Online Learning versus Offline Learning

Shai Ben-David[1] and Eyal Kushilevitz[2] and Yishay Mansour[3]

[1] Computer Science Dept., Technion, Israel.[†]
[2] Computer Science Dept., Technion, Israel.[‡]
[3] Computer Science Dept., Tel-Aviv University, Israel.[§]

Abstract. We present an off-line variant of the mistake-bound model of learning. Just like in the well studied on-line model, a learner in the off-line model has to learn an unknown concept from a sequence of elements of the instance space on which he makes "guess and test" trials. In both 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.

We give a combinatorial characterization of the number of mistakes in the off-line model. We apply this characterization to solve several natural questions that arise for the new model. First, we compare the mistake bounds of an off-line learner to those of a learner learning the same concept classes in the on-line scenario. We show that the number of mistakes in the on-line learning is at most a $\log n$ factor more than the off-line learning, where n is the length of the sequence. In addition, we show that if there is an off-line algorithm that does not make more than a constant number of mistakes for each sequence then there is an on-line algorithm that also does not make more than a constant number of mistakes.

The second issue we address is the effect of the ordering of the elements on the number of mistakes of an off-line learner. It turns out that there are sequences on which an off-line learner can guarantee at most one mistake, yet a permutation of the same sequence forces him to err on many elements. We prove, however, that the gap, between the off-line mistake bounds on permutations of the same sequence of n-many elements, cannot be larger than a multiplicative factor of $\log n$, and we present examples that obtain such a gap.

[†] shai@cs.technion.ac.il
[‡] eyalk@cs.technion.ac.il
[§] 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.

1 Introduction

Since defined by Littlestone [L88, L89], the *on-line* model of learning has attracted a considerable amount of attention (e.g., [L88, L89, LW89, B90a, B90b, M91, CM92, HLL92]). In this model the learner has to make predictions on the next instance based on the previous instances that it has already saw and their "labels". The quantity that we would like to minimize is the number of mistakes the learner does in this process.

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 he would be challenged on in the future. While the first uncertainty is common to almost any learning model, the second is indicative to the on-line learning model.

The main 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 affect that this uncertainty has on the on-line learning model. To formalize this abstract question, we start with the following variant of the on-line model. The instances to the learner are chosen from a set which is known to the learner, but the order in which it receives the instances is unknown to him. We compare this on-line learner to an *off-line* learner, one that 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 the *best sequence* (the sequence in which the optimal learner, knowing the sequence, makes the fewest mistakes, denoted by m_{best}) and the *worst sequence* (the sequence in which the optimal learner, knowing the sequence, makes the most mistakes, denoted by m_{worst}). In addition, we denote the worst case number of mistakes made by the on-line learner by $m_{on-line}$.

One way to view our setting is through the model of "experts" [CFHHSW93, FMG92, MF93]. For each sequence there is an expert, and each expert makes at most m_{worst} mistakes. The on-line learner does not know the sequence in advance, so the question is how close can it get to the best expert, which makes m_{best} mistakes. The problem is that the number of experts is overwhelming; there are $n!$ experts. Therefore, previous results about experts do not apply here.

The abovementioned quantities have another intuitive interpretation. The cost $m_{on-line}$ is the number of mistakes made if predicting on the elements as they arrive. For m_{worst} we are allowed to have a look ahead on the elements that we need to process, but are not allowed to change the order. The cost of m_{best} is the cost when we can rearrange the elements in a way that is most favorable to the learner. An illustrative example from the stock market may be that in the "on-line" scenario we are given a stock and have to predict (on the fly) if it will rise or fall; in the "worst" off-line scenario, we are given a sequence of stocks and have to predict them in the given order (e.g., the order they appear on the screen); while in the "best" off-line scenario we can reorder the stocks as we like (e.g., choosing the leading ones first).

The following example would be helpful in understanding both the notions and our results. Consider the class of functions which are a suffix of the interval $[0,1]$. Given n points there are only $n+1$ possible concepts, and therefore the *Halving* algorithm [L88, L89] 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 of the sequence). We generalize this example, and show that for any concept class, the exact number of mistakes is the rank of a tree (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 [L88, L89], provides us a powerful combinatorial characterization.

