Machine Learning, 29, 45–63 (1997)

Online Learning versus Offline Learning*

SHAI BEN-DAVID shai@cs.technion.ac.il
Computer Science Dept., Technion, Israel.

EYAL KUSHILEVITZ** eyalk@cs.technion.ac.il
Computer Science Dept., Technion, Israel.

YISHAY MANSOUR† mansour@gemini.math.tau.ac.il
Computer Science Dept., Tel-Aviv University, Israel.

Editor: Sally Goldman

Abstract. We present an off-line variant of the mistake-bound model of learning. This is an intermediate model between the *on-line* learning model (Littlestone, 1988, Littlestone, 1989) and the *self-directed* learning model (Goldman, Rivest & Schapire, 1993, Goldman & Sloan, 1994). Just like in the other two models, a learner in the off-line model has to learn an unknown concept from a sequence of elements of the instance space on which it makes "guess and test" trials. In all models, the aim of the learner is to make as few mistakes as possible. The difference between the models is that, while in the on-line model only the *set* of possible elements is known, in the off-line model the *sequence* of elements (i.e., the identity of the elements as well as the order in which they are to be presented) is known to the learner in advance. On the other hand, the learner is weaker than the self-directed learner, which is allowed to choose adaptively the sequence of elements presented to him.

We study some of the fundamental properties of the *off-line* model. In particular, we compare the number of mistakes made by the off-line learner on certain concept classes to those made by the on-line and self-directed learners. We give bounds on the possible gaps between the various models and show examples that prove that our bounds are tight.

Another contribution of this paper is the extension of the combinatorial tool of labeled trees to a unified approach that captures the various mistake bound measures of all the models discussed. We believe that this tool will prove to be useful for further study of models of incremental learning.

Keywords: On-Line Learning. Mistake-Bound, Rank of Trees

1. Introduction

The *mistake-bound* model of learning, introduced by Littlestone (Littlestone, 1988; 1989), has attracted a considerable amount of attention (e.g., (Littleston, 1988, Littlestone, 1989, Littlestone & Warmuth, 1994, Blum 1994, Blum 1992a, Maass, 1991, Chen & Maass, 1994, Helmbold, Littlestone & Long, 1992, Goldman, Rivest & Schapire, 1993, Goldman & Sloan, 1994)) and is recognized as one of the central models of computational learning

[*] This paper is an extended version of a paper that appeared in EuroCOLT 1995. In addition it contains some of the results of the paper "On Self-Directed Learning" by S. Ben-David, N. Eiron, and E. Kushilevitz that appeared in COLT 1995.

[**] http://www.cs.technion.ac.il/∼eyalk. This research was supported by Technion V.P.R. Fund 120-872 and by Japan Technion Society Research Fund.

[†] http://www.math.tau.ac.il/ mansour. This research was supported in part by *The Israel Science Foundation* administered by the Israel Academy of Science and Humanities, and by a grant of the Israeli Ministry of Science and Technology.

theory. Basically it models a process of incremental learning, where the learner discovers the ‘labels’ of instances one by one. At any given stage of the learning process, the learner has to predict the label of the next instance based on his current knowledge, i.e. the labels of the previous instances that it has already seen. The quantity that the learner would like to minimize is the number of mistakes it makes along this process. Two variants of this model were considered, allowing the learner different degrees of freedom in choosing the instances presented to him:

- The *on-line* model (Littlestone, 1988, Littlestone, 1989), in which the sequence of instances is chosen by an adversary and the instances are presented to the learner one-by-one.

- The *self-directed* model (Goldman, Rivest & Schapire, 1993, Goldman & Sloan, 1994), in which the learner is the one who chooses the sequence of instances; moreover, it may make his choices *adaptively*; i.e., each instance is chosen only after seeing the labels of all previous instances.

In the on-line model, the learner is faced with two kinds of uncertainties. The first is which function is the target function, out of all functions in the concept class which are consistent with the data. The second is what are the instances that it would be challenged on in the future. While the first uncertainty is common to almost any learning model, the second is particular to the on-line learning model.

A central aim of this research is to focus on the uncertainty regarding the target function by trying to “neutralize” the uncertainty that is involved in not knowing the future elements, and to understand the effect that this uncertainty has on the mistake-bound learning model. One way of doing this, is to allow the learner a full control of the sequence of instances, as is done in the *self-directed* model. Our approach is different: we define the *off-line* learning model as a one in which the learner knows the sequence of elements in advance. Since the difference between the on-line learner and the off-line learner is the uncertainty regarding the order of the instances, this comparison gives insight into the “information” that is in knowing the sequence.

Once we define the off-line cost of a sequence of elements, we can also define a *best sequence* (a sequence in which the optimal learner, knowing the sequence, makes the fewest mistakes) and a *worst sequence* (a sequence in which the optimal learner, knowing the sequence, makes the most mistakes). These are best compared to the *on-line* and *self-directed* models if we think of them in the following way:

- The *worst sequence* (off-line) model is a one in which the sequence of instances is chosen by an adversary but the whole sequence (without the labels) is presented to the learner before the prediction process starts.

- The *best sequence* (off-line) model is a one in which the whole sequence of instances is chosen by the learner *before* the prediction process starts.

Denote by $M_{on-line}(\mathcal{C})$, $M_{worst}(\mathcal{C})$, $M_{best}(\mathcal{C})$ and $M_{sd}(\mathcal{C})$ the number of mistakes made by the best learning algorithm in the online, worst sequence, best sequence and self-directed model (respectively) on the worst target concept in a concept class $\mathcal{C}$. Obviously, for all $\mathcal{C}$,

$$M_{on-line}(\mathcal{C}) \geq M_{worst}(\mathcal{C}) \geq M_{best}(\mathcal{C}) \geq M_{sd}(\mathcal{C}).$$

The main issue we consider is to what degree these measures can differ from one another. We emphasize that the only complexity measure is the number of mistakes and that other complexity measures, such as the computational complexity of the learning algorithms, are ignored. It is known that in certain cases M_{sd} can be strictly smaller than $M_{on-line}$. For example, consider the class of monotone monomials over n variables. It can be seen that this class has $M_{sd} = 1$ (Goldman & Sloan, 1994) but $M_{on-line} = n$ (Littlestone, 1988, Littlestone, 1989). In addition, we give examples that show, for certain concept classes, that M_{sd} may be smaller than M_{best} by a multiplicative factor of $O(\log n)$; hence, showing the power of adaptiveness. The following example shows that there are also gaps between M_{best} and M_{worst}. Given n points in the interval $[0, 1]$, consider the class of functions which are a suffix of this interval (i.e., the functions $f_a(x)$ that are 1 for $x \geq a$ and 0 otherwise). As there are n points, there are only $n + 1$ possible concepts, and therefore the *Halving* algorithm (Littlestone, 1988, Littlestone, 1989) is guaranteed to make at most $O(\log n)$ mistakes, i.e. $M_{on-line} = O(\log n)$. For this class, a best sequence would be receiving the points in increasing order, in which case the learner makes at most one mistake (i.e., $M_{best} = 1$). On the other hand, we show that the worst sequence forces $M_{worst} = \Theta(\log n)$ mistakes. An interesting question is what is the optimal strategy for a given sequence. We show a simple optimal strategy and prove that the number of mistakes is exactly the rank of a search tree (into which we insert the points in the order they appear in the sequence). We generalize this example, and show that for any concept class, the exact number of mistakes is the rank of a certain tree corresponding to the concept class and the particular sequence (this tree is based on the consistent extensions). This formalization of the number of mistakes, in the spirit of Littlestone‘s formalization for the on-line case (Littlestone, 1988, Littlestone, 1989), provides a powerful combinatorial characterization.

All the above examples raise the question of how large can these gaps be. We prove that the gaps demonstrated by the above mentioned examples are essentially the largest possible; more precisely, we show that $M_{on-line}$ can be greater than M_{sd} by at most a factor of $\log |\mathcal{X}|$, where $\mathcal{X}$ is the instance space (e.g., in the monomials example $\mathcal{X}$ is the set of 2^n boolean assignments).

