On-line Learning with Linear Loss Constraints

David P. Helmbold
Computer Science Department
University of California at Santa Cruz
Santa Cruz, CA 95064
Email: dph@cse.ucsc.edu

Nicholas Littlestone
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
Email: nickl@research.nj.nec.com

Philip M. Long*
Department of Computer Science
National University of Singapore
Singapore 119260, Republic of Singapore
Email: plong@comp.nus.edu.sg

September 20, 2000

*Supported by AFOSR grant F49620–92–J–0515, a Lise Meitner Postdoctoral Fellowship from the FWF (Austria), and a UC Santa Cruz Chancellor's postdoctoral fellowship.

Abstract

We consider a generalization of the mistake-bound model (for learning $\{0, 1\}$-valued functions) in which the learner must satisfy a general constraint on the number M_+ of incorrect 1 predictions and the number M_- of incorrect 0 predictions. We describe a general-purpose optimal algorithm for our formulation of this problem.

We describe several applications of our general results, involving situations in which the learner wishes to satisfy linear inequalities in M_+ and M_-.

1 Introduction

In this paper we look at a generalization of the standard mistake-bound model [Ang88, Lit88] in which the learner wishes to guarantee that the total numbers of false positive and false negative mistakes satisfy certain constraints. In the most basic form of the standard mistake-bound model, a target function f, unknown to the learner, is chosen from a known class F of $\{0,1\}$-valued functions over some domain X. Learning is an on-line process, proceeding in *trials*. In a given trial t:

1. The learner receives instance $x_t \in X$ from the environment,
2. The learner outputs a prediction $\lambda_t \in \{0,1\}$,
3. The learner discovers the value $f(x_t)$.

Note that the learner only discovers the value of $f(x_t)$ after making the prediction in trial t. If $\lambda_t \neq f(x_t)$, we say that the learner makes a *mistake* on trial t, and (in the standard mistake-bound model) the goal of the learner is to make few mistakes in an adversarial environment.

The constraints on the learner treated in this paper can be thought of as limits on the loss incurred by the learner. We imagine that the learner suffers one or more kinds of loss. Each kind of loss is a function of the number of incorrect 1 predictions (false positive mistakes) and the number of incorrect 0 predictions (false negative mistakes). The learner's goal is to keep each kind of loss below some limit.

Our primary interest in this paper is in loss functionals that represent losses that are determined by summing per trial losses. We consider the case that the learner suffers 0 loss in any trial in which no mistake is made, and non-negative loss in each trial in which a mistake is made. If each false positive mistake incurs some loss $a \geq 0$ and each false negative mistake incurs some loss $b \geq 0$ then the total loss is given by the linear functional $g(M_+, M_-) = aM_+ + bM_-$, where M_+ is the number of false positive mistakes, and M_- is the number of false negative mistakes. We pay special attention to cases where the learner is constrained to keep one or more such functionals below given limits. The standard mistake bound model can be seen to be a special case of this setting, where the learner wishes only to constrain the value of $M_+ + M_-$.

A constraint-satisfaction learning algorithm is asked to meet certain constraints. We only look at whether or not this happens, and pay no attention to the amount by which the losses incurred by the algorithm are over or under the constraints. Thus the constraint identifies a set of acceptable M_+, M_- pairs, and the algorithm succeeds if the number of false positive and false negative mistakes it makes are one of the acceptable pairs. If the goal of the user of the algorithm is to obtain the smallest possible loss, then the user must choose the tightest achievable constraint. By choosing appropriate constraints, these algorithms can be used in various ways: They can be used to minimize a single loss where false negative and false positive mistakes have possibly different costs. They can also be used to deal with cases where the size of a loss is irrelevant while it remains below some threshold, but the loss becomes catastrophic above the threshold. Or they can be applied to the problem of keeping such a loss below a threshold while making another kind of loss as small as possible—and so forth. Taking this point of view allows us to unify our treatment of all of these cases.

