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.

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:

$$W \leq n(3/4)^M. \tag{1}$$

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

$$W \geq 1/2^m. \tag{2}$$

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

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

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:

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

Taking the natural log of both sides we get:

$$\ln n + \sum_{i=1}^{t} \ln(1-(1-\beta)F_i) \geq m \ln \beta$$

$$-\ln n - \sum_{i=1}^{t} \ln(1-(1-\beta)F_i) \leq m \ln(1/\beta)$$

$$-\ln n + (1-\beta)\sum_{i=1}^{t} F_i \leq m \ln(1/\beta)$$

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

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

Warmuth [27] show that modifications of the algorithms described above are constant-competitive with respect to the best linear combination of experts, when the squared loss measure is used. Merhav and Feder [29] show that one can be competitive with respect to the best off-line strategy that can be implemented by a finite state machine.

Another variation on this problem is to remove all semantics associated with specific predictions by the experts and to simply talk about losses. In this variation, the learning algorithm is required in each iteration to select an expert to "go with". For instance, suppose we are playing a 2-player matrix game as the row player. Each row can be viewed as an expert. To play the game we probabilistically select some expert (row) to use, and then, after the game is done, we find out our loss and that of each expert. If we are playing repeatedly against some adversary, we then would get another opportunity to probabilistically select an expert to use and so forth. Freund and Schapire show that extensions of the randomized Weighted Majority Algorithm discussed above can be made to fit nicely into this scenario [19] (see also the classic work of Blackwell [4]). Another scenario fitting this framework would be a case where each expert is a page-replacement algorithm, and an operating system needs to decide which algorithm to use. Periodically the operating system computes losses for the various algorithms that it could have used and based on this information decides which algorithm to use next.

Ordentlich and Cover [14] [30] describe strategies related to the randomized Weighted Majority algorithm for a problem of on-line portfolio selection. They give an on-line algorithm that is optimally competitive against the best "constant-rebalanced portfolio" (CRP). Their algorithm can be viewed as creating one expert for every CRP and then allocating its resources among them. This setting has the nice property that the market automatically adjusts the weights, so the algorithm itself just initially divides its funds equally among all infinitely-many CRPs and then lets it sit. A simple analysis of their algorithm with extensions to transaction costs is given in [10].

3 On-Line Learning from Examples

The previous section considered the problem of "learning from expert advice". We now broaden our focus to consider the more general scenario of on-line learning from examples. In this setting there is an example space $\mathcal{X}$, typically $\{0,1\}^n$. Learning proceeds as a sequence of trials. In each trial, an example $x \in \mathcal{X}$ is presented to the learning algorithm. The algorithm then predicts either 1 or 0 (whether the example is positive or negative) and finally the algorithm is told the true label $\ell \in \{0,1\}$. The algorithm is penalized for each mistake made; i.e., whenever its prediction differs from ℓ. Our goal is to make as few mistakes as possible. Typically, the presentation of examples will be assumed to be under the control of an adversary. This setting is also broadly called the Mistake Bound learning model.

The scenario described so far is not too different from the standard framework of On-Line Algorithms: we are given an on-line sequence of tasks and we want our penalty to be not too much larger than that of the best off-line algorithm. However, for the task of predicting labels, the "best we could do if there was no hidden information" would be to make zero mistakes, whereas no on-line algorithm could do better than make mistakes half the time if the labels were chosen randomly. Thus, some further restriction on the problem is necessary in order to make nontrivial statements about algorithms. Several natural restrictions are: (1) to restrict the labels to being determined by some "reasonable" function of the examples, (2) to restrict the off-line algorithms being compared against to some "reasonable" class of algorithms, and (3) to restrict the adversary to having some sort of randomness in its behavior. Each of these restrictions corresponds to a standard model studied in Computational Learning Theory, and we describe these in more detail below.

To describe these models, we need the notion of a *concept class*. A *concept class* $\mathcal{C}$ is simply a set of Boolean functions over the domain $\mathcal{X}$ (each Boolean function is sometimes called a *concept*), along with an associated representation of these functions. For instance, the class of *disjunctions* over $\{0,1\}^n$ is the class of all functions that can be described as a disjunction over the variables $\{x_1, \ldots, x_n\}$. The class of *DNF formulas* contains all Boolean functions, each with a description length equal to the size of its minimum DNF formula representation. In the discussion below, we will use n to denote the description length of the examples, and $size(c)$ to denote the description length of some concept $c \in \mathcal{C}$.

We now describe three standard learning problems.

