, by adjusting β dynamically using a typical "guess and double" approach, one can achieve the following:

Corollary 3. *On any sequence of trials, the expected number of mistakes M made by a modified version of the Randomized Weighted Majority algorithm described above satisfies:*

$$M \leq m + \ln n + O(\sqrt{m \ln n})$$

where m is the number of mistakes made by the best expert so far.

Proof of Theorem 2. Define F_i to be the fraction of the total weight on the *wrong* answers at the i^{th} trial. Say we have seen t examples. Let M be our expected number of mistakes so far, so $M = \sum_{i=1}^{t} F_i$.

On the i^{th} example, the total weight changes according to:

$$W \leftarrow W(1 - (1 - \beta)F_i)$$

since we multiply the weights of experts that made a mistake by β and there is an F_i fraction of the weight on these experts. Hence the final weight is:

$$W = n \prod_{i=1}^{t} (1 - (1 - \beta)F_i)$$

Let m be the number of mistakes of the best expert so far. Again, using the fact that the total weight must be at least as large as the weight on the best expert, we have:

$$n \prod_{i=1}^{t} (1 - (1 - \beta)F_i) \geq \beta^m \tag{3}$$

Chapter 14 in "Online Algorithms: the state of the art", Fiat and Woeginger eds., LNCS #1442, 1998.

Taking the natural log of both sides we get:

$$\begin{aligned}
\ln n + \sum_{i=1}^{t} \ln(1-(1-\beta)F_i) &\geq m \ln \beta \\
-\ln n - \sum_{i=1}^{t} \ln(1-(1-\beta)F_i) &\leq m \ln(1/\beta) \\
-\ln n + (1-\beta)\sum_{i=1}^{t} F_i &\leq m \ln(1/\beta) \\
M &\leq \frac{m \ln(1/\beta) + \ln n}{1-\beta}
\end{aligned}$$

Where we get the third line by noting that $-\ln(1-x) > x$, and the fourth by using $M = \sum_{i=1}^{t} F_i$. □

2.3 History and Extensions

Within the Computational Learning Theory community, the problem of predicting from expert advice was first studied by Littlestone and Warmuth [28], DeSantis, Markowsky and Wegman [15], and Vovk [35]. The algorithms described above as well as Theorems 1 and 2 are from Littlestone and Warmuth [28], and Corollary 3, as well as a number of refinements, are from Cesa-Bianchi et al. [12]. Perhaps one of the key lessons of this work in comparison to work of a more statistical nature is that one can remove all statistical assumptions about the data and still achieve extremely tight bounds (see Freund [18]). This problem and many variations and extensions have been addressed in a number of different communities, under names such as the “sequential compound decision problem” [32] [4], “universal prediction” [16], “universal coding” [33], “universal portfolios” [13], and “prediction of individual sequences”; the notion of the competitiveness is also called the “min-max regret” of an algorithm. A web page uniting some of these communities and with a discussion of this general problem now exists at `http://www-stat.wharton.upenn.edu/Seq96`.

A large variety of extensions to the problem described above have been studied. For example, suppose that each expert provides a real number between 0 and 1 as its prediction (e.g., interpret a real number p as the expert's belief in the probability of rain) and suppose that the algorithm also may produce a real number between 0 and 1. In this case, one must also specify a loss function — what is the penalty for predicting p when the outcome is x? Some common loss functions appropriate to different settings are the absolute loss: $|p-x|$, the square loss: $(p-x)^2$, and the log loss: $-x \ln p - (1-x)\ln(1-p)$. Papers of Vovk [35, 36], Cesa-Bianchi et al. [12, 11], and Foster and Vohra [17] describe optimal algorithms both for these specific loss functions and for a wide variety of general loss functions.

A second extension of this framework is to broaden the class of algorithms against which the algorithm is competitive. For instance, Littlestone, Long, and