The above result implies, in particular, that the ratio between the number of mistakes for the best sequence and the worst sequence is $O(\log n)$, where n is the length of the sequence.[1] We also show that $M_{worst} = \Omega(\sqrt{\log M_{on-line}})$, which implies that either both are constant or both are non-constant. Finally we show examples in which $M_{on-line} = \frac{3}{2} \cdot M_{worst}$, showing that $M_{worst} \neq M_{on-line}$. In a few cases we are able to derive better bounds: for the cases that $M_{worst} = 1$ and $M_{worst} = 2$ we show simple on-line algorithms that have at most 1 and 3 mistakes, respectively.

One way to view the relationships among the above models is through the model of “experts” (Cesa-Bianchi, et al., 1993, Feder, Merhav & Gutman, 1992, Merhav & Feder, 1993). For each sequence σ there is an expert, E_σ, that makes its predictions under the assumption that the sequence is σ. Let σ^* be the sequence chosen by the adversary, then the expert

E_{σ^*} makes at most M_{worst} mistakes. The on-line learner does not know the sequence σ^* in advance, so the question is how close can it get to the best expert, E_{σ^*}. The problem is that the number of experts is overwhelming; initially there are $n!$ experts (although the number of experts consistent with the elements of the sequence σ^* seen so far decreases with each element presented). Therefore, previous results about experts do not apply here.

The rest of this paper is organized as follows: In Section 2, we give formal definitions of the model and the measures of performance that are discussed in this paper, followed by some simple properties of these definitions. In Section 3, we give the definition of the *rank* of a tree (as well as some other related definitions) and prove some properties of it. In Section 4, we present the various gaps. Then, we characterize the off-line complexity (for completeness, we present in Section 4.2.1 a characterization of the same nature, based on (Littlestone, 1988, Littlestone, 1989), for the on-line complexity and for the self-directed complexity (Goldman & Sloan, 1994)) and we use these characterizations to obtain some basic results. Finally, in Section 5, we use these characterizations to study the gap between the on-line complexity and off-line complexity.

2. The Model

2.1. Basic Definitions

In this section we formally present our versions of the mistake bound learning model which is the subject of this work. The general framework is similar to the *on-line* learning model defined by Littlestone (Littlestone, 1988, Littlestone, 1989).

Let $\mathcal{X}$ be any set, and let $\mathcal{C}$ be a collection of boolean functions defined over the set $\mathcal{X}$ (i.e. $\mathcal{C} \subseteq \{f | f : \mathcal{X} \rightarrow \{0,1\}\}$). We refer to $\mathcal{X}$ as the *instance space* and to $\mathcal{C}$ as the *concept class*. Let S be a finite subset of $\mathcal{X}$. An *on-line learning algorithm with respect to* S (and a concept class $\mathcal{C}$) is an algorithm $\mathcal{A}$ that is given (in advance) S as an input; Then, it works in steps as follows: In the i-th step the algorithm is presented with a new element $s_i \in S$. It then outputs its prediction $p_i \in \{0,1\}$ and in response it gets the true value $c_t(s_i)$, where $c_t \in \mathcal{C}$ denotes the *target* function. The prediction p_i may depend on the set S, the values it has seen so far (and of course the concept class $\mathcal{C}$). The process continues until all the elements of S have been presented. Let $\sigma = s_1, s_2, \ldots, s_n$ denote the order according to which the elements of S are presented to the learning algorithm. Denote by $M(\mathcal{A}[S], \sigma, c_t)$ the number of mistakes made by the algorithm on a sequence σ as above and target function $c_t \in \mathcal{C}$, when the algorithm is given S in advance (i.e., the number of elements for which $p_i \neq c_t(s_i)$). Define the mistake bound of the algorithm, for a *fixed* S, as $M(\mathcal{A}[S]) \overset{\triangle}{=} \max_{\sigma, c_t} M(\mathcal{A}[S], \sigma, c_t)$. Finally, let

$$M_{\text{on-line}}(S, \mathcal{C}) \overset{\triangle}{=} \min_{\mathcal{A}} M(\mathcal{A}[S]) = \min_{\mathcal{A}} \max_{\sigma, c_t} M(\mathcal{A}[S], \sigma, c_t).$$

The original definitions of (Littlestone, 1988, Littlestone, 1989) are obtained (at least for finite $\mathcal{X}$) by considering $S = \mathcal{X}$.

An *off-line learning algorithm* is an algorithm $\mathcal{A}$ that is given (in advance) not only the set S, but also the actual sequence σ as an input. The learning process remains unchanged (except that each prediction p_i can now depend on the actual sequence, σ, and not only on the set of elements, S). Denote by $M(\mathcal{A}[\sigma], c_t)$ the number of mistakes made by an off-line algorithm, $\mathcal{A}$, on a sequence σ and a target c_t. Define $M(\mathcal{A}[\sigma]) \stackrel{\triangle}{=} \max_{c_t} M(\mathcal{A}[\sigma], c_t)$ and for a particular sequence σ, define

$$M(\sigma, \mathcal{C}) \stackrel{\triangle}{=} \min_{\mathcal{A}} M(\mathcal{A}[\sigma]) = \min_{\mathcal{A}} \max_{c_t} M(\mathcal{A}[\sigma], c_t).$$

For a given S, we are interested in the *best* and *worst* sequences. Denote by $M_{\text{best}}(S, \mathcal{C})$ the smallest value of $M(\sigma, \mathcal{C})$ over all σ, an ordering of S, and let σ_{best} be a sequence that achieves this minimum (if there are several such sequences pick one of them arbitrarily). Similarly, $M_{\text{worst}}(S, \mathcal{C})$ is the maximal value of $M(\sigma, \mathcal{C})$ and σ_{worst} is a sequence such that $M(\sigma_{\text{worst}}, \mathcal{C}) = M_{\text{worst}}(S, \mathcal{C})$.

A *self-directed learning algorithm* $\mathcal{A}$ is a one that chooses its sequence adaptively; hence the sequence may depend on the classifications of previous instances (i.e., on the target function). Denote by $M_{\text{sd}}(\mathcal{A}[S], c_t)$ the number of mistakes made by a self-directed algorithm $\mathcal{A}$ on a target function $c_t \in \mathcal{C}$, when the algorithm is given in advance S (the set from which it is allowed to pick its queries). Define $M_{\text{sd}}(\mathcal{A}[S]) \stackrel{\triangle}{=} \max_{c_t} M_{\text{sd}}(\mathcal{A}[S], c_t)$ and

$$M_{\text{sd}}(S, \mathcal{C}) \stackrel{\triangle}{=} \min_{\mathcal{A}} M_{\text{sd}}(\mathcal{A}[S]) = \min_{\mathcal{A}} \max_{c_t} M_{\text{sd}}(\mathcal{A}[S], c_t).$$

The following is a simple consequence of the definitions:

LEMMA 1 *For any $\mathcal{X}, \mathcal{C}$, and a finite $S \subseteq \mathcal{X}$,*

$$M_{\text{sd}}(S, \mathcal{C}) \leq M_{\text{best}}(S, \mathcal{C}) \leq M_{\text{worst}}(S, \mathcal{C}) \leq M_{\text{on-line}}(S, \mathcal{C}).$$

2.2. Relations to Equivalence Query Models

It is well known that the on-line learning model is, basically, equivalent to the *Equivalence Query* (EQ) model (Littlestone, 1989). It is not hard to realize that our versions of the on-line scenario give rise to corresponding variants of the EQ model. For this we need the following definitions:

- An *equivalence-query learning algorithm with respect to* S (and a concept class $\mathcal{C}$) is an algorithm $\mathcal{A}$ that is given in advance S as an input; Then, it works in steps as follows: In the i-th step the algorithm outputs its hypothesis, $h_i \subseteq S$, and in response it gets a counterexample; i.e., an element $x_i \in (h_i \Delta c_t) \cap S$, where $c_t \in \mathcal{C}$ denotes the *target* function. The process goes on until $(h_i \Delta c_t) \cap S = \emptyset$ (i.e. $h_i = c_t \cap S$).

- Let F denote the function that chooses the counterexamples x_i. We denote by $EQ(\mathcal{A}[S], F, c_t)$ the number of counterexamples, x_i, presented by F to the algorithm, $\mathcal{A}$, in the course of a learning process on the target, c_t, when $\mathcal{A}$ knows S in advance (but does not know F).