Learning a concept class $\mathcal{C}$ (in the Mistake Bound model): In this setting, we assume that the labels attached to examples are generated by some unknown target concept $c \in \mathcal{C}$. That is, there is some hidden concept c belonging to the class $\mathcal{C}$, and in each trial, the label ℓ given to example x is equal to $c(x)$. The goal of the learning algorithm is to make as few mistakes as possible, assuming that both the choice of target concept and the choice of examples are under the control of an adversary. Specifically, if an algorithm has the property that for any target concept $c \in \mathcal{C}$ it makes at most $poly(n, size(c))$ mistakes on any sequence of examples, and its running time per trial is $poly(n, size(c))$ as well, then we say that the algorithm *learns class $\mathcal{C}$ in the mistake bound model.* If, furthermore, the number of mistakes made is only $poly(size(c)) \cdot polylog(n)$ — that is, if the algorithm is robust to the presence of many additional irrelevant variables — then the algorithm is also said to be *attribute efficient.*

Algorithms have been developed for learning a variety of concept classes in the Mistake Bound model, such as disjunctions, k-DNF formulas, decision lists, and linear threshold functions. Below we will describe a very elegant and practical algorithm called the Winnow Algorithm, that learns disjunctions in the mistake bound model and makes only $O(r \log n)$ mistakes, where r is the number of variables that actually appear in the target disjunction. Thus, Winnow is attribute-efficient. This algorithm also has the property that it can be used to track a target concept that changes over time, and we will describe a sense in which the algorithm can be viewed as being $O(\log n)$ competitive for this task. We will also discuss a few general results on attribute-efficient learning and a model known as the infinite-attribute model.

Agnostic Learning / Being Competitive with the class $\mathcal{C}$: In this model, we make no assumptions about the existence of any relationship between the labels and the examples. Instead, we simply set our goal to be that of performing nearly as well as the best concept in $\mathcal{C}$. This is sometimes called the *agnostic* learning model and can be viewed as the problem of learning a concept class in the presence of adversarial noise. In this article, to use the terminology from On-Line Algorithms, we will call this the goal of being *competitive with respect to the best concept in* $\mathcal{C}$. Specifically, let us say that an algorithm is α*-competitive with respect to* $\mathcal{C}$ if there exists a polynomial p such that for any sequence of examples and any concept $c \in \mathcal{C}$, the number of mistakes made by the algorithm is at most $\alpha m_c + p(n, size(c))$, where m_c is the number of mistakes made by concept c. The algorithm should have running time per trial polynomial in n and $size(c)$ where c is the best concept in $\mathcal{C}$ on the data seen so far.

If we consider the class $\mathcal{C}$ of single-variable concepts over $\{0,1\}^n$ (that is, $\mathcal{C}$ consists of n concepts $\{c_1, \ldots, c_n\}$ where $c_i(x) = x_i$), then this is really the same as the problem of "learning from expert advice" discussed in Section 2 (just think of the example as the list of predictions of the experts), and the algorithms of Section 2 show that for all $\epsilon > 0$, one can achieve $(1+\epsilon)$-competitiveness with respect to the best concept in this class.

It is worth noting that if we do not care about computational complexity (i.e., we remove the restriction that the algorithm run in polynomial time per trial) then we can achieve $(1+\epsilon)$-competitiveness for any concept class $\mathcal{C}$ over $\{0,1\}^n$. Specifically, we have the following theorem.

Theorem 4. *For any concept class $\mathcal{C}$ over $\{0,1\}^n$ and any $\epsilon > 0$ there is a non-polynomial time algorithm that on any sequence of examples, for all $c \in \mathcal{C}$, makes at most $(1+\epsilon)m_c + O(size(c))$ mistakes.*

Proof. We simply associate one "expert" with each concept $c \in \mathcal{C}$, and run the Randomized Weighted Majority algorithm described in Section 2 with the modification that the initial weight given to a concept c is $2^{-2size(c)}$. This assignment of initial weights means that initially, the total weight W is at most 1. Therefore, inequality (3) is replaced by the statement that for any concept $c \in \mathcal{C}$, after t trials we have:

$$\prod_{i=1}^{t}(1-(1-\beta)F_i) \geq \beta^{m_c} 2^{-2size(c)}$$

where m_c is the number of mistakes make by c. Solving this inequality as in the proof of Theorem 2 yields the guarantee that for any $c \in \mathcal{C}$, the total number of mistakes M made by the algorithm satisfies:

$$m \leq \frac{m_c \ln(1/\beta) + 2size(c)}{1-\beta}.$$

On the other hand, this algorithm clearly does not run in polynomial time for most interesting concept classes since it requires enumerating all of the possible concepts $c \in \mathcal{C}$.[1] □

A second fact worth noting is that in many cases there are NP-hardness results if we require the learning algorithm to use representations from the class $\mathcal{C}$. For instance, it is NP-hard, given a set S of labeled examples, to find the disjunction that minimizes the number of disagreements with this sample. However, this does not necessarily imply that it is NP-hard to achieve a competitive ratio approaching 1 for learning with respect to the class of disjunctions, since the hypothesis used by the learning algorithm need not be a disjunction.

