Abstract

We present an off-line variant of the mistake-bound model of learning. Just like in the well studied on-line model, a student in the off-line model has to learn an unknown concept from a sequence of "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 queries is known, in the off-line model the sequence of queries (i.e., the identity of the queries 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 student to those of a student 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 affect of the ordering of the queries on the number of mistakes of an off-line student. It turns out that there are sequences on which an off-line student can guarantee at most one mistake, yet a permutation of the same sequence forces him to err on many queries. We prove, however, that the gap, between the off-line mistake bounds on permutations of the same sequence of $n$-many queries, cannot be larger than a multiplicative factor of $\log n$, and we present examples that obtain such a gap.
1 Introduction

The on-line model of learning [L88, L89], has attracted a considerable amount of attention (e.g., [L88, L89, IW89, 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 queries, and to understand the affect that this uncertainty has on the model of on-line learning model. In order 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 queries in advance. Since the main difference between the on-line learner and the off-line learner is the uncertainty regarding the order of the instances, this comparison gives insight to the “information” that is in knowing the sequence.

Once we define the off-line cost of a sequence of queries, we can 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 is guaranteed to have at most \( m_{worst} \) 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 quantities \( m_{best} \), \( m_{worst} \) and \( m_{on-line} \) have another intuitive interpretation. The cost \( m_{on-line} \) is the cost of serving the queries as they arrive. For \( m_{worst} \) we are allowed to have a look ahead on the queries 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 queries in a way that is most favorable to the learner. An illustrative example from the stock market may be that in the “online” 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 the worst sequence forces \( m_{worst} = \Theta(\log n) \) mistakes, where \( n \) is the number of points. 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 gives us a powerful combinatorial tool to work with.

Another way to state our goal, is to establish relationships between $m_{\text{best}}$, $m_{\text{worst}}$ and $m_{\text{on-line}}$. Clearly $m_{\text{best}} \leq m_{\text{worst}} \leq m_{\text{on-line}}$. We show that $m_{\text{on-line}} = O(m_{\text{best}} \log n)$, namely that the factor in the worst case is $\log n$. Note that this also implies that the worst ratio between the number of mistakes for the best sequence and the worst sequence is $O(\log n)$.\footnote{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 not learnable at all.} We also show that $m_{\text{worst}} = \Omega(\log \log m_{\text{on-line}})$. This implies that either both are constant or both are non-constant, in any case the worst case ratio between them is at most $\log n$, when they are not constant. Finally we have an example in which $m_{\text{worst}} = 2$ and $m_{\text{on-line}} = 3$, showing that $m_{\text{worst}} \neq m_{\text{on-line}}$. In a few cases we are able to derive better bounds. For the cases that $m_{\text{worst}} = 1$ and $m_{\text{worst}} = 2$ we show simple on-line algorithms that have at most 1 and 3 mistakes, respectively.

The connection to the VC dimension of the class is also addressed. If a class has VC dimension $d$ then there is a sequence that makes $d$ mistakes and no sequence makes more than $d \log n$ mistakes. We are able to show that classes of VC dimension 1, have $m_{\text{best}} = 1$; we also show an example in which VC dimension is $d$ and $m_{\text{best}} = 1$. We leave as an open problem whether $m_{\text{best}}$ can be bounded from above as a function of the VC dimension, independent of $n$.

The rest of this paper is organized as follows: In Section 2.1 we give formal definitions of the model and the measures of performance that are discussed in this paper. In Section 2.2 we give the definition of the \textit{rank} of a tree, and prove some properties of it. In Section 3 we characterize the off-line complexity (we present a characterization of the same nature for the on-line complexity in Section 3.1). In Section 4 we use these characterizations to study the gap between the on-line complexity and off-line complexity. Finally, in Section 5 we study some connections with the VC dimension.