- Finally, let $EQ(S,\mathcal{C}) \overset{\triangle}{=} \min_{\mathcal{A}} \max_{F,c_t \in \mathcal{C}} EQ(\mathcal{A}[S], F, c_t)$.

Note that the original definitions of Angluin (Angluin, 1989) are obtained by considering $S = \mathcal{X}$. The following is a well known (and easy to prove) fact:

CLAIM 1 *For every $\mathcal{X}$, $\mathcal{C}$ and $S \subseteq \mathcal{X}$ as above, $EQ(S,\mathcal{C}) = M_{\text{on-line}}(S,\mathcal{C})$.*

One aspect of the last definition above is that it considers the worst case performance of the learner over all possible choices, F, of counterexamples to its hypotheses. It turns out that by relaxing the definition so that the learner is only confronted with F's of a certain type, one gets EQ models that match the various offline learning measures presented in the previous subsection.

- Let σ denote an ordering of the set S. Let F_σ be the following strategy for choosing counterexamples. Given a hypothesis h, the counterexample, $F_\sigma(h)$, is the minimal element of $(h \Delta c_t) \cap S$, according to the ordering σ.

- Let $EQ(\sigma,\mathcal{C}) \overset{\triangle}{=} \min_{\mathcal{A}} \max_{c_t \in \mathcal{C}} EQ(\mathcal{A}[\sigma], F_\sigma, c_t)$.

- Let $EQ_{\text{best}}(S,\mathcal{C}) \overset{\triangle}{=} \min_\sigma EQ(\sigma,\mathcal{C})$, and let $EQ_{\text{worst}}(S,\mathcal{C}) \overset{\triangle}{=} \max_\sigma EQ(\sigma,\mathcal{C})$.

This variant of the equivalence query model in which the minimal counterexample is provided to the algorithm is studied, e.g., in (Porat & Feldman, 1991).

CLAIM 2 *For every $\mathcal{X}$, $\mathcal{C}$ and $S \subseteq \mathcal{X}$ as above, for every ordering σ of S, $EQ(\sigma,\mathcal{C}) = M(\sigma,\mathcal{C})$.*

Proof: Given an EQ algorithm for $(S, \mathcal{C}, \sigma)$ construct an off-line algorithm by predicting, on each element s_i, the value that the current hypothesis of the EQ algorithm, h_{k_i}, assigns to s_i. Whenever the teacher's response indicates a prediction was wrong, present that element as a counterexample to the EQ algorithm (and replace the current hypothesis by its revised hypothesis).

For the other direction, given an offline algorithm, define at each stage of its learning process a hypothesis by assigning to each unseen element, $s \in S$, the value the algorithm would have guessed for s if it got responses indicating it made no mistakes along the sequence, σ, from the current stage up to that element. Whenever a counterexample is being presented, pass on to the offline algorithm the fact that it has erred on that element (and update the hypothesis according to its new guesses). ■

COROLLARY 1 *For every $\mathcal{X}$, $\mathcal{C}$ and $S \subseteq \mathcal{X}$ as above,*

1. $EQ_{\text{best}}(S,\mathcal{C}) = M_{\text{best}}(S,\mathcal{C})$.
2. $EQ_{\text{worst}}(S,\mathcal{C}) = M_{\text{worst}}(S,\mathcal{C})$.

3. Labeled Trees

A central tool that we employ in the quantitative analysis in this work is the notion of ranks of trees. We shall consider certain classes of labeled trees, depending upon the classes to be learned and the type of learning we wish to analyze. The following section introduces these technical notions and their basic combinatorial properties.

3.1. Rank of Trees

In this subsection we define the notion of the *rank* of a binary tree (see, e.g., (Cormen, Leiserson & Rivest, 1990, Ehrenfeucht & Haussler, 1989, Blum, 1992b)), which plays a central role in this paper. We then prove some simple properties of this definition.

For a tree T, if T is empty then rank$(T) = -1$. Otherwise, let T_L be its left subtree and T_R be its right subtree. Then,

$$\text{rank}(T) = \begin{cases} \max\{\text{rank}(T_L), \text{rank}(T_R)\} & \text{if rank}(T_L) \neq \text{rank}(T_R) \\ \text{rank}(T_L) + 1 & \text{otherwise} \end{cases}$$

For example, the rank of a leaf is 0.
Let $\binom{d}{\leq r}$ denotes $\sum_{i=0}^{r} \binom{d}{i}$. The following lemma is a standard fact about the rank:

LEMMA 2 *A depth d rank r tree has at most $\binom{d}{\leq r}$ leaves.*

Proof: By induction on d and r. If $r = 0$ then there is exactly one leaf (if there were two or more, then their least common ancestor is of rank 1). If $d = 1$ either there is one leaf (which is a special case of $r = 0$) or two leaves, in which case r must be 1. In all these cases the claim holds. For the induction step, let T be a depth d rank r tree. Each of T_L and T_R are of depth at most $d - 1$ and, by the definition of rank, in the worst case one of them is of rank r and the other of rank $r - 1$. Hence, by the induction hypothesis, the number of leaves is bounded by

$$\sum_{i=0}^{r} \binom{d-1}{i} + \sum_{i=0}^{r-1} \binom{d-1}{i} = \binom{d}{0} + \sum_{i=1}^{r} \left(\binom{d-1}{i} + \binom{d-1}{i-1} \right) = \sum_{i=0}^{r} \binom{d}{i}$$

which completes the proof. ■

If r is small relative to d then it may be convenient to use the weaker d^r $(\geq \binom{d}{\leq r})$ bound on the number of leaves.

A *subtree* of a tree T is a subset of the nodes of T ordered by the order induced by T.

LEMMA 3 *The rank of a binary tree T is at least k iff it has a subtree T' which is a complete binary tree of depth k.*

Proof: Mark in T the nodes where the rank increases. Those are the nodes of T'. For a marked node with rank i, each of its children in T has rank $i - 1$, hence it has a marked descendant with rank $i-1$. Therefore T' is a complete binary tree. For the other direction, note that the rank of a tree is at least the rank of any of its subtrees, and that a complete binary tree of depth k has rank k. ■

LEMMA 4 *Let T be a complete binary tree of depth k. Let $L_1,\ldots,L_t$ be a partition of the leaves of T into t disjoint subsets. For $1 \le i \le t$, define T_i to be the subtree induced by the leaves in L_i (that is, T_i is the tree consists of all ancestors of leaves in L_i). Then, there exists $1 \le i \le t$ such that*

$$\text{rank}(T_i) \ge \left\lfloor \frac{k}{t} \right\rfloor .$$

Proof: The proof is by induction on t. For $t = 1$ we have $T_1 = T$ hence the claim is obvious. For $t > 1$ we consider the nodes in depth $\lfloor k/t \rfloor$ in T. There are two cases: (a) if all these nodes belong to all trees T_i then each of these trees contains a complete subtree of depth $\lfloor k/t \rfloor$ and by Lemma 3 each of them has rank of at least $\lfloor k/t \rfloor$. (b) if there exists a node v in depth $\lfloor k/t \rfloor$ which does not belong to all the trees T_i then we consider the subtree T' whose root is v and consists of all the nodes below v. By the definition of v, the leaves of the tree T' belong to at most $t-1$ of the sets L_i. In addition the depth of T' is at least $\frac{(t-1)k}{t}$. Hence, by induction hypothesis, one of the subtrees T_i' of T' is of rank at least

$$\left\lfloor \frac{1}{t-1} \cdot \frac{(t-1)k}{t} \right\rfloor = \left\lfloor \frac{k}{t} \right\rfloor .$$

Finally note that T_i' is a subtree of T_i hence T_i has the desired rank. ■

Let us just mention that the above lower bound, on the rank of the induced subtrees, is essentially the best possible. For example, take T to be a complete binary tree. Each leaf corresponds to a path from the root to this leaf. Call an edge of such a path a left (right) edge if it goes from a node to its left (right) son. Let L_0 (L_1) be the set of leaves with more left (right) edges. Then, it can be verified that $rank(T_0) = rank(T_1) = k/2$.

3.2. Labeled Trees

Let $\mathcal{X}$ denote some domain set, $S \subseteq \mathcal{X}$ and $\mathcal{C} \subseteq \{0,1\}^{\mathcal{X}}$ as above.