We study the relationships between m_{best}, m_{worst} and $m_{on-line}$. Clearly $m_{best} \leq m_{worst} \leq m_{on-line}$. We show that $m_{on-line} = O(m_{best} \log n)$, namely that the ratio in the worst case is $\log n$. This also implies that the worst ratio between the number of mistakes for the best sequence and the worst sequence is $O(\log n)$.[7] We also show that $m_{worst} = \Omega(\log\log m_{on-line})$, which implies that either both are constant or both are non-constant. (Even when they are not constant, the worst case ratio between them is bounded by $\log n$.) Finally we have an example in which $m_{worst} = 2$ and $m_{on-line} = 3$, 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.

A related model is the *self-directed learning* studied in [GRS93, GS94]. In this model the learner is allowed to choose the sequence of instances. In a sense this is a model in which the leaner can ask membership-queries but is charged according to the number of mistakes. The number of mistakes made in self-directed learning is related to our measure m_{best}, however there are few differences. First, in our model the learner is given a set of instances, and then considers the best ordering of them, while in [GS94] the best sequence is out of the entire universe. Second, the self-directed learner is "adaptive"; he may generate the sequence of elements one-by-one based on previous replies, while our model assumes a "static" sequence. (One of our results shows that if there is an adaptive sequence that makes a constant number of errors then there is a static one that makes a constant number of errors.) Goldman and Sloan [GS94] address the connection between the complexity of learning a class in their model to the VC-dimension of the class. Their results can be easily generalized to our setting.

[7] A related result by [B90a] 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".

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 characterize the off-line complexity (for completeness, we present a characterization of the same nature, based on [L88, L89], for the on-line complexity in Section 4.1) and we use these characterizations to obtain some basic results. In Section 5 we use these characterizations to study the gap between the on-line complexity and off-line complexity. The results regarding the VC-dimension are omitted from this version of the paper.

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 [L88, L89].

Let $\mathcal{X}$ be any set, and let $\mathcal{C}$ be a collection of boolean functions defined over $\mathcal{X}$ (i.e. $\mathcal{C} \subseteq \{0,1\}^{\mathcal{X}}$). 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 and in response he 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 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 $M(\mathcal{A}[S]) \stackrel{\triangle}{=} \max_{\sigma, c_t} M(\mathcal{A}[S], \sigma, c_t)$. Finally, let

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

The original definitions of [L88, L89] 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 σ and not only on 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

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

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 corresponding sequence.

Lemma 1. *For any $\mathcal{X}, \mathcal{C}$, and a finite $S \subseteq \mathcal{X}$, $M_{\text{worst}}(S, \mathcal{C}) \leq M_{\text{on-line}}(S, \mathcal{C})$.*

Proof. The algorithm that gets σ as an input can always "ignore" the order and use the on-line algorithm for S. □

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 [L89]. 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 he 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 $(h_i \Delta c_t) \cap S = \emptyset$.
- Let F denote the function that chooses the counter examples 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 $EQ(S, \mathcal{C}) \stackrel{\triangle}{=} \min_{\mathcal{A}} \max_{F, c_t \in \mathcal{C}} EQ(\mathcal{A}[S], F, c_t)$.

Note that the original definitions of Angluin [A89] are obtained by considering $S = \mathcal{X}$. The following is a well known (and easy to prove) fact:

Claim 2. *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 fixed 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 counter examples. Given a hypothesis h, the counterexample, x, is the minimal element of $(h \Delta c_t) \cap S$, according to the ordering σ. Let $EQ(\mathcal{A}[\sigma], c_t) \triangleq EQ(\mathcal{A}[\sigma], F_\sigma, c_t)$.
– Let $EQ(\sigma, \mathcal{C}) \triangleq \min_{\mathcal{A}} \max_{c_t \in \mathcal{C}} EQ(\mathcal{A}[\sigma], c_t)$.
– Let $EQ_{\text{best}}(S, \mathcal{C}) \triangleq \min_\sigma EQ(\sigma, \mathcal{C})$, and let $EQ_{\text{worst}}(S, \mathcal{C}) \triangleq \max_\sigma EQ(\sigma, \mathcal{C})$.

This variant of the equivalence query model in which the minimal counterexample is provided to the algorithm is studied, e.g., in [PF88].

Claim 3. *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, h_{k_i}, of the EQ algorithm 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 it 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 4. *For every $\mathcal{X}, \mathcal{C}$ and $S \subseteq \mathcal{X}$ as above, (1) $EQ_{\text{best}}(S, \mathcal{C}) = M_{\text{best}}(S, \mathcal{C})$; and (2) $EQ_{\text{worst}}(S, \mathcal{C}) = M_{\text{worst}}(S, \mathcal{C})$.*

3 Labeled Trees

The 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 Ranks of Trees

In this subsection we define the notion of the *rank* of a binary tree (see, e.g., [CLR90, EH89, B92]), 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 $\text{rank}(T) = -1$. Otherwise, let T_L be its left subtree and T_R be its right subtree. Then,

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

For example, the rank of a leaf is 0.