As mentioned in the Open Problems section, it is unknown whether it is possible to achieve a good competitive ratio with respect to the class of disjunctions with a polynomial time algorithm.

Learning $\mathcal{C}$ in the presence of random noise: This model lies somewhat in between the two models discussed so far. In this model, we assume that there is a target concept $c \in \mathcal{C}$ just like in the standard Mistake Bound model. However, after each example is presented to the learning algorithm, the adversary flips a coin and with probability $\eta < 1/2$, gives the algorithm the wrong label. That is, for each example x, the correct label $c(x)$ is seen with probability $1-\eta$, and the incorrect label $1-c(x)$ is seen with probability η, independently for each example. Usually, this model is only considered for the case in which the adversary itself is restricted to selecting examples according to some fixed (but unknown) distribution D over the instance space. We will not elaborate further on this model in this article, since the results here have less of an "on-line algorithms" feel to them, except to say that a very nice theory has been developed for learning in this setting, with some intriguing open problems, including one we list in Section 4.

One final point worth mentioning is that there are a collection of simple reductions between many standard concept classes. For instance, if one has an algorithm to learn the class of monotone disjunctions (functions such as $x_1 \vee x_5 \vee x_9$), then one can also learn non-monotone disjunctions (like $\overline{x}_1 \vee x_5$), conjunctions, k-CNF formulas for constant k, and k-DNF formulas for constant k, by just performing a transformation on the input space. Thus, if several classes are related in this way, we need only discuss the simplest one.

[1] In the PAC learning setting, there is a similar but simpler fact that one can learn any concept class in the presense of malicious noise by simply finding the concept in $\mathcal{C}$ that has the fewest disagreements on the sample.

3.1 Some Simple Algorithms

As an example of learning a class in the Mistake Bound model, consider the following simple algorithm for learning monotone disjunctions. We begin with the hypothesis $h = x_1 \vee x_2 \vee \ldots \vee x_n$. Each time a mistake is made on a negative example x, we simply remove from h all the variables set to 1 by x. Notice that we only remove variables that are guaranteed to not be in the target function, so we never make a mistake on a positive example. Since each mistake removes at least one variable from h, this algorithm makes at most n mistakes.

A more powerful concept class is the class of decision lists. A *decision list* is a function of the form: "if ℓ_1 then b_1, else if ℓ_2 then b_2, else if ℓ_3 then b_3, ..., else b_m," where each ℓ_i is a literal (either a variable or its negation) and each $b_i \in \{0, 1\}$. For instance, one possible decision list is the rule: "if $\overline{x}_1$ then positive, else if x_5 then negative, else positive." Decision lists are a natural representation language in many settings and have also been shown to have a collection of useful theoretical properties.

The following is an algorithm that learns decision lists, making at most $O(rn)$ mistakes if the target function has r relevant variables (and therefore has length $O(r)$). The hypotheses used by the algorithm will be a slight generalization of decision lists in which we allow several "if/then" rules to co-exist at the same level: if several "if" conditions on the same level are satisfied, we just arbitrarily choose one to follow.

1. Initialize h to the one-level list, whose level contains all $4n + 2$ possible "if/then" rules (this includes the two possible ending rules).
2. Given an example x, look at the first level in h that contains a rule whose "if" condition is satisfied by x. Use that rule for prediction (if there are several choices, choose one arbitrarily).
3. If the prediction is mistaken, move the rule that was used down to the next level.
4. Return to step 2.

This algorithm has the property that at least one "if/then" rule moves one level lower in h on every mistake. Moreover, notice that the very first rule in the target concept c will never be moved, and inductively, the ith rule of c will never move below the ith level of h. Therefore, each "if/then" rule will fall at most L levels, where L is the length of c, and thus the algorithm makes at most $O(nL) = O(nr)$ mistakes.