The motivation for the generalization of the mistake bound model that this approach provides should be obvious. First of all, there are natural learning situations in which mistakes of different types have widely different ramifications. For instance, when diagnosing a serious disease, incorrectly hypothesizing its absence can (sometimes) be far worse than falsely conjecturing its presence. This motivates the use of a more general loss function. The simultaneous satisfaction of several constraints enables one to model situations in which the total loss according to one loss function *must* be kept below some value, and one wishes to obtain the best performance possible relative to another loss function, subject to this constraint. For example, a babysitter might be faced with occasional emergencies where it is unclear whether he should deal with them on his own or should call his employers. For a given family he works for, he wants to learn what situations are in each category. Suppose that he is convinced from past experience that if he fails to call his employers more than twice when he ought to he will be fired. He wants to minimize the number of unnecessary calls while keeping his job. This can be modeled by trying to obtain the best bound possible on M_+, given a constraint l on M_-. This, in turn, can be achieved by determining for which k a learner can simultaneously achieve $M_+ \leq k$ and $M_- \leq l$. One can also easily imagine related situations in which the learner wishes to optimize the total number of mistakes, given a constraint on one type of mistake. Finally, as discussed in a companion paper [HLL], algorithms which "trade" effectively between false positive and false negative mistakes are useful as subroutines for algorithms for learning in situations, such as visually identifying tasty apples, when the value of the hidden function f is obtained only when the learner (effectively) predicts 1 (e.g. bites into the apple).

The technical part of the paper divides into two major parts. We first present general results relating to arbitrary reasonable (see Section 2 for what we mean here by reasonable) constraints on the numbers of false positive and false negative mistakes. Later, in Section 3, we restrict our attention to linear constraints, and consider a number of particularly interesting special cases.

Since Section 2 deals with arbitrary constraint predicates, multiple constraints can be combined into a single predicate. We first present a strategy that, when invoked for a given target class and constraint, is able to guarantee satisfaction of that constraint for any sequence of trials generated by any target in the class, if any algorithm can make this guarantee. We call this strategy the Standard Constraint Satisfier Algorithm (*SCS*). The *SCS* algorithm is based on the observation that if a mistake is made in any trial then a tighter constraint must be satisfied by the total numbers of false positive and false negative mistakes made in subsequent trials. Fortunately, after each trial the target class can be reduced by eliminating those members whose values are inconsistent with the information received in that trial. The *SCS* algorithm always chooses a prediction such that if it is wrong then the resulting tighter constraint can be satisfied for the resulting reduced target class. If the original constraint can be satisfied, then there must always be an acceptable prediction for the choice *SCS* algorithm. A more formal description is given in Section 2. Algorithm *SCS* is useful for various theoretical purposes, but it may require a very time-consuming examination of the target class to determine precisely whether or not various tighter constraints are satisfiable for various subclasses.

In Section 2 we also present another algorithm, the Counting Constraint Satisfier (*CCS*), that makes a less detailed examination of the target class. We can say more about the bounds of this algorithm than the bounds of *SCS*, and, though we have not looked carefully at the efficiency of *SCS*, *SCS* appears to be much less efficient. Algorithm *CCS* can be computationally efficient for sufficiently small target classes, though it is not in general an efficient algorithm. Although *CCS* does not in general have the optimality property described for *SCS*, there are many target classes where the two algorithms are equivalent. Some sense of the relationship between the two algorithms can be gained by imagining that we start with some target class for which the algorithms are equivalent and shrink the domain of each element of the target class to some new domain X that leaves the elements distinct, thereby forming a new target class. If we give *CCS* a sequence of points from X then it will make the same predictions for the new target class that it would have made for the original target class over the larger domain. But the shrinking of the domain restricts the challenges that the learner may face in the future, perhaps permitting tighter constraints to be satisfied; *SCS* takes full advantage of this.