Lemma 5. *A depth d rank r tree has at most $\binom{d}{\le r}$ leaves, where $\binom{d}{\le r}$ denotes $\sum_{i=0}^{r} \binom{d}{i}$.*

Proof. By induction on d and r. If $d = 1$ clearly there is exactly one leaf. Also if $r = 0$ there is exactly one leaf (if there were two or more, then their least common ancestor is of rank 1). 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

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

as desired. □

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

Lemma 6. *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, its children have rank $i-1$, hence each of them has a marked ancestor 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 7. *Let T be a complete binary tree. Let L_0 and L_1 be subsets of the leaves such that $L_0 \cup L_1$ includes all the leaves. Define T_0 to be the subtree induced by the leaves in L_0 (i.e., T_0 is the tree of all the nodes of T that have a member of L_0 below them), similarly, let T_1 be the subtree induced by L_1. Then,*

$$\text{rank}(T) \le 2 \max\{\text{rank}(T_0), \text{rank}(T_1)\}.$$

Proof. Let k be the depth of T. Consider the subtrees of T whose roots are the nodes of T in level $k/2$ (there are $2^{k/2}$ such subtrees). If one of them is fully contained in either T_0 or T_1, then the rank of this T_i is at least $k/2$, and we are done. Otherwise, each of the nodes in the $k/2's$ level of T contains leaves from both L_0 and L_1, in this case, both T_0 and T_1 contain the first $k/2$ levels of T so the rank of each of them is at least $k/2$. □

Let us just mention that the above lower bound, on the rank of the induced subtrees, is the best possible. To see this, 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 nodes of) T into $\mathcal{X}$. Furthermore, we impose the following restriction on F:

$$\forall s \neq t \in T, \text{ if } s \text{ ancestor of } t \text{ then } F(s) \neq F(t).$$

  I.e., we require that F is one to one on branches of T.
- A branch $(s_1, \ldots, s_n)$ is *consistent with* $\mathcal{C}$ if there exists some concept, $c \in \mathcal{C}$, such that, for all $1 \leq i < n$, if s_{i+1} is a left son of s_i then $F(s_i) \in c$, and if s_{i+1} is a right son of s_i then $F(s_i) \notin c$. Note that the values of F on leaves of T do not affect consistency, and therefore, for the purposes of this work, may be ignored.
- An $\mathcal{X}$-labeled tree is $\mathcal{C}$-*sound* if each of its branches is consistent with $\mathcal{C}$.
- Let $\mathcal{T}_S^{\mathcal{C}}$ denote the set of all S-trees that are $\mathcal{C}$-sound.
- For a sequence $\sigma = (s_1, \ldots, s_n)$ of elements of $\mathcal{X}$, let $T_\sigma^{\mathcal{C}}$ denote the labeled tree that is defined by starting with a complete binary tree, T', with $n+1$ levels, setting $F(t) = s_k$ for every node t on the k'th level of T', and restricting the tree to the branches that are consistent with $\mathcal{C}$.

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. We can therefore conclude that, for any class $\mathcal{C}$,
(We shall usually omit the superscript, $\mathcal{C}$, when it is clear from the context.)

4 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 given any sequence σ, $M(\mathcal{A}[\sigma]) = M(\sigma, \mathcal{C})$. Before doing that, we briefly discuss the measure $M_{\text{on-line}}(S, \mathcal{C})$.

Given an instance space $\mathcal{X}$, a concept class $\mathcal{C}$ and a set S we can 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 $\mathcal{C}_S$). In particular, using the Halving algorithm [L88, L89] we get,

Theorem 8. *For all $\mathcal{X}, \mathcal{C}$ and S as above $M_{\text{on-line}}(S, \mathcal{C}) \leq \log |\mathcal{C}_S|$.*

Example 1. 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 $0 \leq a \leq 1$. By Theorem 8, 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 the off-line 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 $M_{\text{best}}(S, \mathcal{C}) = 1$ we take $\sigma_{\text{best}} = (\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n}{n})$. 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