3.2 The Winnow Algorithm

We now describe a more sophisticated algorithm for learning the class of (monotone) disjunctions than that presented in the previous section. This algorithm, called the Winnow Algorithm, is designed for learning with especially few mistakes when the number of relevant variables r is much less than the total number of variables n. In particular, if the data is consistent with a disjunction of r out of the n variables, then the algorithm will make at most $O(r \log n)$ mistakes. After describing this result, we then show how the Winnow algorithm can be used to achieve in essence an $O(\log n)$ competitive ratio for learning a disjunction that changes over time. We also discuss the behavior of Winnow in the agnostic setting. Variations on this algorithm can be used to learn Boolean threshold functions as well, but we will stick to the problem of learning disjunctions to keep the analysis simpler.

Like the Weighted Majority algorithm discussed earlier, the Winnow algorithm maintains a set of weights, one for each variable.

The Winnow Algorithm (a simple version)

1. Initialize the weights $w_1, \ldots, w_n$ of the variables to 1.
2. Given an example $x = \{x_1, \ldots, x_n\}$, output 1 if
$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \geq n$$
and output 0 otherwise.
3. If the algorithm makes a mistake:
   (a) If the algorithm predicts negative on a positive example, then for each x_i equal to 1, double the value of w_i.
   (b) If the algorithm predicts positive on a negative example, then for each x_i equal to 1, cut the value of w_i in half.
4. Goto 2.

Theorem 5. *The Winnow Algorithm learns the class of disjunctions in the Mistake Bound model, making at most $2 + 3r(1 + \lg n)$ mistakes when the target concept is a disjunction of r variables.*

Proof. Let us first bound the number of mistakes that will be made on positive examples. Any mistake made on a positive example must double at least one of the weights in the target function (the *relevant* weights), and a mistake made on a negative example will *not* halve any of these weights, by definition of a disjunction. Furthermore, each of these weights can be doubled at most $1 + \lg n$ times, since only weights that are less than n can ever be doubled. Therefore, Winnow makes at most $r(1 + \lg n)$ mistakes on positive examples.

Now we bound the number of mistakes made on negative examples. The total weight summed over all the variables is initially n. Each mistake made on a positive example increases the total weight by at most n (since before doubling, we must have had $w_1 x_1 + \ldots w_n x_n < n$). On the other hand, each mistake made on a negative example decreases the total weight by at least $n/2$ (since before halving, we must have had $w_1 x_1 + \ldots + w_n x_n \geq n$). The total weight never drops below zero. Therefore, the number of mistakes made on negative examples is at most twice the number of mistakes made on positive examples, plus 2. That is, $2 + 2r(1 + \lg n)$. Adding this to the bound on the number of mistakes on positive examples yields the theorem. □

How well does Winnow perform when the examples are not necessarily all consistent with some target disjunction? For a given disjunction c, let us define

m_c to be the number of mistakes made by concept c, and let A_c be the number of *attribute errors* in the data with respect to c, which we define as follows. For each example labeled positive but that satisfies no relevant variables of c, we add 1 to A_c; for each example labeled negative but that satisfies k relevant variables of c, we add k to A_c. So, if concept c is a disjunction of r variables, then $m_c \leq A_c \leq rm_c$. It is not hard to show that Winnow has the following behavior for agnostic learning of disjunctions.

Theorem 6. *For any sequence of examples and any disjunction c, the number of mistakes made by Winnow is $O(A_c + r\log n)$, where r is the number of relevant variables for c. Since $A_c \leq rm_c$, this means that Winnow is $O(r)$-competitive with respect to the best disjunction of r variables.*

In fact, by randomizing and tuning the Winnow algorithm to the specific value of r, one can achieve the following stronger statement.

Theorem 7. *Given r, one can tune a randomized Winnow algorithm so that on any sequence of examples and any disjunction c of r variables, the expected number of mistakes made by the algorithm is*

$$A_c + (2 + o(1))\sqrt{A_c r \ln(n/r)}$$

as $A_c/(r \ln \frac{n}{r}) \to \infty$.

These kinds of theorems can be viewed as results in a generalization of the "experts" scenario discussed in Section 2. Specifically, consider an algorithm with access to n "specialists". On every trial, each specialist may choose to make a prediction or it may choose to abstain (unlike the "experts" scenario in which each expert must make a prediction on every trial). That is, we can think of the specialists as only making a prediction when the situation fits their "specialty". Using a proof much like that used to prove Theorem 6, one can show that a version of the Winnow algorithm is constant-competitive with respect to the best *set* of specialists, where we charge a set one unit for every mistake made by a specialist in the set, and one unit whenever all specialists in the set abstain.