- An *$\mathcal{X}$-labeled tree* is a pair, (T, F), where T is a binary tree and F a function mapping the internal nodes of T into $\mathcal{X}$. Furthermore, we impose the following restriction on F:

$$\forall t \ne t' \in T,\ \text{ if } t' \text{ is an ancestor of } t \text{ then } F(t') \ne F(t).$$

- A *branch* in a tree is a path from the root to a leaf. It follows that the above mapping F is one to one on branches of T.

- A branch $(t_1,\ldots,t_n)$ of T *realizes a function*

$$h : \{F(t_1),\ldots,F(t_{n-1})\} \mapsto \{0,1\}$$

if for all $1 \leq i < n$, t_{i+1} is a left son of t_i if and only if $h(F(t_i)) = 1$. Note that a branch can realize more than one function. On the other hand, if $\{F(t_1), \ldots, F(t_n)\} = \mathcal{X}$ then the branch realizes a single function.

- An $\mathcal{X}$-labeled tree is an $(S, \mathcal{C})$-*tree* if the set of functions realized by its branches is exactly $\mathcal{C}_S$.
- Let $\mathcal{T}_S^{\mathcal{C}}$ denote the set of all $(S, \mathcal{C})$-trees.
- For a sequence $\sigma = (s_1, \ldots, s_n)$ of elements of $\mathcal{X}$, let $T_\sigma^{\mathcal{C}}$ denote the maximal tree in $\mathcal{T}_S^{\mathcal{C}}$ for which every node v in the k-th level is labeled $F(v) = s_k$.

Note, that using this notation, a class $\mathcal{C}$ shatters the set of elements of a sequence, σ, if and only if $T_\sigma^{\mathcal{C}}$ is a complete binary tree (recall that a class $\mathcal{C}$ *shatters* a set $\{s_1, \ldots, s_n\}$ if for every $b_1, \ldots, b_k \in \{0, 1\}$ there exists a function $f \in \mathcal{C}$ that for all i ($1 \leq i \leq k$) satisfies $f(s_i) = b_i$). We can therefore conclude that, for any class $\mathcal{C}$,

$$\text{VC-dim}(\mathcal{C}) = \max\{\textit{rank}(T_\sigma^{\mathcal{C}}) \ : \ T_\sigma^{\mathcal{C}} \in \mathcal{T}_{\mathcal{X}}^{\mathcal{C}} \text{ and } T_\sigma^{\mathcal{C}} \text{ is a complete binary tree}\}.$$

(We shall usually omit the superscript $\mathcal{C}$ when it is clear from the context.)

4. Gaps between the Complexity Measures

Lemma 1 provides the basic inequalities concerning the learning complexity of the different models. In this section we turn to a quantitative analysis of the sizes of possible gaps between these measures. We begin by presenting, in Section 4.1, some examples of concept classes for which there exist large gaps between the learning complexity in different models. In Section 4.3, we prove upper bounds on the possible sizes of these gaps, bounds that show that the examples of Section 4.1 obtain the maximal possible gap sizes. A useful tool in our analysis is a characterization of the various complexity measures as the ranks of certain trees. This characterization is given in section 4.2.

Let us begin our quantitative analysis by stating a basic upper bound on the learning complexity in the most demanding model (from the student's point of view), namely, $M_{\text{on-line}}(S, \mathcal{C})$. Given an instance space $\mathcal{X}$, a concept class $\mathcal{C}$ and a set S we define $\mathcal{C}_S$ to be the projection of the functions in $\mathcal{C}$ on the set S (note that several functions in $\mathcal{C}$ may collide into a single function in $\mathcal{C}_S$). Using the Halving algorithm (Littlestone, 1988, Littlestone, 1989) we get,

THEOREM 1 *For all $\mathcal{X}, \mathcal{C}$ and S as above $M_{\text{on-line}}(S, \mathcal{C}) \leq \log |\mathcal{C}_S|$.*

4.1. Some Examples

The first example demonstrates that $M_{\text{best}}(S, \mathcal{C})$ may be much smaller than $M_{\text{worst}}(S, \mathcal{C})$ and $M_{\text{on-line}}(S, \mathcal{C})$. This example was already mentioned in the introduction and appears here in more details.

	z	x_1	x_2	$\dots$		x_d	$\dots$					x_{2^d}	y_1	y_2	$\dots$		y_d	$\dots$					y_{2^d}
f_1	0	1	0	0			$\dots$				0	0	0	0	$\dots$	0	0	$\dots$	0	0	$\dots$	0	0
f_2	0	0	1	0			$\dots$				0	0	0	0	$\dots$	0	1	$\dots$	0	0	$\dots$	0	1
$\vdots$	$\vdots$						$\vdots$								$\vdots$			$\vdots$			$\vdots$		
f_{2^d}	0	0	0	0			$\dots$				0	1	1	1	$\dots$	1	1	$\dots$	1	1	$\dots$	1	1
g_1	1	0	0	$\dots$	0	0	$\dots$	0	0	$\dots$	0	0	1	0	0			$\dots$				0	0
g_2	1	0	0	$\dots$	0	1	$\dots$	0	0	$\dots$	0	1	0	1	0			$\dots$				0	0
$\vdots$	$\vdots$			$\vdots$			$\vdots$			$\vdots$								$\vdots$					
g_{2^d}	1	1	1	$\dots$	1	1	$\dots$	1	1	$\dots$	1	1	0	0	0			$\dots$				0	1

Figure 1. The concept class of Example

Example: Let $\mathcal{X}$ be the unit interval $[0,1]$. For, $0 \leq a \leq 1$ define the function $f_a(x)$ to be 0 if $x \leq a$ and 1 if $x > a$. Let $\mathcal{C} \triangleq \{f_a : 0 \leq a \leq 1\}$. In other words, the concept class $\mathcal{C}$ consists of all intervals of the form $[a, 1]$ for $0 \leq a \leq 1$. Let $n = 2^m$ and $S = \{\frac{1}{n}, \frac{2}{n}, \dots, \frac{n}{n}\}$. By Theorem 1, it is easy to see that in this example $M_{\text{on-line}}(S, \mathcal{C}) \leq \log(n+1)$. We would like to understand how an off-line algorithm performs in this case.

Clearly, for every sequence σ, an adversary can always force a mistake on the first element of the sequence. Hence, $M_{\text{best}}(S, \mathcal{C}) \geq 1$. To see that $M_{\text{best}}(S, \mathcal{C}) = 1$ let $\sigma_{\text{best}} = (\frac{1}{n}, \frac{2}{n}, \dots, \frac{n}{n})$. On this sequence the following strategy makes at most 1 mistake: predict "0" until a mistake is made. Then, all the other elements are "1"s of the function. For a worst sequence consider

$$\sigma_{\text{worst}} = \left(\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \frac{7}{8}, \dots, \frac{1}{2^m}, \frac{3}{2^m}, \dots, \frac{2^m - 1}{2^m}, 1\right).$$

It may be seen that the adversary can force the learning algorithm to make one mistake on each of the sets $\{\frac{1}{2}\}$, $\{\frac{1}{4}, \frac{3}{4}\}$, $\{\frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \frac{7}{8}\}, \dots, \{\frac{1}{2^m}, \frac{3}{2^m}, \dots, \frac{2^m-1}{2^m}\}$, and hence a total of $m = \log n$ mistakes. This is the worst possible, as this performance is already granted for an on-line algorithm as discussed above (and, by Lemma 1 an off-line algorithm can always match the on-line performance). □

The next example shows that for all n there exist sets $S = \mathcal{X}$ of size n, and a concept class $\mathcal{C}$, such that $M_{\text{sd}}(S, \mathcal{C}) \leq 2$ and $M_{\text{best}}(S, \mathcal{C}) = \Omega(\log n)$.