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

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

We now wish to compute $M(\sigma, \mathcal{C})$ for any sequence σ. For doing this we consider the *search tree* corresponds to the sequence $\sigma = s_1 s_2 \ldots s_n$. Namely, the tree constructed by the sequence of operations $insert(s_1), insert(s_2), \ldots, insert(s_n)$. Note that this is exactly the tree T_σ, defined in the previous section. We claim that $M(\sigma, \mathcal{C}) = \text{rank}(T_\sigma)$. To predict on s_1, we consider the tree T_σ. Its root is s_1 and we denote by T_L its left subtree and by T_R its right subtree. If $\text{rank}(T_L) > \text{rank}(T_R)$ we predict "1", if $\text{rank}(T_L) < \text{rank}(T_R)$ we predict "0", and if $\text{rank}(T_L) = \text{rank}(T_R)$ we can predict arbitrarily. If the prediction is 1 and the true value is 0 we already know the value of each element in T_L (as they are all smaller than s_1). Therefore, the only elements which we are not sure about, in this case, are those in T_R. (If the prediction of 1 is correct we get "for free" the values of all elements in T_R and remain only with the elements in T_L undetermined). Similarly when the prediction is 0. Note that in this case, by the definition of rank, both $\text{rank}(T_L)$ and $\text{rank}(T_R)$ are smaller than $\text{rank}(T_\sigma)$. In subsequent steps if we are asked about elements which were already determined we answer with their value, and if not we use the same strategy with respect to the remained subtree. A crucial point is that since this is a search tree, then the "higher" vertices correspond to points that appear earlier in the sequence. To conclude, at each step where we made a mistake, the rank decreased by (at least) 1 so no more than $\text{rank}(T_\sigma)$ mistakes are made. Note that the algorithm is also efficient. An adversary argument shows that this is the best possible. Again, this will be formalized by Theorem 9 below.

The next theorem generalizes the intuition given by the above example to any concept class. In particular, it 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 [L88, L89] for the *on-line* case (see section 4.1 below).

Theorem 9. *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 $\text{rank}(T_L) < \text{rank}(T_R)$ we predict "1", otherwise ($\text{rank}(T_L) = \text{rank}(T_R)$) we can predict arbitrarily. Again, recall that in this case, 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 we use for the prediction a subtree of T_σ which is consistent with all the values we have seen so far. To conclude, at each step where we 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 $\mathcal{A}$ does at least $\text{rank}(T_\sigma)$ mistakes. The adversary constructs for himself 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 he 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. □

Remark. It is worth noting that, by Sauer Lemma [S72], 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.

A natural question is what is the maximum ratio between $M_{\text{worst}}(S, \mathcal{C})$ and $M_{\text{best}}(S, \mathcal{C})$. In Example 1 the best is 1 and the worst is about $\log n$, which can be easily generalized to k versus $\Theta(k \log n)$. The following theorem shows that the gap between $M_{\text{best}}(S, \mathcal{C})$ and the optimal on-line cost cannot exceed $O(\log n)$, which implies a similar bound for the gap between σ_{best} and σ_{worst}. By Example 1 the bound is tight, i.e., there are cases which achieve this gap.

Theorem 10. *For $\mathcal{X}, \mathcal{C}$ and S as above, $M_{\text{on-line}}(S, \mathcal{C}) = O(M_{\text{best}}(S, \mathcal{C}) \cdot \log n)$.*

Proof. Consider the tree $T_{\sigma_{\text{best}}}$. Its depth is $n+1$ and its rank, by Theorem 9, is $m \triangleq M_{\text{best}}(S, \mathcal{C})$. By Lemma 5, this tree contains at most $\binom{n+1}{\leq m}$ leaves. That is, $|\mathcal{C}_S| \leq \binom{n+1}{\leq m}$. By Theorem 8, $M_{\text{on-line}}(S, \mathcal{C}) \leq \log \binom{n+1}{\leq m} = O(m \cdot \log n)$, as needed. □

Corollary 11. *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 10 with Lemma 1. □

4.1 Characterizing the On-line

It is useful, for the study of the relations between the on-line and the off-line, to have a combinatorial characterization of $M_{\text{on-line}}(S, \mathcal{C})$ as well. Such a characterization was given by Littlestone [L88, L89]. We reformulate this characterization in terms of ranks of trees. The proof remains similar to the one given by Littlestone [L88, L89] and we provide it here for completeness.

Theorem 12. *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 9. 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 decreased by 1 the prediction algorithm made 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 9. 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 predict according to the larger one. The crucial point is that at least one of the those 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. □

4.2 The Effect of Knowing S in Advance

We are now in a position to 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).