If a constraint C has the property that for every target class of size n there is an algorithm that guarantees its satisfaction, then *CCS* will guarantee its satisfaction for any class of size n. Thus *CCS* can be seen as giving a way of answering the following question: for each n, which constraints can be satisfied for all target classes F of size n. Results of this type are in the same spirit as the fundamental "Halving Algorithm," [BF72, Ang88, Lit88] which has served as a useful tool in analyzing target classes in the standard mistake-bound model (c.f., [GRS89] [Lit88] [Lit89] [MT89] [MT90]).

SCS is roughly a generalization of the Standard Optimal Algorithm of [Lit88] and *CCS* roughly generalizes the Halving Algorithm. In fact, for a single linear constraint that bounds the sum of the numbers of false positive and false negative mistakes (i.e. a standard mistake bound) the worst-case loss bounds obtainable using *SCS* and the worst-case bounds for *CCS* in terms of the size of the target class match the equivalent bounds for the Standard Optimal Algorithm and the Halving Algorithm, respectively. However, the Standard Optimal Algorithm and the Halving Algorithm make their predictions by asking a different question than that asked by the constraint satisfaction algorithms. The latter algorithms ask only whether predicting 0 or 1 will guarantee they can satisfy whatever constraint they were given; *SOA* and the Halving Algorithm, which aren't given a constraint, ask which prediction will let them be sure of satisfying the strongest constraint. This question makes sense where constraints have a clear total order, but is less natural for more complex constraints (for example, those constraining two losses) since there may be no single total order on the constraints that appropriately guides the choice of a better constraint for all applications. For example, sometimes the user might want to leave one loss fixed and minimize the other, and other times the user might want to minimize the maximum of the two losses. We do not consider these kinds of optimization here. The constraint-satisfaction algorithms that we consider are content to satisfy whatever constraint they were given. However, it is easy to construct algorithms, based on the ideas presented here and the ideas behind the construction of the Halving Algorithm, that perform a variety of such optimizations appropriate for various applications.

In the second part of the paper (Section 3) we restrict our attention to linear constraints. For the case of a single linear constraint on the total loss of the form $aM_+ + bM_- \leq v$, we look at how small we can guarantee the loss to be. Given a, b and n we are interested in the least value v^* such that for any target class F of size n there exists an algorithm that guarantees that $aM_+ + bM_- \leq v^*$ for any sequence of trials generated from a target in F. We show that v^* is between $\log_\alpha n - \max\{a, b\}$ and $\log_\alpha n$, where α is the solution to $\alpha^{-a} + \alpha^{-b} = 1$. Note that in the the mistake bound model (where $a = b = 1$) the halving algorithm result [Lit88] follows as a special case. When $a = b = 1$ we have bounded v^* in the range $(\log_2 n - 1, \log_2 n]$, which contains only the single integer $\lfloor \log_2 n \rfloor$.

Vovk ([Vov90]) studies asymmetric loss from a different point of view. A special case of his algorithm approximates *CCS* and can be used to obtain the $\log_\alpha n$ upper bound on v^*. However, for certain values of a and b his algorithm's performance can be substantially worse than the performance of *CCS* (see Section A.2 as well as Theorem 18 and the discussion following it). Since Vovk and we have looked at this learning question from rather different perspectives and have made different sorts of generalizations, it should be interesting to make further comparisons between his approach and ours.

Although the $\log_\alpha n$ upper bound on v^* is good when $a = b = 1$, if $a = 1$ and b is sufficiently large then $\log_\alpha n$ will be larger than n, and thus a poor upper bound on v^*. In [Vov90], Vovk does not discuss the behavior of the solution α to $\alpha^{-a} + \alpha^{-b} = 1$ as a function of a and b, and we do not know how to solve explicitly for α. However, we can make a further approximation that gives a better sense of the behavior of the bounds. We show that if $b \geq a$, then the value of the quantity v^* described above is $\Theta\left(\min\left\{a(n-1), \frac{b \ln n}{\ln(1+b/a)}\right\}\right)$. where the constant hidden by the Θ-notation is between $\frac{1}{2}$ and 2 for all choices of $n \geq 2$, $a > 0$, and $b > 0$ (see Theorem 27). Of course, a symmetric bound holds if $a \geq b$. Since algorithm *CCS* is able to satisfy the the constraint $aM_+ + bM_- \leq v^*$, this also bounds its performance.