Example: Without loss of generality, assume that for some value d, $n = 2 \cdot 2^d + 1$. Let $S = \mathcal{X} = \{z, x_1, \dots, x_{2^d}, y_1, \dots, y_{2^d}\}$. The concept class $\mathcal{C}$ (see Figure 1) consists of $2 \cdot 2^d$ functions $f_1, \dots, f_{2^d}, g_1, \dots, g_{2^d}$. Each function f_i is defined as follows: $f_i(z) = 0$; $f_i(x_i) = 1$; $f_i(x_j) = 0$ for all $j \neq i$ (i.e., there is a single x_i which is assigned 1 and hence can be viewed as an indicator for the corresponding function f_i). The elements $y_1, \dots, y_{2^d}$ are partitioned into $2^d/d$ "blocks" each of size d. In each of these blocks the 2^d functions $f_1, \dots, f_{2^d}$ get all the 2^d possible combinations of values (as in Figure 1). The functions $g_1, \dots, g_{2^d}$ are defined similarly by switching the roles of x's and y's. More precisely, $g_i(z) = 1$; $g_i(y_i) = 1$; $g_i(y_j) = 0$ for all $j \neq i$ (i.e., this time y_i serves as an indicator for the corresponding function g_i). Again, the elements $x_1, \ldots, x_{2^d}$ are partitioned into $2^d/d$ blocks each of size d. In each of these blocks the 2^d functions $g_1, \ldots, g_{2^d}$ get all the 2^d possible combinations of values.

To see that $M_{\text{sd}}(S, \mathcal{C}) \leq 2$ we describe a self-directed learner for this concept class. The learner first asks about z and predicts in an arbitrary way. Whether it is right or wrong the answer indicates whether the target functions is one of the f_i's or one of the g_i's. In each case the learner looks for the corresponding indicator. That is, if $c_t(z) = 0$ it asks the x's one by one, predicting 0 on each. A mistake on some x_i (i.e., $c_t(x_i) = 1$) immediately implies that the target function is f_i and no mistakes are made anymore. Symmetrically, if $c_t(z) = 1$ the learner asks the y's one by one, predicting 0 on each. A mistake on some y_i (i.e., $c_t(y_i) = 1$) implies that the target function is g_i and again no mistakes are made anymore. In any case, the total number of mistakes is at most 2.

We now prove that $M_{\text{best}}(S, \mathcal{C}) = \Omega(\log n)$.[2] The idea is that a learner must choose its sequence in advance, but does not know whether it looks for one of the f_i's or one of the g_i's. Formally, let σ be the (best) sequence chosen by the learner. We describe a strategy for the adversary to choose a target function in a way that forces the learner at least $d/4 = \Omega(\log n)$ mistakes. Let σ' be a prefix of σ of length $2d$. The adversary considers the number of x_i's queried in σ' versus the number of y_i's. Assume, without loss of generality, that the number of x_i's in σ' is smaller than the number of y_i's in σ'. The adversary then restricts itself to choosing one of the f_i's. Moreover, it eliminates all those functions f_i whose corresponding element x_i appears in σ'. Still, there are at least $2^d/2$ possible functions to choose from. Now, consider the y's queried in σ'. By the construction of $\mathcal{C}$ we can partition the elements $y_1, \ldots, y_{2^d}$ into d "groups" of size $2^d/d$ such that every function f_j gives the same value for all elements in each group. There are at least $2d/2$ elements y's that are queried in σ' and they belong to ℓ groups. By simple counting, $\frac{d}{2} \leq \ell \leq d$. We estimate the number of possible behaviors on these ℓ groups as follows: originally all $2d$ behaviors on the d groups were possible. Hence, to eliminate one of the behaviors on the ℓ elements one needs to eliminate $2^{d-\ell}$ functions. As we eliminated at most $2^d/2$ functions, the number of eliminated behaviors is at most $\frac{1}{2}2d/2^{d-\ell} = \frac{1}{2}2^\ell$. In other words, there are at least $\frac{1}{2}2^\ell$ behaviors on these ℓ elements. On the other hand, if we are guaranteed to make at most r mistakes it follows from Theorem 4 and Lemma 2 that the number of functions is at most $\binom{\ell}{\leq r}$. Hence, r must be at least $\ell/2 \geq d/4 = \Omega(\log n)$.

□

4.2. Characterizing $M(\sigma, \mathcal{C})$ Using the Rank

The main goal of this section is to characterize the measure $M(\sigma, \mathcal{C})$. As a by-product, we present an optimal offline prediction algorithm. I.e., an algorithm, $\mathcal{A}$, such that for every sequence σ, $M(\mathcal{A}[\sigma]) = M(\sigma, \mathcal{C})$.

The next theorem provides a characterization of $M(\sigma, \mathcal{C})$ in terms of the rank of the tree $T_\sigma^{\mathcal{C}}$ (for any concept class $\mathcal{C}$ and any sequence σ). A similar characterization was proved by Littlestone (Littlestone, 1988, Littlestone, 1989) for the *on-line* case (see section 4.2.1 below).

THEOREM 2 *For all $\mathcal{X}, \mathcal{C}$ and σ as above, $M(\sigma, \mathcal{C}) = \text{rank}(T_\sigma^{\mathcal{C}})$.*

Proof: To show that $M(\sigma, \mathcal{C}) \leq \text{rank}(T_\sigma)$ we present an appropriate algorithm. For predicting on s_1, the algorithm considers the tree T_σ, defined above, whose root is s_1. Denote by T_L its left subtree and by T_R its right subtree. If $\text{rank}(T_L) > \text{rank}(T_R)$ the algorithm predicts "0", if $\text{rank}(T_L) < \text{rank}(T_R)$ the algorithm predicts "1", otherwise ($\text{rank}(T_L) = \text{rank}(T_R)$) the algorithm can predict arbitrarily. Again, recall that in the case that $\text{rank}(T_L) = \text{rank}(T_R)$, by the definition of rank, both $\text{rank}(T_L)$ and $\text{rank}(T_R)$ are smaller than $\text{rank}(T_\sigma)$. Therefore, at each step the algorithm uses for the prediction a subtree of T_σ which is consistent with all the values it has seen so far. To conclude, at each step where the algorithm made a mistake, the rank decreased by (at least) 1, so no more than $\text{rank}(T_\sigma)$ mistakes are made.

To show that no algorithm can do better, we present a strategy for the adversary for choosing a target in $\mathcal{C}$ so as to guarantee that a given algorithm $\mathcal{A}$ makes at least $\text{rank}(T_\sigma)$ mistakes. The adversary constructs for itself the tree T_σ. At step i, it holds a subtree T whose root is a node marked s_i which is consistent with the values it already gave to $\mathcal{A}$ as the classification of $s_1, \ldots, s_{i-1}$. After getting $\mathcal{A}$'s prediction on s_i the adversary decides about the true values as follows: If one of the subtrees, either T_L or T_R, has the same rank as the rank of T then it chooses the value according to this subtree. Note that, by definition of rank, at most one of the subtrees may have this property, so this is well defined. In this case, it is possible that $\mathcal{A}$ guessed the correct value (for example, the algorithm we described above does this) but the rank of the subtree that will be used by the adversary in the $i+1$-th step is not decreased. The second possible case, by the definition of rank, is that the rank of both T_L and T_R is smaller by 1 than the rank of T. In this case, the adversary chooses the negation of $\mathcal{A}$'s prediction; hence, in such a step $\mathcal{A}$ makes a mistake and the rank is decreased by 1. Therefore, the adversary can force a total of $\text{rank}(T_\sigma)$ mistakes.

■

The above theorem immediately implies:

COROLLARY 2 *For all $\mathcal{X}, \mathcal{C}$ and S as above,*

$$M_{\text{worst}}(S, \mathcal{C}) = \max\left\{\text{rank}(T) \;:\; T = T_\sigma^{\mathcal{C}},\;\; \sigma \textit{ is an ordering of } S\right\}.$$

$$M_{\text{best}}(S, \mathcal{C}) = \min\left\{\text{rank}(T) \;:\; T = T_\sigma^{\mathcal{C}},\;\; \sigma \textit{ is an ordering of } S\right\}.$$

Remark. It is worth noting that, by Sauer's Lemma (Sauer, 1972), if the concept class $\mathcal{C}$ has VC dimension d then the size of $T_\sigma^{\mathcal{C}}$ is bounded by n^d (where n, as usual, is the length of σ). It follows that, for $\mathcal{C}$ with small VC, the tree is small and therefore, if consistency can be checked efficiently then the construction of the tree is efficient. This, in turn, implies the efficiency of the generic (optimal) off-line algorithm of the above proof, for classes with "small" VC dimension.