\section{Preliminaries}

\subsection{The Model}

In this section we give the basic definitions that are used in this paper. The general framework is similar to the \textit{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 $\mathcal{X}$. We refer to $\mathcal{X}$ as the \textit{instance space} and to $\mathcal{C}$ as the \textit{concept class}. Let $S$ be a finite subset of $\mathcal{X}$. An \textit{on-line learning algorithm with respect to $S$} is an algorithm $\mathcal{A}$ that is given (in advance) $S$ as an input. Then, the learning process consists of 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 $c_i(s_i)$, where $c_i \in \mathcal{C}$ denotes the \textit{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 goes on until all the elements of $S$ have been presented. Let $\sigma = s_1, s_2, \ldots, s_n$ denote the the order according to which the elements of $S$ are presented to the learning algorithm. We denote by $M(\mathcal{A}[S], \sigma, c_1)$ the number of mistakes made by the algorithm on a sequence $\sigma$, target function $c_1 \in \mathcal{C}$, when he knows $S$ in advance. That is, $M(\mathcal{A}[S], \sigma, c_1) \triangleq \left| \{i : p_i \neq c_1(s_i)\} \right|$. We then define the mistake
lemmadefinitions of [L88, L89] are obtained (at least for finite \( \mathcal{X} \)) by considering \( S = \mathcal{X} \).

An \textit{on-line learning algorithm} is an algorithm \( \mathcal{A} \) that is given (in advance) not only the set \( S \), but also the actual sequence \( \sigma \) as an input. The learning process remain unchanged (except that each prediction \( p_i \) can now depend on \( \sigma \) and not only on \( S \)). We denote by \( M(\mathcal{A}[\sigma], c_t) \) the number of mistakes made by an on-line algorithm, \( \mathcal{A} \), on a sequence \( \sigma \) and a target \( c_t \). That is, \( M(\mathcal{A}[\sigma], c_t) \stackrel{\Delta}{=} |\{i : p_i \neq c_t(s_i)\}| \). We then define \( M(\mathcal{A}[\sigma]) \stackrel{\Delta}{=} \max_{c_t} M(\mathcal{A}[\sigma], c_t) \)

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

Note that the original definitions of [L88, L89] are obtained (at least for finite \( \mathcal{X} \)) by considering \( S = \mathcal{X} \).

For a particular sequence \( \sigma \), we define

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

For a given \( S \) we will be interested in two particular sequences, \( \sigma_{\text{best}} \) and \( \sigma_{\text{worst}} \). The former is a sequence such that for all \( \sigma \), \( M(\sigma_{\text{best}}) \leq M(\sigma) \) (if there is more than one such sequence pick one of them arbitrarily), and similarly the latter is a sequence such that for all \( \sigma \), \( M(\sigma_{\text{worst}}) \geq M(\sigma) \).

**Lemma 1:** For any \( \mathcal{X}, \mathcal{C} \), \( M(\sigma_{\text{worst}}) \leq M_{\text{on-line}}(S, \mathcal{C}) \).

**Proof:** The algorithm that gets \( \sigma \) as an input can always ignore the order and use the on-line algorithm for \( S \). \hfill \Box

### 2.2 Rank of Trees

In this subsection we define the notion of the \textit{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 2:** A depth \( d \) rank \( r \) tree has at most \( d^r \) leaves.

**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 (as 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

\[
(d - 1)^r + (d - 1)^{r - 1} = (d - 1)^{r - 1} \cdot ((d - 1) + 1) = (d - 1)^{r - 1} \cdot d < d^r
\]
as desired. \hfill \Box
Lemma 3: The rank of a binary tree is at least \( k \) iff it has a subtree which is a complete binary tree of depth \( k \).

Proof: Mark in \( T \) the nodes in which the rank increases. Those are the nodes of \( T' \). From a marked node with rank \( i \), its children have rank \( i - 1 \), hence each of them has an ancestor marked with rank \( i - 1 \). Therefore \( T' \) is a complete binary tree.

For the other direction, first note that the rank of a tree is at least the rank of any of its subtrees, and then that a complete binary tree of depth \( k \) has rank \( k \).

Lemma 4: 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,

\[
\text{rank}(T) \leq 2 \max\{\text{rank}(T_0), \text{rank}(T_1)\}.
\]

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 best possible. To see this, we take \( T \) to be a complete binary tree. Each leaf corresponds to a path from the root to this leaf. We 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 \( \text{rank}(T_0) = \text{rank}(T_1) = k/2 \).