Theorem 12 above states the the number of mistakes an optimal online algorithm makes learning a class $\mathcal{C}$ is $\max\left\{\text{rank}(T) \ : \ T \in \mathcal{T}_{\mathcal{X}}^{\mathcal{C}}\right\}$. Furthermore, the proof of that theorem shows that any online algorithm may be forced to err whenever the rank of the relevant subtree (of the tree giving rise to the maximal rank) decreases. By Lemma 6, there exists a tree of maximal rank which is a complete binary tree. It follows that, for every class $\mathcal{C}$, an online algorithm that does not know the set of elements in advance may be forced to err on every single element, in a sequence of elements of length $\max\left\{\text{rank}(T) \ : \ T \in \mathcal{T}_{\mathcal{X}}^{\mathcal{C}}\right\}$.

Now, when the set of elements, S, is known in advance, the maximal number of mistakes reduces to $\max\{\text{rank}(T) : T \in \mathcal{T}_S^{\mathcal{C}}\}$. Furthermore, by Theorem 8, there exists an algorithm that errs only on a logarithmic fraction of the number of elements in a sequence. Hence, the gap between knowing S and not knowing it may be exponential.

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 makes k mistakes, then any off-line algorithm makes $\Omega(\log\log k)$ 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 any 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. 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 15 below).

Call an element x *bivalent* with respect to y if there exist sequences σ_0 and σ_1 both start with xy and for σ_0 the on-line algorithm predicts $x = 0$ and for σ_1 the on-line algorithm predicts $x = 1$ (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 "$x = 0$". Else, predict "$x = 1$".
- *So far we made one mistake on a positive w:* If we made such a mistake then we predicted $w = 0$, 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 $w = 0$ and makes a mistake. Use $b = \mathcal{A}[\sigma](1)$ as the prediction on x. 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 $w = 1$, 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 $w = 1$ 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 $w = 1, y = \bar{b}$. 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 $w = 1, y = b, x = \bar{c}$. 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 either 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 a non-constant number of mistakes on a set of elements S, then there exists a sequence σ (an ordering of S) for which a non-constant number of mistakes must be made even when σ is known in advance. Quantitatively, if the on-line makes k mistakes, then any off-line algorithm makes $\Omega(\log\log k)$ 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 $\mathcal{T}_S^{\mathcal{C}}$ with maximum rank (this rank by Theorem 12 exactly characterizes the number of mistakes made by the on-line algorithm) and construct a subtree of it 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 13. *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 $\Omega(\log\log k)$.*

Proof. We will construct an appropriate sequence σ one element after the other. At each phase we add (2^i) new elements to σ, so that the rank of $T_\sigma^{\mathcal{C}}$ is increased by one. While constructing σ we keep the following invariant: at the beginning of the ith phase, there are 2^i subtrees of T, each is of rank at least $k/2^{2^i}$, and is consistent with some assignment to the elements included in σ in all previous phases.

The ith phase consists of 2^i sub-phases. In each sub-phase, we consider one of the 2^i subtrees, and add its root r to the sequence σ. For any other subtree ST we define L_0 as all the leaves that are consistent with $r = 0$ and L_1 are the ones consistent with $r = 1$. We define ST_0 as the subtree with leaves in L_0 and ST_1 as the subtree with leaves in L_1. We replace ST by the subtree with the larger rank between ST_0 and ST_1. By lemma 7, the rank is at least half the original rank.

If before the ith phase the ranks of the subtrees are k_i after it they are at least $k_i/2^{2^i}$. Since $k_i \geq k/2^{2^i}$, then

$$k_{i+1} \geq \frac{k_i}{2^{2^i}} = \frac{k}{2^{2^i}2^{2^i}} \geq \frac{k}{2^{2^{i+1}}}.$$

Therefore, we can repeat this process for $\Omega(\log\log k)$ phases. □

Theorem 14. *Let $\mathcal{C}$ be a concept class, $\mathcal{X}$ an instance space and $S \subseteq \mathcal{X}$ the set of elements. Then $M_{\text{worst}}(S, \mathcal{C}) = \Omega(\log\log M_{\text{on-line}}(S, \mathcal{C}))$.*

Proof. Assume that $M_{\text{on-line}}(S, \mathcal{C}) = k$. By Theorem 12, there is a rank k tree in $T_{\mathcal{C}_S}$, and by Lemma 7 it contains a complete binary subtree T of depth k. By Lemma 13, there is a subtree T', with rank $\log\log k$ that in each level there is a single element from S. Let σ be the order of the elements of T' (note that on every path the order is the same). The tree of T_σ is an extension of T', therefore, $\text{rank}(T') \leq \text{rank}(T_\sigma)$ and hence by Theorem 9, $M(\sigma, \mathcal{C}) \geq \log\log k$. □

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 through an example that just demonstrates that they are not equal.