3.3 Learning drifting disjunctions

For the problem of learning a static target concept with no noise in the data, there is no real notion of "competitiveness". The algorithm just makes some fixed upper bounded number of mistakes. However, a natural variation on this scenario, which is also relevant to practice, is to imagine that the target function is not static and instead changes with time. For instance, for the case of learning a disjunction, we might imagine that from time to time, variables are added to or removed from the target function. In this case, a natural measure of "adversary cost" is the number of additions and deletions made to the target function, and the obvious goal is to make a number of mistakes that is not too much larger than the adversary's cost.

Specifically, consider the following game played against an adversary. There are n variables and a target concept that initially is the disjunction of zero of them (it says everything is negative). Then, each round of the game proceeds as follows.

Adversary's turn: The adversary may change the target concept by adding or removing some variables from the target disjunction. The adversary pays a cost of 1 for each variable added. (Since the number of variables removed over time is bounded by the number added over time, we may say that removing variables is free.) The adversary then presents an example to the learning algorithm.

Learner's turn: The learning algorithm makes a prediction on the example given, and then is told the correct answer (according to the current target concept). The algorithm is charged a cost of 1 if it made a mistake.

Consider the variation on the Winnow algorithm that never allows any weight to decrease below 1/2; that is, when a mistake is made on a negative example, only weights of value 1 or more are cut in half. Surprisingly, this Winnow variant guarantees that its cost is at most $O(\log n)$ times the adversary cost. So in a sense it is $O(\log n)$-competitive for this problem. Note that Theorem 5 can be viewed as a special case of this in which in its first move, the adversary adds r variables to the target function and then makes no changes from then on.

Theorem 8. *The Winnow variant described above, on any sequence of examples, makes at most $O(c_A \log n)$ mistakes, where c_A is the adversary's total cost so far.*

Proof. Consider the total weight on all the variables. The total weight is initially n. Each mistake on a positive example increases the total weight by at most n and each mistake on a negative example decreases the total weight by at least $n/4$ (because $\sum w_i x_i \geq n$ and at most $n/2$ of this sum can come from weights equal to $1/2$, so at least $n/2$ of the sum gets cut in half). Therefore, the number of mistakes on negative examples is bounded by $4(1 + M_p)$ where M_p is the number of mistakes made on positive examples. So, we only need to bound the number of mistakes on positives.

Let R denote the set of variables in the current target function (i.e., the currently relevant variables), and let $r = |R|$. Consider the potential function

$$\Phi = r \log(2n) - \sum_{i \in R} \lg w_i .$$

Consider now how our potential function Φ can change. Each time we make a mistake on a positive example, Φ decreases by at least 1. Each time we make a mistake on a negative example, Φ does not change. Each time the adversary adds a new relevant variable, Φ increases by at most $\log(2n) + 1$ ($\log(2n)$ for the increase in r and 1 for the possibility that the new weight w_i equals $1/2$ so $\lg w_i = -1$). Each time the adversary removes a relevant variable, Φ does not increase (and may decrease if the variable removed has weight less than $2n$). In summary, the only way that Φ can increase is by the adversary adding a new relevant variable, and each mistake on a positive example decreases Φ by at least 1; furthermore, Φ is initially zero and is always non-negative. Therefore, the number of mistakes we make on positive examples is bounded by $\log(2n)+1$ times the adversary's cost, proving the theorem. □

3.4 Learning from String-Valued attributes and the Infinite Attribute Model

The discussion so far has focused on learning over the instance space $\mathcal{X} = \{0,1\}^n$. I.e., examples have Boolean-valued attributes. Another common setting is one in which the attributes are string-valued; that is, $\mathcal{X} = (\Sigma^*)^n$. For instance, one attribute might represent an object's color, another its texture, etc. If the number of choices for each attribute is small, we can just convert this to the Boolean case, for instance by letting "x_1 = red" be a Boolean variable that is either true or false in any given example. However, if the number of choices for an attribute is large or is unknown apriori, this conversion may blow up the number of variables.

This issue motivates the "infinite attribute" learning model. In this model, there are infinitely many boolean variables $x_1, x_2, x_3, \ldots$, though any given example satisfies only finite number of them. An example is specified by listing the variables satisfied by it. For instance, a typical example might be $\{x_3, x_9, x_{32}\}$, meaning that these variables are true and the rest are false in the example. Let n be the size of (the number of variables satisfied by) the largest example seen so far. The goal of an algorithm in this setting is to make a number of mistakes polynomial in the size of the target function and n, but independent of the total number of variables (which is infinite). The running time per trial should be polynomial in the size of the target function and the description length of the longest example seen so far. It is not hard to see that this can model the situation of learning over $(\Sigma^*)^n$.