In the special case that $a = 1$ and $b = 2$ (or vice versa), we obtain a cute result: if fib_i is the ith fibonacci number ($\mathit{fib}_0 = \mathit{fib}_1 = 1$), we show that the total loss can be made at most v for all target classes of size n if and only if $n < \mathit{fib}_{\lfloor v \rfloor + 2}$.

We also consider cases where there are two linear constraints that must be simultaneously satisfied. The most basic case is one where there are separate constraints on the numbers of false positive and false negative mistakes. We show that a learner can satisfy the pair of constraints $M_+ \leq k$ and $M_- \leq l$ for all classes of size n if and only if $\binom{k+l+2}{k+1} > n$. We also show that a learner can, for any target class of size n, simultaneously satisfy the constraints $M_+ + M_- \leq v_1$ and $M_- \leq v_2$ exactly when $\sum_{i=0}^{\lfloor v_2 \rfloor + 1} \binom{\lfloor v_1 \rfloor + 1}{i} > n$.

We conclude Section 3 by examining the relationship between satisfying a single constraint of the form $aM_+ + bM_- \leq v$ and satisfying a pair of constraints of the form $M_+ \leq k$ and $M_- \leq l$. Constraint-satisfaction algorithms allow us to trade off the numbers of false positive and false negative mistakes that the learner makes in various ways. Two basic methods are: (1) give to an algorithm the single constraint $aM_+ + bM_- \leq v$, and vary a and b depending on the desired trade-off, and (2) give to an algorithm the pair of constraints $M_+ \leq k$ and $M_- \leq l$ and vary k and l depending on the desired trade-off. How do these compare? One comparison can be made by considering the task of minimizing the maximum of the loss $aM_+ + bM_-$ over targets in some class F. The natural way to do this is to determine the minimum achievable maximum loss v and then to give an appropriate constraint-satisfaction algorithm the constraint $aM_+ + bM_- \leq v$. How well could we do if instead we gave an appropriate constraint satisfaction algorithm the pair of constraints $M_+ \leq k$ and $M_- \leq l$ for some k and l? It turns out that if we are willing to accept a doubling of the loss, then one type of constraint can be used to substitute for the other: We can use separate constraints on the numbers of false positive and false negative mistakes to approximately solve the original problem of minimizing the loss $aM_+ + bM_-$, if we are willing to incur twice the loss that we would have incurred if we had used the single constraint $aM_+ + bM_- \leq v$ (Corollary 31). Similarly, one can use a single constraint based on a linear combination of M_+ and M_- to solve the problem of keeping the numbers of false positive and false negative mistakes separately (and simultaneously) below distinct bounds if a bound which is twice the one that could be achieved by using separate constraints on M_+ and M_- is acceptable (Corollary 32). These corollaries and the more general result given in Theorem 28 indicate that in cases where it suffices to know the behavior of optimal loss bounds up to a constant factor then one need not consider algorithms for all of the different types of constraints that we consider here. One can use an algorithm set up to satisfy a single linear constraint with appropriate coefficients or, if one prefers, one can use an algorithm set up to satisfy separate constraints on the numbers of false positive and false negative mistakes.

2 General Constraints

In this section, we describe two general purpose algorithms for satisfying general constraints.

2.1 Definitions and Notation

We begin by introducing notation and making a number of definitions.

Let $\mathbf{Z}_{0+}$ denote the non-negative integers, and $\mathbf{Z}_+$ denote the positive integers.

Fix a domain X. For $x \in X$ and $c \in \{0,1\}$, $VAL_c(x)$ denotes the set of all functions from X to $\{0,1\}$ that take the value c at x. The appropriate domain should always be clear from context.