Claim 15. *For all $k \geq 2$, there exist $\mathcal{X}, \mathcal{C}$ and $S \subseteq \mathcal{X}$, for which $M_{\text{on-line}}(S, \mathcal{C}) = k + 1$ while $M_{\text{worst}}(S, \mathcal{C}) = k$.*

Proof. We give an example for the case $k = 2$ (the general case can be easily obtained from it). Let $\mathcal{X}$ be a space of 4 elements, $S = \mathcal{X}$, and $\mathcal{C}$ 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 $M_{\text{worst}}(S, \mathcal{C}) = 2$. □

Acknowledgment

We wish to thank Moti Frances and Nati Linial for helpful discussions.

References

[A89] D. Angluin, "Equivalence Queries and Approximate Fingerprints" *Proc. of 2nd COLT* pp. 134-145, 1989.

[B90a] A. Blum, "Separating distribution-free and mistake-bound learning models over the Boolean domain" Proceedings of *FOCS90*, pp. 211-218, 1990.

[B90b] A. Blum, "Learning Boolean Functions in an Infinite Attribute Space" *Machine Learning*, Vol. 9, 1992. (Also, Proceedings of *STOC90*, pp. 64–72, 1990.)

[B92] A. Blum, "Rank-r Decision Trees are a Subclass of r-Decision Lists" *Information Processing Letters*, Vol. 42, pp. 183–185, 1992.

[CFHHSW93] N. Cesa-Bianchi, Y. Freund, D. P. Helmbold, D. Haussler, R. E. Schapire, and M. K. Warmuth "How to use expert advice", Proceedings of *STOC93*, pp. 382–391, 1993.

[CM92] Z. Chen, and W. Maass, "On-line Learning of Rectangles" Proceedings of *COLT92*, pp. 16–27, 1992.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Algorithms.* MIT Press, 1990.

[EH89] A. Ehrenfeucht, and D. Haussler, "Learning Decision Trees from Random Examples" *Information and Computation*, Vol. 82, pp. 231-246, 1989.

[FMG92] M. Feder, N. Merhav, and M. Gutman, "Universal Prediction of Individual Sequences", *IEEE Trans. on Information Theory*, IT-38, No. 4, pp. 1258-1270, 1992.

[GRS93] S. A. Goldman, R. L. Rivest, and R. E. Schapire, "Learning Binary Relations and Total Orders", *SIAM Journal on Computing*, Vol. 22, No. 5, pp. 1006–1034, 1993.

[GS94] S. A. Goldman, and R. H. Sloan, "The Power of Self-Directed Learning", *Machine Learning*, Vol. 14, pp. 271–294, 1994.

[HLL92] D.P. Helmbold, N. Littlestone, and P.M. Long, "Apple tasting and nearly one-sided learning", Proceedings of *FOCS92*, pp. 493-502, 1992.

[L88] N. Littlestone, "Learning when Irrelevant Attributes Abound: A New Linear-Threshold Algorithm" *Machine Learning*, Vol. 2, pp. 285–318, 1988.

[L89] N. Littlestone, "Mistake bounds and logarithmic linear-threshold learning algorithms" PhD thesis, U.C. Santa Cruz, March 1989.

[LW89] N. Littlestone, M. K. Warmuth, "The weighted majority algorithm", Proceedings of *FOCS89*, pp. 256-261, 1989.

[M91] W. Maass, "On-line Learning with an Oblivious Environment and the Power of Randomization" Proceedings of *COLT91*, pp. 167–175, 1991.

[MF93] N. Merhav and M. Feder, "Universal Sequential Decision Schemes from Individual Sequences", *IEEE Trans. on Information Theory*, IT-39, pp. 1280-1292, July 1993.

[PF88] S. Porat, and J. Feldman, "Learning Automata from Ordered Examples", *Proc. of 1st COLT* pp. 386-396, 1988.

[S72] N. Sauer, "On the Density of Families of Sets", *Journal of Combinatorial Theory (A)*, Vol. 13, pp. 145–147, 1972.

This article was processed using the LaTeX macro package with LLNCS style