Some algorithms in the standard Boolean-attribute setting fail in the infinite attribute model. For instance, listing all variables and then crossing off the ones found to be irrelevant as in the simple disjunction-learning algorithm presented in Section 3.1 clearly does not work. The decision-list algorithm presented fails as well; in fact, there is no known polynomial-time algorithm for learning decision lists in this setting (see the Open Problems section).

On the other hand, algorithms such as Winnow can be adapted in a straightforward way to succeed in the infinite attribute model. More generally, the following theorem is known.

Theorem 9. *Let $\mathcal{C}$ be a projection and embedding-closed concept class*[2]*. If there is an attribute-efficient algorithm for learning $\mathcal{C}$ over $\{0,1\}^n$, then $\mathcal{C}$ can be learned in the Infinite-Attribute model.*

[2] This is just a "reasonableness condition" saying that one can take a concept in $\mathcal{C}$ defined on n_1 variables and embed it into a space with $n_2 > n_1$ variables and still stay within the class $\mathcal{C}$, and in the reverse direction, one can fix values of some of the variables and still have a legal concept. See [9] for details.

3.5 History

The Winnow algorithm was developed by Littlestone in his seminal paper [24], which also gives a variety of extensions and introduces the Mistake-Bound learning model. The Mistake Bound model is equivalent to the "extended equivalence query" model of Angluin [1], and is known to be strictly harder for polynomial-time algorithms than the PAC learning model of Valiant [34, 22] in which (among other differences) the adversary is required to select examples from a fixed distribution [6]. Agnostic learning is disussed in [23].

Littlestone [26] gives a variety of results on the behavior of Winnow in the presence of various kinds of noise. The improved bounds of Theorem 7 are from Auer and Warmuth [3]. The use of Winnow for learning changing concepts is folklore (and makes a good homework problem); Auer and Warmuth [3] provide a more sophisticated algorithm and analysis, achieving a stronger result than Theorem 8, in the style of Theorem 7. The Winnow algorithm has been shown to be quite successful in practical tasks as well, such as predicting links followed by users on the Web [2], and a calendar scheduling application [7].

The algorithm presented for learning decision lists is based on Rivest's algorithm for the PAC model [31], adapted to the Mistake Bound model by Littlestone [25] and Helmbold, Sloan and Warmuth [20]. The Infinite-Attribute model is defined in Blum [5] and Theorem 9 is from Blum, Hellerstein, and Littlestone [9].

4 Open Problems

1. **Can the bounds of Corollary 3 be achieved and improved with a smooth algorithm?** The bound of Corollary 3 is achieved using a "guess and double" algorithm that periodically throws out all it has learned so far and restarts using a new value of β. It would seem more natural (and likely to work better in practice) to just smoothly adjust β as we go along, never restarting from scratch. Can an algorithm of this form be shown to achieve this bound, preferably with even better constants? (See [12] for the precise constants.)

2. **Can Decision Lists be learned Attribute-Efficiently?** Recall from Section 3.1 that a *decision list* is a function of the form: "if ℓ_1 then b_1, else if ℓ_2 then b_2, else if ℓ_3 then b_3, ..., else b_m," where each ℓ_i is a literal (either a variable or its negation) and each $b_i \in \{0, 1\}$. We saw in Section 3.1 that decision lists with r relevant variables can be learned with at most $O(rn)$ mistakes in the mistake-bound model. An alternative approach using the Winnow algorithm makes $O(r^{2r} \log n)$ mistakes. *Can decision lists be learned attribute-efficiently? I.e., with mistake bound* $poly(r) \cdot polylog(n)$*?*

3. **Can Parity functions be learned Attribute-Efficiently?** Let $\mathcal{C}_{parity}$ denote the class of functions over $\{0, 1\}^n$ that compute the parity of some subset of variables. For instance, a typical function in $\mathcal{C}_{parity}$ would be $x_1 \oplus$

$x_5 \oplus x_{22}$. It is easy to learn $\mathcal{C}_{parity}$ in the mistake-bound model making at most n mistakes, by viewing each labeled example as a linear equality modulo 2 (each new example either is linearly dependent on the previous set and therefore its label can be deduced, or else it provides a new linearly independent vector). *Can $\mathcal{C}_{parity}$ be learned attribute-efficiently?*

4. **Can Decision Lists or Parity functions be learned in the Infinite Attribute model?** Can either the class of decision lists or the class of parity functions be learned in the Infinite Attribute model? For the case of decision lists, you may assume, if you wish, that none of the literals ℓ_i are negations of variables.