3 Characterizing \( M(\sigma) \) Using the Rank

The main goal of this section is to characterize the measure \( M(\sigma) \). As a by-product, we present an optimal prediction algorithm. I.e., an algorithms, \( A \), such that given any sequence \( \sigma \), \( M(A[\sigma]) = M(\sigma) \). Before doing that, we briefly discuss the measure \( M_{\text{on-line}}(S, C) \).

Given an instance space \( X \), a concept class \( C \) and a set \( S \) we can define \( C_S \) to be the projection of the functions in \( C \) on the set \( S \) (note that several functions in \( C \) may collide into a single function in \( C_S \)). In particular, using the Halving algorithm we get,

Theorem 1: For all \( X, C \) and \( S \) as above \( M_{\text{on-line}}(S, C) \leq \log |C_S| \).

Example 1: Let \( X \) 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 \( C \triangleq \{f_a : 0 \leq a \leq 1\} \). In other words, the concept class \( C \) consists of all intervals of the form \([a, 1]\) for \( 0 \leq a \leq 1 \). By Theorem 1, it is easy to see that in this example \( M_{\text{on-line}}(S, C) \leq \log (n + 1) \).

We would like to understand how the off-line performs in this case.

Clearly, for every sequence \( \sigma \), an adversary can always force a mistake on the first element of the sequence. Hence, \( M(\sigma_{\text{best}}) \geq 1 \). To see that \( M(\sigma_{\text{best}}) = 1 \) we take \( \sigma_{\text{best}} = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}) \). 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

\[
\sigma_{\text{worst}} = \left( \frac{1}{2}, \frac{1}{4}, \frac{3}{8}, \frac{1}{8}, \frac{5}{8}, \frac{7}{8}, \ldots, \frac{1}{2m}, \frac{3}{2m}, \ldots, \frac{2m-1}{2m} \right).
\]
particular, it provides a characterization of $T$ defined for any concept class. We now wish to compute $M(\sigma)$ for any sequence $\sigma$. For doing this we consider the search tree $T_\sigma$ corresponding to the sequence $\sigma = s_1 s_2 \ldots s_n$. Namely, the tree constructed by the sequence of operations $\text{insert}(s_1), \text{insert}(s_2), \ldots, \text{insert}(s_n)$. We claim that $M(\sigma) = \text{rank}(T_\sigma)$. To predict on $s_1$ we consider the tree $T_\sigma$. 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 the prediction is wrong (i.e., 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 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 if $\text{rank}(T_L) < \text{rank}(T_R)$ we predict “0”. If $\text{rank}(T_L) = \text{rank}(T_R)$ we can predict arbitrarily. 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 sub-tree. 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 2 below.

The next theorem generalizes the intuition given by the above example to any concept class. In particular, it provides a characterization of $M(\sigma)$ in terms of the rank of a certain tree, $T_\sigma^C$, which is defined for any concept class $C$ and a sequence $\sigma$. This tree generalizes the idea beyond the construction of $T_\sigma$ of Example 1 above.

**Definition 1:** Consider a complete binary tree with $n + 1$ levels (recall that $n$ denotes the length of the sequence $\sigma$). We think of each level $i$ of the first $n$ levels as representing the element $s_i$ of $\sigma$. We think of an edge going from a vertex at level $i$ to its left son as representing “$s_i = 0$” and an edge going from a vertex at level $i$ to its right son as representing “$s_i = 1$”. Therefore, each leaf of the tree represents, by considering the path from the root to this leaf, an assignment of 0/1 to each of the $n$ elements of the sequence. We will call a leaf legal if there exists $c \in C$ which is consistent with all these values. Let $T_\sigma^C$ be the tree obtained by taking all the paths corresponding to legal leaves. (We shall usually drop the superscript $C$, when it is clear from the context).

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

**Proof:** To show that $M(\sigma) \leq \text{rank}(T_\sigma)$ we present an appropriate algorithm. For predicting on $s_1$, the algorithm considers the tree $T_{s_1}$ 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 the prediction is wrong (i.e., the true value of $s_1$ is 1) we repeat the same strategy on $s_2$ using the sub-tree $T_R$, which contains all the functions consistent with $s_1 = 1$. Similarly if $\text{rank}(T_L) < \text{rank}(T_R)$ we predict “1”. If $\text{rank}(T_L) = \text{rank}(T_R)$ 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 $T_\sigma$ 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 rank($T_e$) mistakes are made.