Example: Consider again the concept class of Example 4.1. Note that in this case, the tree T_σ is exactly the *binary search tree* corresponding to the sequence $\sigma = s_1 s_2 \ldots s_n$. Namely,

T_σ is the tree constructed by starting with an empty tree and performing the sequence of operations $insert(s_1), insert(s_2), \ldots, insert(s_n)$ (for details see, e.g. (Cormen, Leiserson & Rivest, 1990)). Hence, $M(\sigma, \mathcal{C})$ is exactly the rank of this search tree. □

4.2.1. Characterizing the On-line and Self-Directed Learning

To complete the picture one would certainly like to have a combinatorial characterization of $M_{\text{on-line}}(S, \mathcal{C})$ as well. Such a characterization was given by Littlestone (Littlestone, 1988, Littlestone, 1989). We reformulate this characterization in terms of ranks of trees. The proof remains similar to the one given by Littlestone (Littlestone, 1988, Littlestone, 1989) and we provide it here for completeness.

THEOREM 3 *For all $\mathcal{X}, \mathcal{C}$ and S as above,*

$$M_{\text{on-line}}(S, \mathcal{C}) = \max \left\{ \text{rank}(T) \ : \ T \in \mathcal{T}_S^{\mathcal{C}} \right\}.$$

Proof: To show that $M_{\text{on-line}}(S, \mathcal{C}) \geq \max \left\{ \text{rank}(T) \ : \ T \in \mathcal{T}_S^{\mathcal{C}} \right\}$ we use an adversary argument similar to the one used in the proof of Theorem 2. The adversary uses the tree that gives the maximum in the above expression to choose both the sequence and the classification of its elements, so that at each time that the rank is decreased by 1 the prediction algorithm makes a mistake.

To show that $M_{\text{on-line}}(S, \mathcal{C})$ is at most $m = \max \left\{ \text{rank}(T) \ : \ T \in \mathcal{T}_S^{\mathcal{C}} \right\}$ we present an appropriate algorithm, which is again similar to the one presented in the proof of Theorem 2. For predicting on $s \in S$, we first define $\mathcal{C}_S^0$ to be all the functions in $\mathcal{C}_S$ consistent with $s = 0$, and $\mathcal{C}_S^1$ to be all the functions in $\mathcal{C}_S$ consistent with $s = 1$. The algorithm compares $\max \left\{ \text{rank}(T) \ : \ T \in \mathcal{T}_S^{\mathcal{C}^0} \right\}$ and $\max \left\{ \text{rank}(T) \ : \ T \in \mathcal{T}_S^{\mathcal{C}^1} \right\}$ and predicts according to the larger one. The crucial point is that at least one of these two values must be strictly smaller than m otherwise there is a tree in $\mathcal{T}_S^{\mathcal{C}}$ whose rank is more than m. The prediction continues in this way, so that the maximal rank is decreased with each mistake. ■

Finally, the following characterization is implicit in (Goldman & Sloan, 1994):

THEOREM 4 *For all $\mathcal{X}, \mathcal{C}$ and S as above,*

$$M_{\text{sd}}(S, \mathcal{C}) = \min \left\{ \text{rank}(T) \ : \ T \in \mathcal{T}_S^{\mathcal{C}} \right\}.$$

Proof: Consider the tree T whose rank is the minimal one in $\mathcal{T}_S^{\mathcal{C}}$. We will show that $M_{\text{sd}}(S, \mathcal{C})$ is at most the rank of T. For this, we present an appropriate algorithm that makes use of this tree. At each point, the learner asks for the instance which is the current node in the tree. In addition, it predicts according to the subtree of the current node whose rank is higher (arbitrarily, if the ranks of the two subtrees are equal). The true classification determines the child of the current node from which the learner needs to proceed. It follows from the definition of rank that whenever the algorithm makes a mistake the remaining subtree has rank which is strictly smaller than the previous one.

For the other direction, given a strategy for the learner that makes at most $M_{\text{sd}}(S, \mathcal{C})$ mistakes we can construct a tree T that describes this strategy. Namely, at each point the instances that the learner will ask at the next stage, given the possible classifications of the current instance, determine the two children of the current node. Now, if the rank of T was more than $M_{\text{sd}}(S, \mathcal{C})$ then this gives the adversary a strategy to fool the learner: at each node classify the current instance according to the subtree with higher rank. If the ranks of both subtrees are equal then on any answer by the algorithm the adversary says the opposite. By the definition of rank, this gives rank(T) mistakes. Hence, rank(T) is at most $M_{\text{sd}}(S, \mathcal{C})$ and certainly the minimum over all trees can only be smaller. ■

4.3. A Bound on the Size of the Gap

A natural question is how large can be the gaps between the various complexity measures. For example, what is the maximum ratio between $M_{\text{worst}}(S, \mathcal{C})$ and $M_{\text{best}}(S, \mathcal{C})$. In Example 4.1 the best is 1 and the worst is $\log n$, which can be easily generalized to k versus $\Theta(k \log n)$. The following theorem shows that the gap between the smallest measure, $M_{\text{sd}}(S, \mathcal{C})$, and the largest measure, the on-line cost, cannot exceed $O(\log n)$. This, in particular, implies a similar bound for the gap between σ_{best} and σ_{worst}. By Example 4.1, the bound is tight; i.e., there are cases which achieve this gap. Similarly, the gap between $M_{\text{sd}}(S, \mathcal{C})$ and $M_{\text{best}}(S, \mathcal{C})$ exhibited by Example is also optimal.

THEOREM 5 *For $\mathcal{X}, \mathcal{C}$ and S of size n as above,*

$$M_{\text{on-line}}(S, \mathcal{C}) \leq M_{\text{sd}}(S, \mathcal{C}) \cdot \log n.$$

We shall present two quite simple but very different proofs for this theorem. The first proof employs the tool of labeled trees (but gives a slightly weaker result) while the second is by an information - theoretic argument.

Proof: [using labeled trees] Consider the tree T that gives the minimum in Theorem 4. Its depth is n and its rank, by Theorem 4, is $m = M_{\text{sd}}(S, \mathcal{C})$. By Lemma 2, this tree contains at most $\binom{n}{\leq m}$ leaves. That is, $|\mathcal{C}_S| \leq \binom{n}{\leq m}$. By Theorem 1, $M_{\text{on-line}}(S, \mathcal{C}) \leq \log \binom{n}{\leq m} = O(m \cdot \log n)$. ■

Proof: [information theoretic argument] Let $\mathcal{C}_S$ be the projection of the functions in $\mathcal{C}$ on the set S (note that several functions in $\mathcal{C}$ may collide into a single function in $\mathcal{C}_S$). Consider the number of bits required to specify a function in $\mathcal{C}_S$. On one hand, at least $\log |\mathcal{C}_S|$ bits are required. On the other hand, any self-directed learning algorithm that learns this class yields a natural coding scheme: answer the queries asked by the algorithm according to the function $c \in \mathcal{C}_S$; the coding consists of the list of names of elements of S on which the prediction of the algorithm is wrong. This information is enough to uniquely identify c. It follows that $M_{\text{sd}}(S, \mathcal{C}) \cdot \log n$ bits are enough. Hence,

$$\log |\mathcal{C}_S| \leq M_{\text{sd}}(S, \mathcal{C}) \cdot \log n.$$

Finally, by the Halving algorithm (Littlestone, 1988, Littlestone, 1989), it is known that

$$M_{\text{on-line}}(S, \mathcal{C}) \leq \log |\mathcal{C}_S|.$$

The theorem follows. ■

COROLLARY 3 *For $\mathcal{X}, \mathcal{C}$ and S as above,* $M_{\text{worst}}(S, \mathcal{C}) = O(M_{\text{best}}(S, \mathcal{C}) \cdot \log n)$.

Proof: Combine Theorem 5 with Lemma 1. ■

5. $M_{\text{worst}}(S, \mathcal{C})$ vs. $M_{\text{on-line}}(S, \mathcal{C})$

In this section we further discuss the question of how much can a learner benefit from knowing the learning sequence in advance. In the terminology of our model this is the issue of determining the possible values of the gap between $M_{\text{on-line}}(S, \mathcal{C})$ and $M_{\text{worst}}(S, \mathcal{C})$. We show (in Section 5.2) that if one of these two measures is non-constant then so is the other. Quantitatively, if the on-line algorithm makes k mistakes, then any off-line algorithm makes $\Omega(\sqrt{\log k})$ mistakes on σ. For the special cases where $M_{\text{worst}}(S, \mathcal{C})$ is either 1 or 2, we prove (in Section 5.1) that $M_{\text{on-line}}(S, \mathcal{C})$ is at most 1 or 3 (respectively).