5. **Is there a converse to Theorem 9?**

6. **Can tolerance to random noise be boosted?** Suppose for some concept class $\mathcal{C}$ and some fixed constant noise rate $\eta > 0$ there exists a polynomial time algorithm A with the following property: for any target concept $c \in C$ and any distribution D on examples, A achieves an expected mistake rate less than $1/2 - 1/p(n)$ for some polynomial p after seeing polynomially many examples. Does this imply that there must exist a polynomial time algorithm B that succeeds in the same sense for *all* constant noise rates $\eta < 1/2$. (See Kearns [21] for related issues.)

7. **What Competitive Ratio can be achieved for learning with respect to the best Disjunction?** Is there a polynomial time algorithm that given any sequence of examples over $\{0,1\}^n$ makes a number of mistakes at most $cm_{disj} + p(n)$, where m_{disj} is the number of mistakes made by the best disjunction, for some constant c and polynomial p? How about $c = n^\alpha$ or $c = r^\alpha$ for some $\alpha < 1$, where r is the number of relevant variables in the best disjunction. (Making nm_{disj} mistakes is easy using any of the standard disjunction-learning algorithms, and we saw that the Winnow algorithm makes $O(rm_{disj})$ mistakes.)

8. **Can Disjunctions be Weak-Learned in the presence of adversarial noise?** For some polynomial $p(n)$ and some constant $c > 0$, does there exist an algorithm with the following guarantee: Given any sequence of t examples over $\{0,1\}^n$ such that at least a $(1-c)$ fraction of these examples are consistent with some disjunction over $\{0,1\}^n$, the algorithm makes at most $t[\frac{1}{2} - \frac{1}{p(n)}]$ mistakes (in expectation, if the algorithm is randomized). That is, given that there exists a disjunction that is "nearly correct" (say 99%) on the data, can the algorithm achieve a performance that is slightly ($1/poly$) better than guessing? The algorithm may require that $t \geq q(n)$ for some polynomial q.

9. **Can Linear Threshold Functions be Weak-Learned in the presence of adversarial noise?** Same question as above, except replace "disjunctions" with "linear threshold functions". An affirmative answer to this question would yield a quasi-polynomial ($n^{polylog(n)}$) time algorithm for learning

DNF formulas, and more generally for learning AC^0 functions, in the PAC learning model. This implication follows from standard complexity theory results that show that AC^0 can be approximated by low-degree polynomials.

5 Conclusions

This article has surveyed a collection of problems, models, and algorithms in Computational Learning Theory that look particularly interesting from the point of view of On-Line Algorithms. These include algorithms for combining the advice of experts, the model of on-line agnostic learning (or learning in the presence of worst-case noise) and the problem of learning a drifting target concept. It seems clear that a further crossover of ideas between Computational Learning Theory and On-Line Algorithms should be possible. Listed below are a few of the respective strengths and weaknesses of these areas where this crossover may prove to be especially fruitful.

The notion of state. The notion of an algorithm having a *state*, where there is a cost associated with changing state, is central to the area of On-Line Algorithms. This allows one to study problems in which the decisions made by an algorithm involve "doing something" rather than just predicting, and where the decisions made in the present (e.g., whether to rent or buy) affect the costs the algorithm will pay in the future. This issue has been virtually ignored in the Computational Learning Theory literature since that literature has tended to focus on prediction problems. In prediction problems the state of an algorithm is usually just its current hypothesis and there is no natural penalty for changing state. Nonetheless, as Computational Learning Theory moves to analyze more general sorts of learning problems, it seems inevitable that the notion of state will begin to play a larger role, and ideas from On-Line Algorithms will be crucial. Some work in this direction appears in [8].

Limiting the power of the adversary. In the On-Line Algorithms literature, it is usually assumed that the adversary has unlimited power to choose a worst-case sequence for the algorithm. In the machine learning setting, it is natural to assume there is some sort of regularity to the world (after all, if the world is completely random, there is nothing to learn). Thus, one often assumes that the world produces labels using a function from some limited concept class, or that examples are are drawn from some fixed distribution, or even that this fixed distribution is of some simple type. One can then parametrize one's results as a function of the adversary's power, producing especially good bounds when the adversary is relatively simple. This sort of approach may prove useful in On-Line Algorithms (in fact, it already has) for achieving less pessimistic sorts of bounds for many of the problems commonly studied.

Limiting the class of off-line algorithms being compared to. In the typical machine learning setup, if one does not restrict the adversary, then to achieve any non-trivial bound one must limit the class of off-line algorithms against which one is competing. This sort of approach may also be useful in On-Line Algorithms for achieving more reasonable bounds.

Acknowledgements

I would like to thank Yoav Freund for helpful discussions and pointers. This work was supported in part by NSF National Young Investigator grant CCR-9357793 and a Sloan Foundation Research Fellowship.