To show that no algorithm can do better, we present a strategy for the adversary for choosing a target in $C$ so as to guarantee that a given algorithm $A$ does at least rank($T_e$) mistakes. The adversary constructs for himself the tree $T_e$. 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 $A$ as the classification of $s_1, \ldots, s_{i-1}$. After getting $A$’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 $T$ than 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 $A$ 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 says that the true value is exactly the opposite than $A$’s prediction. Therefore, in such a step $A$ makes a mistake and the rank is decreased by 1. Therefore, the adversary can force a total of rank($T_e$) mistakes.

\[ \text{Remark 1: It is worth noting that, by Sauer Lemma, if the concept class } C \text{ has } VC \text{ dimension } d \text{ then the size of } T_C \text{ is bounded by } n^d. \text{ It follows that, for } C \text{ with small } VC, \text{ the tree is small and, therefore, if consistency can be checked efficiently then the construction of the tree is efficient. This, in turn, implies the efficiency of the generic (optimal) off-line algorithm of the above proof.} \]

A natural question is what is the maximum ratio between $M(\sigma_{\text{worst}})$ and $M(\sigma_{\text{best}})$. 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 $M(\sigma_{\text{best}})$ and the optimal on-line cost can not exceed $O(\log n)$. This implies a similar bound for the gap between $\sigma_{\text{best}}$ and $\sigma_{\text{worst}}$. Both are tight by our examples.

\[ \text{Theorem 3: Let } X, C \text{ be as above and } S, \text{ be a set of elements. Let } \sigma_{\text{best}} \text{ be the best sequence based on the elements of } S. \text{ Then } \mathcal{M}_{\text{on-line}}(S, C) = O(M(\sigma_{\text{best}}) \cdot \log n). \]

\[ \text{Proof: Consider the tree } T_{\sigma_{\text{best}}}. \text{ Its depth is } n + 1 \text{ and its rank, by Theorem 2, is } m \triangleq M(\sigma_{\text{best}}). \text{ By Lemma 2, this tree contains at most } (n + 1)^m \text{ leaves. That is, } |C_S| \leq (n + 1)^m. \text{ By Theorem 1, } \mathcal{M}_{\text{on-line}}(S, C) = O(m \cdot \log n), \text{ as needed.} \]

\[ \text{Corollary 4: Let } X, C \text{ be as above and } S, \text{ be a set of elements. Let } \sigma_{\text{best}} \text{ be the best sequence based on the elements of } S, \text{ and } \sigma_{\text{worst}} \text{ be the worst sequence based on the elements of } S. \text{ Then } M(\sigma_{\text{worst}}) = O(M(\sigma_{\text{best}}) \cdot \log n). \]

\[ \text{Proof: Combine Theorem 3 with Lemma 1.} \]

3.1 Characterizing the On-line

It is useful, for the study of the relations between the on-line and the off-line, to have a characterization of $M_{\text{on-line}}(S)$ in terms of ranks of trees.
Theorem 5: For all $X, C$ and $S$ as above, $M_{\text{on-line}}(S) = \max \{ \text{rank}(T) : T \in T_{C_s} \}$.

Proof: To show that $M_{\text{on-line}}(S) \geq \max \{ \text{rank}(T) : T \in T_{C_s} \}$ 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 decreased by 1 the prediction algorithm made a mistake.

To show that $M_{\text{on-line}}(S)$ is at most $m = \max \{ \text{rank}(T) : T \in T_{C_s} \}$ 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 $C^0_s$ to be all the functions in $C_s$ consistent with $s = 0$, and $C^1_s$ to be all the functions in $C_s$ consistent with $s = 1$. The algorithm compares $\max \{ \text{rank}(T) : T \in T_{C^0_s} \}$ and $\max \{ \text{rank}(T) : T \in T_{C^1_s} \}$ 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 $T_{C_s}$ whose rank is more than $m$. The prediction continues in this way, so that the maximal rank is decreased with each mistake. 

\[\square\]

4 \textbf{ } $\sigma_{\text{worst}}$ vs. On-line

4.1 Simple Algorithms

In this section we present two simple on-line algorithms, $E_1$ and $E_2$, 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 $X$. If for any sequence $\sigma$, 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 $E_1$ that makes at most one mistake on $S$, without knowing the actual order in advance. Our on-line algorithm, $E_1$, uses the guaranteed off-line algorithm $A$ and works as follows.