5.1. Simple Algorithms

In this section we present two simple on-line algorithms, $E1$ and $E2$, for the case that the off-line algorithm is bounded by one and two mistakes (respectively) for any sequence.

Let S be a set of elements of $\mathcal{X}$. If for every sequence σ, which is a permutation of S, the off-line learning algorithm makes at most one mistake, then we show that there is an on-line algorithm $E1$ that makes at most one mistake on S, without knowing the actual order in advance. The algorithm $E1$ uses the guaranteed off-line algorithm $\mathcal{A}$ and works as follows:

- Given an element $x \in S$, choose any sequence σ that starts with x, and predict according to $\mathcal{A}$'s prediction on σ, i.e. $\mathcal{A}[\sigma]$. If a mistake is made on x, then $\mathcal{A}[\sigma]$ made a mistake and it will not make any more mistakes on this sequence σ. Hence, we can use $\mathcal{A}[\sigma](c_t(x))$ to get the true values for all the elements of the sequence (where by $\mathcal{A}[\sigma](c_t(x))$ we denote the predictions that $\mathcal{A}$ makes on the sequence σ after getting the value $c_t(x)$). In other words, for any $y \in S$ there is a unique value that is consistent with the value $c_t(x) \neq \mathcal{A}[\sigma]$ (otherwise $\mathcal{A}[\sigma]$ can make another mistake). Therefore, $E1$ will make at most one mistake.

In the case that for any sequence the off-line learning algorithm makes at most two mistakes, we present an on-line algorithm $E2$ that makes at most three mistakes (which is optimal due to Claim 3 below).

Call an element x *bivalent* with respect to y if there exist sequences σ_0 and σ_1 that both start with xy and for σ_0 the on-line algorithm predicts "$c_t(x) = 0$" and for σ_1 the on-line algorithm predicts "$c_t(x) = 1$" (i.e., $\mathcal{A}[\sigma_0] = 0$ and $\mathcal{A}[\sigma_1] = 1$). Otherwise x is *univalent* with respect to y (we say that x is 1-univalent with respect to y if the prediction is always 1 and 0-univalent if the prediction is always 0). Our on-line procedure $E2$, on input x, works as follows.

- *So far we made no mistakes:*
  If there is no y such that x is 1-univalent with respect to y, predict "$c_t(x) = 0$". Else, predict "$c_t(x) = 1$".

- *So far we made one mistake on a positive w:*
  If we made such a mistake then we predicted "$c_t(w) = 0$", which implies that there is no y such that w is 1-univalent with respect to y. In particular, with respect to x, w is either 0-univalent or bivalent. In both cases there is a sequence $\sigma = wx\sigma'$ such that $\mathcal{A}[\sigma]$ predicts "$c_t(w) = 0$" and makes a mistake. Use $b = \mathcal{A}[\sigma](1)$ as the prediction on x (where again, $\mathcal{A}[\sigma](1)$ denotes the prediction that $\mathcal{A}$ makes on sequence σ after getting the value $c_t(w) = 1$). In case of another mistake, we have a sequence on which we already made two mistakes so it will not make any more mistakes. Namely, we can use $\mathcal{A}[\sigma](1, \bar{b})$ to get the value for all elements in S.

- *So far we made one mistake on a negative w:*
  If w is either 1-univalent with respect to x or bivalent with respect to x then this is similar to the previous case. The difficulty is that this time there is also a possibility that w is 0-univalent with respect to x. However, in this case, if we made a mistake this means that we predicted "$c_t(w) = 1$", which implies that there exists a y such that w is 1-univalent with respect to y. Consider a sequence $\sigma = wyx\sigma'$. By the definition of y, $\mathcal{A}[\sigma]$ predicts "$c_t(w) = 1$" and therefore makes its first mistake on w. Denote by $b = \mathcal{A}[\sigma](0)$ the prediction on y. If this is wrong again then all the other elements of the sequence are uniquely determined. Namely, there is a unique function f_1 that is consistent with $c_t(w) = 1, c_t(y) = \bar{b}$. If, on the other hand, b is indeed the true value of y, we denote by $c = \mathcal{A}[\sigma](0, b)$ its prediction on x. Again, if this is wrong, we have a unique function f_2 which is consistent with $c_t(w) = 1, c_t(y) = b, c_t(x) = \bar{c}$. Therefore, we predict c on x. In case we made a mistake (this is our second mistake) we know for sure that the only possible functions are f_1 and f_2 (in fact, if we are lucky then $f_1(x) = c$ and we are done). To know which of the two functions is the target we will need to make (at most) one more mistake (3 in total).

5.2. A General Bound

In this section we further discuss the gap between the measures $M_{\text{on-line}}(S, \mathcal{C})$ and $M_{\text{worst}}(S, \mathcal{C})$. We show that if the on-line makes k mistakes, then any off-line algorithm makes $\Omega(\sqrt{\log k})$ mistakes on σ. The proof makes use of the properties proved in Section 3.1 and the characterizations of both the on-line and the off-line mistake bounds as ranks of trees, proved in Section 4. More precisely, we will take a tree in $T_S^{\mathcal{C}}$ with maximum rank (this rank by Theorem 3 exactly characterizes the number of mistakes made by the on-line algorithm) and use it to construct a tree with rank which is "not too small" and such that the nodes at each level are labeled by the same element of S. Such a tree is of the form T_σ, for some sequence σ.

LEMMA 5 *Given a complete labeled binary tree of depth k, $(T, F) \in \mathcal{T}_S^{\mathcal{C}}$, there is a sequence, σ, of elements of S, such that the tree $T_\sigma^{\mathcal{C}}$ has rank $\Omega(\sqrt{\log k})$.*

Proof: We will construct an appropriate sequence σ in phases. At phase i (starting with $i = 0$) we add (at most 2^i) new elements to σ, so that the rank of $T_\sigma^{\mathcal{C}}$ is increased by at least one (hence, at the beginning of the ith phase the rank is at least i). At the beginning of the ith phase, we have a collection of 2^i subtrees of T, each is of rank at least $k/2^{O(i^2)} \geq 1$ (in particular, at the beginning of phase 0 there is a single subtree, T itself, whose rank is k). Each of these subtrees is consistent with one of the leaves of the tree $T_\sigma^{\mathcal{C}}$ we already built in previous phases (i.e., the subtree is consistent with some assignment to the elements included in σ in all previous phases). Moreover, the corresponding 2^i leaves induce a complete binary subtree of depth i.

In the ith phase, we consider each of the 2^i subtrees in our collection. From each such subtree T' we add to the sequence σ, an element r such that the rank of the subtree of T' rooted at r is rank(T') and the rank in each of the subtrees T_L and T_R corresponding to the sons of r is rank$(T') - 1$. We remark that the order in which we treat the subtrees within the ith phase is arbitrary and that if some of the elements r already appear in σ we do not need to add them again. After adding all the $\ell_i \leq 2i$ new elements of σ we examine again the trees T_L and T_R corresponding to each subtree T'. For each of them the other $\ell_i - 1$ elements partitions the leaves of the tree into $2^{\ell_i - 1}$ groups according to the possible values for the other $\ell_i - 1$ elements. Hence, by Lemma 4, there exists subtrees T_L' and T_R' of T_L and T_r respectively which have rank at least $\left\lfloor \frac{\text{rank}(T')-1}{2^{\ell_i - 1}} \right\rfloor \geq \left\lfloor \frac{\text{rank}(T')}{2^{\ell_i}} \right\rfloor$ and each of them is consistent with one of the leaves of the extended $T_\sigma^{\mathcal{C}}$. The 2^{i+1} subtrees that we get in this way form the collection of trees for phase $i + 1$. Finally note that by the choice of elements r added to σ, we now get in $T_\sigma^{\mathcal{C}}$ a complete binary subtree of depth $i + 1$.