References

1. D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.
2. R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *1995 AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments*, March 1995.
3. P. Auer and M.K. Warmuth. Tracking the best disjunction. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 312–321, 1995.
4. D. Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific J. Math.*, 6:1–8, 1956.
5. A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9:373–386, 1992.
6. A. Blum. Separating distribution-free and mistake-bound learning models over the boolean domain. *SIAM J. Computing*, 23(5):990–1000, October 1994.
7. A. Blum. Empirical support for winnow and weighted-majority based algorithms: results on a calendar scheduling domain. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 64–72, July 1995.
8. A. Blum and C. Burch. On-line learning and the metrical task system problem. In *Proceedings of the 10th Annual Conference on Computational Learning Theory*, pages 45–53, 1997.
9. A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *J. Comp. Syst. Sci.*, 50(1):32–40, 1995.
10. A. Blum and A. Kalai. Universal portfolios with and without transaction costs. In *Proceedings of the 10th Annual Conference on Computational Learning Theory*, pages 309–313, 1997.
11. N. Cesa-Bianchi, Y. Freund, D. P. Helmbold, and M. Warmuth. On-line prediction and conversion strategies. In *Computational Learning Theory: Eurocolt '93*, volume New Series Number 53 of *The Institute of Mathematics and its Applications Conference Series*, pages 205–216, Oxford, 1994. Oxford University Press.
12. N. Cesa-Bianchi, Y. Freund, D.P. Helmbold, D. Haussler, R.E. Schapire, and M.K. Warmuth. How to use expert advice. In *Annual ACM Symposium on Theory of Computing*, pages 382–391, 1993.
13. T.M. Cover. Universal portfolios. *Mathematical Finance*, 1(1):1–29, January 1991.
14. T.M. Cover and E. Ordentlich. Universal portfolios with side information. *IEEE Transactions on Information Theory*, 42(2), March 1996.

15. A. DeSantis, G. Markowsky, and M. Wegman. Learning probabilistic prediction functions. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, pages 110–119, Oct 1988.
16. M. Feder, N. Merhav, and M. Gutman. Universal prediction of individual sequences. *IEEE Transactions on Information Theory*, 38:1258–1270, 1992.
17. D.P. Foster and R.V. Vohra. A randomization rule for selecting forecasts. *Operations Research*, 41:704–709, 1993.
18. Y. Freund. Predicting a binary sequence almost as well as the optimal biased coin. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, pages 89–98, 1996.
19. Y. Freund and R. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, pages 325–332, 1996.
20. D. Helmbold, R. Sloan, and M. K. Warmuth. Learning nested differences of intersection closed concept classes. *Machine Learning*, 5(2):165–196, 1990.
21. M. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 392–401, 1993.
22. M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of boolean formulae. In *Proceedings of the Nineteenth Annual ACM Symposium on the Theory of Computing*, pages 285–295, New York, New York, May 1987.
23. M. Kearns, R. Schapire, and L. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2/3):115–142, 1994.
24. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
25. N. Littlestone. personal communication (a mistake-bound version of Rivest's decision-list algorithm). 1989.
26. N. Littlestone. Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 147–156, Santa Cruz, California, 1991. Morgan Kaufmann.
27. N. Littlestone, P. M. Long, and M. K. Warmuth. On-line learning of linear functions. In *Proc. of the 23rd Symposium on Theory of Computing*, pages 465–475. ACM Press, New York, NY, 1991. See also UCSC-CRL-91-29.
28. N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.
29. N. Merhav and M. Feder. Universal sequential learning and decisions from individual data sequences. In *Proc. 5th Annu. Workshop on Comput. Learning Theory*, pages 413–427. ACM Press, New York, NY, 1992.
30. E. Ordentlich and T.M. Cover. On-line portfolio selection. In *COLT 96*, pages 310–313, 1996. A journal version is to be submitted to *Mathematics of Operations Research*.
31. R.L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.
32. H. Robbins. Asymptotically subminimax solutions of compound statistical decision problems. In *Proc. 2nd Berkeley Symp. Math. Statist. Prob.*, pages 131–148, 1951.
33. J. Shtarkov. Universal sequential coding of single measures. *Problems of Information Transmission*, pages 175–185, 1987.
34. L.G. Valiant. A theory of the learnable. *Comm. ACM*, 27(11):1134–1142, November 1984.

35. V. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann, 1990.
36. V. G. Vovk. A game of prediction with expert advice. In *Proceedings of the 8th Annual Conference on Computational Learning Theory*, pages 51–60. ACM Press, New York, NY, 1995.