- Given an element $x \in S$, choose any sequence $\sigma$ that starts with $x$, and predict according to $A$’s prediction on $\sigma$, i.e. $A[\sigma]$. If a mistake is made on $x$, then $A[\sigma]$ made a mistake and it will not make any more mistakes on this sequence $\sigma$. Hence, we can use $A[\sigma](c_t(x))$ 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 $c_t(x) \neq A[\sigma]$ (otherwise $A[\sigma]$ can make another mistake). Therefore, $E_1$ 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 $E_2$ that makes at most three mistakes (which is optimal due to Claim 1 below).

\[\text{The difference between those trees, and the trees of the form } T_s \text{ is that here we are not requiring that the order of elements on each path is fixed.}\]
Call an element \( x \) \emph{bivalent} with respect to \( y \) if there exist sequences \( \sigma_b \) and \( \sigma_1 \) both starts with \( xy \) and for \( \sigma_0 \) the on-line algorithm predicts \( x = 0 \) and for \( \sigma_1 \) the on-line algorithm predicts \( x = 1 \) (i.e., \( A[\sigma_0] = 0 \) and \( A[\sigma_1] = 1 \)). Otherwise \( x \) is \emph{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.

- \textbf{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 “\( x = 1 \).

- \textbf{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 \( A[\sigma] \) predicts \( w = 0 \) and makes a mistake. Use \( b = A[\sigma](1) \) 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 \( A[\sigma](1, b) \) to get the value for all elements in \( S \).

- \textbf{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 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 \( \sigma = wxy\sigma' \). By the definition of \( y \), \( A[\sigma] \) predicts \( w = 1 \) and therefore makes its first mistake on \( w \). Denote by \( b = A[\sigma](0) \) 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 \( w = 1, y = b \). If, on the other hand, \( b \) is indeed the true value of \( y \), we denote by \( c = A[\sigma](0, b) \) its prediction on \( x \). Again, if this is wrong, we have a unique function \( f_2 \) which is consistent with \( w = 1, y = b, x = c \). 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).

\[ \text{Lemma 5: } \] Given a complete labeled binary tree \( T \) of depth \( k \) for a class \( \mathcal{C} \), there is a tree \( T' \) of rank \( \Omega(\log \log k) \) for \( \mathcal{C} \), such that at each level the labels are the same (i.e., it corresponds to a sequence).
We will show how to modify $T$ and to build a sequence $\sigma$ one element after the other. At each phase we add new elements to $\sigma$, so that they increase the rank of $T'$ by one. One may think of the $2 \cdot 2^i$ elements added in the $i$th phase as a super level. We will keep the following invariant: at the beginning of the $i$th phase, there are $2^i$ subtrees, each of rank at least $k/2^{2^i}$, and leading to the subtrees is a sequence subtree, with $i$ super levels.

The $i$th phase has $2^i$ sub-phases, in each sub-phase we split one of the $2^i$ subtrees. In a sub-phase, we consider a subtree, and take its root $r$. 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 4, the rank is at least half the original rank.

If before the $i$th 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

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

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

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

**Proof:** Assume that $M_{\text{on-line}}(S, C) = k$. By Theorem 5, there is a rank $k$ tree in $T_{C_S}$, and by Lemma 4 it contains a complete binary sub-tree $T$ of depth $k$. By Lemma 5, there is a subtree $T'$, with rank $\log \log k$ that in each level there is a single element from $S$. Let $\sigma$ be the order of the elements of $T'$ (note that on every path the order is the same). The tree of $T_{C}$ is an extension of $T'$, therefore, $\text{rank}(T') \leq \text{rank}(T_{C})$ and hence by Theorem 2, $M(\sigma) \geq \log \log k$.

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 1:** There exist $X, C$ and $S \subseteq X$, for which $M_{\text{on-line}}(S, C) = 3$ while $M(\sigma_{\text{worst}}) = 2$. (Proof omitted.)