If before the ith phase the rank of the subtrees in our collection is at least k_i then after the ith phase the rank of the subtrees in our collection is at least $k_i/2^i$. Hence, a simple induction implies that

$$k_i \geq \frac{k}{\Pi_{j=1}^{i} 2^j} = \frac{k}{2^{O(i^2)}}.$$

Therefore, we can repeat this process for $\Omega(\sqrt{\log k})$ phases hence obtaining a tree $T_\sigma^{\mathcal{C}}$ of rank $\Omega(\sqrt{\log k})$. ■

THEOREM 6 *Let $\mathcal{C}$ be a concept class, $\mathcal{X}$ an instance space and $S \subseteq \mathcal{X}$ the set of elements. Then $M_{\text{worst}}(S, \mathcal{C}) = \Omega(\sqrt{\log M_{\text{on-line}}(S, \mathcal{C})})$.*

Proof: Assume that $M_{\text{on-line}}(S, \mathcal{C}) = k$. By Theorem 3, there is a rank k tree in $\mathcal{T}_S^{\mathcal{C}}$, and by Lemma 3 it contains a complete binary subtree T of depth k. By Lemma 5, there is a sequence σ for which the tree $T_\sigma^{\mathcal{C}}$ has rank $\Omega(\sqrt{\log k})$. Hence, by Theorem 2, $M(\sigma, \mathcal{C}) \geq \sqrt{\log k}$. ■

A major open problem is what is the exact relationship between the on-line and the off-line mistake bounds. The largest gap we could show is a multiplicative factor of $3/2$.

CLAIM 3 *For all $k \geq 1$, there exist $\mathcal{X}, \mathcal{C}$ and $S \subseteq \mathcal{X}$, for which $M_{\text{on-line}}(S, \mathcal{C}) = 3k$ while $M_{\text{worst}}(S, \mathcal{C}) = 2k$.*

Proof: We first give an example for the case $k = 1$. Let $\mathcal{X}_1$ be a space of 4 elements, $S = \mathcal{X}_1$, and $\mathcal{C}_1$ be the following 8 functions on the 4 elements: $\{$0000, 0011, 0010, 0111, 1000, 1010, 1100, 1111$\}$. It can be verified (by inspection) that $M_{\text{on-line}}(S, \mathcal{C}) = 3$ while $M_{\text{worst}}(S, \mathcal{C}) = 2$.

For a general k, we just take k independent copies of $\mathcal{X}_1$ and $\mathcal{C}_1$. That is, let $\mathcal{X}_k$ be a space of $4k$ elements partitioned into k sets of 4 elements. Let $\mathcal{C}_k$ be the 8^k functions obtained by applying one of the 8 functions in $\mathcal{C}_1$ to each of the k sets of elements. Let $S = \mathcal{X}_k$. Due to the independence of the k functions, it follows that $M_{\text{on-line}}(S, \mathcal{C}) = 3k$ while $M_{\text{worst}}(S, \mathcal{C}) = 2k$. ■

6. Discussion

In this work we analyze the effect of having various degrees of knowledge on the order of elements in the mistake bound model of learning on the performance (i.e., the number of mistakes) of the learner. We remark that in our setting the learner is deterministic. The corresponding questions in the case of *randomized* learners remain for future research.

We can also analyze quantitatively the advantage that an online algorithm may gain from knowing just the *set* of elements, S, in advance (without knowing the order, σ, of their presentation). That is, we wish to compare the situation where the online algorithm knows nothing a-priori about the sequence (other than that it consists of elements of $\mathcal{X}$) and the case that the algorithm knows the set S from which the elements of the sequence are taken (but has no additional information as for their order). The following example shows that the knowledge of S gives an advantage to the learning algorithm:

Example: Consider the intervals concept class of Example 4.1, with the instance space $\mathcal{X}$ restricted to $\{\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n}{n}\}$. As proven, $M_{\text{on-line}}(\mathcal{X}, \mathcal{C}) = \log(n+1)$. On the other hand, for every set S of size ℓ, we showed that $M_{\text{on-line}}(S, \mathcal{C}) = \log(\ell + 1)$. Therefore, if S is small compared to $\mathcal{X}$ (i.e., ℓ is small compared to n) the number of mistakes can be significantly improved by the knowledge of S. □

Acknowledgments

We wish to thank Andris Ambainis, Moti Frances and Nati Linial for helpful discussions. In particular a discussion with Andris led to an improvement in Lemma 5 over a previous version of this paper.

Notes

1. A related result by (Blum, 1994) implies that if efficiency constraints are imposed on the model, then there are cases in which some orders are "easy" and others are computationally "hard".
2. Again, by the Halving algorithm, $M_{\text{on-line}}(S, \mathcal{C})$ and therefore also $M_{\text{best}}(S, \mathcal{C})$ are $O(\log n)$.

References

D. Angluin, (1989). "Equivalence Queries and Approximate Fingerprints", *Proc. of 2nd COLT* pp. 134-145.

A. Blum, (1994). "Separating Distribution-Free and Mistake-Bound Learning Models over the Boolean Domain", *SIAM Journal on Computing*, Vol. 23, No. 4, pp. 373-386. (Also, Proceedings of *FOCS90*, pp. 211-218, 1990.)

A. Blum, (1992a). "Learning Boolean Functions in an Infinite Attribute Space", *Machine Learning*, Vol. 9. (Also, Proceedings of *STOC90*, pp. 64–72, 1990.)

A. Blum, (1992b). "Rank-r Decision Trees are a Subclass of r-Decision Lists", *Information Processing Letters*, Vol. 42, pp. 183–185.

N. Cesa-Bianchi, Y. Freund, D. P. Helmbold, D. Haussler, R. E. Schapire, & M. K. Warmuth, (1993). "How to Use Expert Advice", Proceedings of *STOC93*, pp. 382–391.

Z. Chen, & W. Maass, (1994). "On-line Learning of Rectangles", *Machine Learning*, Vol. 17, pp. 23–50. (Also, Proceedings of *COLT92*, pp. 16–27, 1992.)

T. H. Cormen, C. E. Leiserson, & R. L. Rivest, (1990). *Introduction to Algorithms*. MIT Press.

A. Ehrenfeucht & D. Haussler, (1989). "Learning Decision Trees from Random Examples", *Information and Computation*, Vol. 82, pp. 231-246.

M. Feder, N. Merhav, & M. Gutman, (1992). "Universal Prediction of Individual Sequences", *IEEE Trans. on Information Theory*, IT-38, No. 4, pp. 1258-1270.

S. A. Goldman, R. L. Rivest, & R. E. Schapire, (1993). "Learning Binary Relations and Total Orders", *SIAM Journal on Computing*, Vol. 22, No. 5, pp. 1006–1034.

S. A. Goldman, & R. H. Sloan, (1994). "The Power of Self-Directed Learning", *Machine Learning*, Vol. 14, pp. 271–294.

D.P. Helmbold, N. Littlestone, & P.M. Long, (1992). "Apple Tasting and Nearly One-Sided Learning", Proceedings of *FOCS92*, pp. 493-502.

N. Littlestone, (1988). "Learning when Irrelevant Attributes Abound: A New Linear-Threshold Algorithm", *Machine Learning*, Vol. 2, pp. 285–318.

N. Littlestone, (1989). "Mistake Bounds and Logarithmic Linear-Threshold Learning Algorithms", PhD thesis, U.C. Santa Cruz.

N. Littlestone & M. K. Warmuth, (1994). "The Weighted Majority Algorithm", *Information and Computation*, Vol. 108, pp. 212-261. (Also, Proceedings of *FOCS89*, pp. 256-261, 1989.)

W. Maass, (1991). "On-line Learning with an Oblivious Environment and the Power of Randomization", Proceedings of *COLT91*, pp. 167–175.

N. Merhav & M. Feder, (1993). "Universal Sequential Decision Schemes from Individual Sequences", *IEEE Trans. on Information Theory*, IT-39, pp. 1280-1292.

S. Porat & J. Feldman, (1991). "Learning Automata from Ordered Examples", *Machine Learning*, Vol. 7, pp. 109-138. (Also, *Proc. of 1st COLT* pp. 386-396, 1988.)

N. Sauer, (1972). "On the Density of Families of Sets", *Journal of Combinatorial Theory (A)*, Vol. 13, pp. 145–147,.

Received March 29, 1996
Accepted January 27, 1997
Final Manuscript July 3, 1997