## 5 The Off-line Mistake Bounds and The VC-dimension

In this section we present some relations between the VC dimension of a concept class and the mistake-bound-complexity of learning this class in our model. The first point one should note is the following simple observation:

**Claim 2:** For any concept class $C$ and a subset $S$ of its domain, $M(\sigma_{\text{worst}}) \geq \text{VC-dim}(C_S)$.

**Proof:** Note that if the elements in an initial segment $(\sigma_1, \ldots, \sigma_d)$ of $\sigma$ are shattered by $C$ then $M(\sigma) \geq d$. 

The more interesting part is the relations between $M(\sigma_{\text{best}})$ and the VC dimension. Although we have not reached a satisfactory understanding of this issue, we summarize in this section the partial information we have gained. We start with the following technical lemma.
Lemma 6: If \( C_S \) has VC-dim 1, then, there exist \( x \in S \) and \( b \in \{0,1\} \) such that there is a unique \( h \in C_S \) for which \( h(x) = b \).

Proof: Let us assume, for a moment, that the zero function, \( \vec{0} \), is a member of \( C_S \), we shall later see how to get rid of this assumption. Define a partial order over \( S \) by setting \( x \leq y \) iff, for every \( h \in C_S \), \( h(x) \leq h(y) \). It is easy to verify that this is indeed a partial order. Now, let \( x_0 \) be a minimal element in \( (S, \leq) \) (the existence of such an \( x_0 \) follows from the finiteness of \( S \)), and let \( b = 1 \).

We show that any function \( h \in C_S \), such that \( h(x_0) = 1 \), if \( y \geq x_0 \) then \( h(y) = 1 \) and otherwise \( h(y) = 0 \). Therefore, that there exists at most one such function.

Let \( h \) be such that \( h(x_0) = 1 \). For \( y \in S \), if \( y \geq x_0 \) then \( h(y) = 1 \) (by the definition of the relation \( \leq \)). On the other hand, for every \( y \) which is not greater than \( x_0 \), the minimality of \( x_0 \) implies that \( x_0 \) and \( y \) are incomparable. It follows that, for such a \( y \), there exists \( h_1, h_2 \in C_S \) such that \( h_1(x) = 1, h_1(y) = 0, h_2(x) = 0 \) and \( h_2(y) = 1 \). If there is any \( h \in C_S \) for which \( h(x) = 1 \) and \( h(y) = 1 \) then the set of functions \( \{h, h_1, h_2, \vec{0}\} \), which is a subset of \( C_S \), shatters \( \{x, y\} \), this contradicts our assumption that the VC-dim of \( C_S \) is 1.

It remains to show how we handle classes \( C_S \) for which \( \vec{0} \not\in C_S \). In such a case, pick any concept \( h \in C_S \) and repeat the above argument relative to the class \( C_S^h \triangleq \{h \oplus h' : h' \in C_S\} \), i.e., replace any pair \( (x,b) \) in the above argument by the pair \( (x, b \oplus h(x)) \). (Note that, for any \( h \in C_S \), \( \vec{0} \in C_S^h \)).

Theorem 7: If the VC-dimension of \( C_S \) is 1 then, \( M(\sigma_{\text{best}}) = 1 \).

Proof: The proof is by induction on \( |S| \). The base \( |S| = 1 \) then clearly \( M(\sigma_{\text{best}}) = 1 \). Otherwise we use Lemma 6 to pick the \( x \) provided by the lemma as the first element, \( s_1 \), of \( \sigma_{\text{best}} \). Let \( b_1 \) be such that there is a unique \( h \in C_S \) such that \( h(s_1) = b_1 \), and let the off-line algorithm predict \( b_1 \) on \( s_1 \). If the off-line makes a mistake then the target function is \( h \), otherwise, we have only \( |S| - 1 \) points, and we can use the induction hypothesis.

Lemma 6 and Theorem 7 fail once the VC-dim of the concept class exceeds 1. Nati Linial has come up with the following example:

Example 2: Let \( A \) be a \( 3 \times 3 \) matrix over \( \{0,1\} \) such that, on every row and on every column there are both a 1 and a 0 off the diagonal. Given any finite \( d \) construct a class, \( C_d \), of binary valued functions over a domain of size \( 3d \) by constructing the following \( 3 \cdot 2^d \times 3 \cdot d \) matrix \( B_d \) and viewing it as a table whose rows describe the functions in \( C_d \).

The matrix \( B_d \) is obtained by replacing each of the 9 entries \( A_{i,j} \) of \( A \) by a \( 2^d \times d \) matrix \( D(i,j) \) as follows: If \( i \neq j \) then \( D(i,j) \) is a constant matrix with value \( A_{i,j} \). Along the diagonal, each \( D(i,i) \) is the matrix whose rows are all the \( 2^d \) \( \{0,1\} \)-vectors of length \( d \). It is not hard to check that \( C_d \) has VC dimension \( d \), and, on every sequence \( \sigma \) enumerating its domain (i.e., the length of such a \( \sigma \) is \( 3d \)), \( M(\sigma) \geq d + 1 \).

Unfortunately, we know nothing beyond the above theorem and example. I.e., on one hand we have not been able to push the example to demonstrate any larger gap between the VC-dimension and \( M(\sigma_{\text{best}}) \), while, on the other hand, we cannot prove any upper bound on \( M(\sigma_{\text{best}}) \) which depends only on the VC-dimension, \( d \). It turns out that the above lemma and theorem also fail for concept classes over infinite domains.

Example 3: Let \( T \) be the full binary tree of all finite \( \{0,1\} \) sequences ordered by end-extension. Let \( f : \mathbb{N} \rightarrow T \) be one-to-one and onto (\( \mathbb{N} \) stands for the set of natural numbers). Now let \( \mathcal{C} \) be the collection
of all (characteristic functions of) subsets of \( N \) that \( f \) maps to a branch in \( T \) (i.e., for a branch \( b \in T \), let \( h_b : N \to \{0, 1\} \) be 1 for \( n \) such that \( f(n) \in b \) and 0 otherwise. \( C = \{h_b : b \text{ is a branch in } T\} \).

The VC-dim of \( C \) is clearly 1, on the other hand, for every finite sequence of labeled members of \( X = N, (s_1, b_1), \ldots, (s_n, b_n) \), if there exists some \( h \in C \) such that for all \( 1 \leq i \leq n \), \( h(s_i) = b_i \), then the restriction of \( C \) to the concepts that are consistent with these labeled points is a class that still has VC-dim 1. It follows that, for every permutation \( \sigma \) of \( N \), \( M(\sigma) = \infty \).

The following example demonstrates that for every \( d \), there exists a concept class \( C_S \) having VC-dim \( d \) for which \( M(\sigma_{\text{best}}) = 1 \). Note that this example uses a set \( S \) of size exponential in \( d \) (i.e., if \( |S| = n \) then \( d = O(\log n) \)). This is inevitable as by Corollary 4 and Claim 2, we have \( \text{VC-dim}(C_S) \leq M(\sigma_{\text{worst}}) = O(M(\sigma_{\text{best}}) \log n) \).

**Example 4:** Let \( C_0 = \{h_i\} \) be any finite concept class over some domain set \( X \). Let \( Z = \{z_h : h \in C\} \) be a set disjoint from \( X \). Now let \( Y \) be the set \( X \cup Z \), and let us extend functions over \( X \) to functions over \( Y \) by defining, for every \( h \in C_0 \), \( h \) to equal \( h \) on \( X \) and, for \( z \in Z \), \( h(z) = 1 \) if \( z = z_h \) and 0 if \( z \neq z_h \). Let \( C \) be \( \{h : h \in C_0\} \). Note that \( \text{VC-dim}(C) \geq \text{VC-dim}(C_0) \).

We claim that, over \( C \), \( M(\sigma_{\text{best}}) = 1 \). To see this, just pick a sequence \( \sigma \) that begins by going over all the points of \( Z \). An algorithm that predicts 0 on every member of \( Z \) will make its first mistake on \( z_{c_1} \), and, therefore, once being told that it erred, will know what \( c_1 \) is and make no further mistakes.

**Acknowledgement**

We wish to thank Nati Linial for his help with Example 2 and to Moti Frances for helpful discussions.

**References**