A *target class* (usually denoted by F) is a class of functions that share the same domain and have the range $\{0,1\}$. We say the domain of a target class is the domain of its functions. Elements of the domain of target class F are called *instances*. To avoid potential computability issues, we typically assume that the domain of F is finite.

We call a target class F *amply splittable* if for every subclass $F' \subseteq F$ and every positive integer $k < |F'|$ there exists an instance x in the domain of F such that $|F' \cap VAL_0(x)| = k$. Note that if a class F is amply splittable then any subclass of F is also amply splittable.

$SVAR_n$ denotes the target class consisting of the functions $f_1, \ldots, f_n$ where $f_i : \{0,1\}^n \to \{0,1\}$ is defined by $f_i(x_1, \ldots, x_n) = x_i$. INTERVAL$_n$ denotes the target class of functions defined on $\{1, ..., n\}$ consisting of $f_1, \ldots, f_n$ where $f_i^{-1}(1) = \{1, ..., i\}$. It is easy to see that both $SVAR_n$ and INTERVAL$_n$ are amply splittable.

This is a fairly strong condition. Note, for example, that if the size of the concept

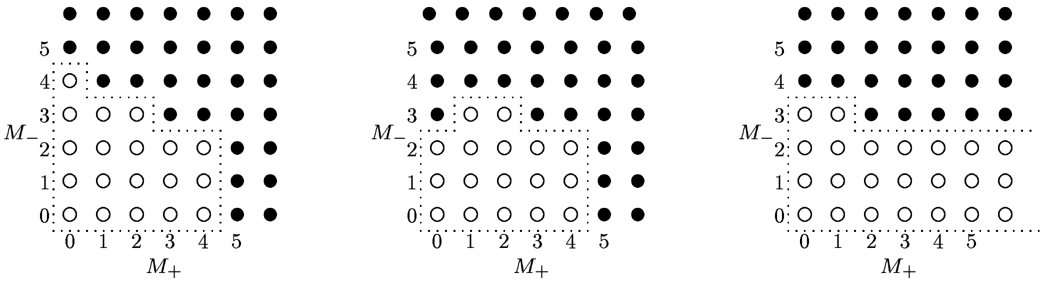

Figure 1: Three constraints are depicted above. Hollow circles and the dashed line indicate the region where the constraints are satisfied. The constraint on the left is an allowed constraint. The constraint in the middle is not allowed, as $(M_+ = 1, M_- = 3)$ satisfies the constraint while $(M_+ = 0, M_- = 3)$ does not. The constraint on the right is also not an allowed constraint as it is true on an unbounded region.

class exceeds the size of the domain by more than one, then the concept class cannot be amply splittable. This shows that many commonly considered concept classes are not amply splittable, such as the class of all conjunctions (allowing negated literals) over a space of $n \geq 2$ boolean variables. It is also easy to see that the class of 2-variable, monotone conjunctions over a space of $n \geq 3$ boolean variables is not amply splittable, since no instance gets mapped to 0 (or false) by exactly one conjunction in the class.

A *constraint predicate* is a predicate over $\mathbf{Z}_{0+}^2$. We will denote the truth of a constraint predicate C at (M_+, M_-) by $SAT(C, M_+, M_-)$. The set of *allowed constraint predicates*, which will be denoted $\mathcal{C}$, is the set of all such predicates C that are true on bounded, downward-closed regions (see Figure 1). That is, for each $C \in \mathcal{C}$ there exists a $B \in \mathbf{Z}_{0+}$ such that $SAT(C, M_+, M_-) \implies M_+ + M_- \leq B$, and for all M_+, M_-, M'_+, M'_- such that $0 \leq M'_+ \leq M_+$ and $0 \leq M'_- \leq M_-$, we have $SAT(C, M_+, M_-) \implies SAT(C, M'_+, M'_-)$. We restrict our discussion to the class of allowed constraint predicates $\mathcal{C}$. The downward-closed requirement means that if it is permissible to make some numbers of mistakes of each kind, then it is also permissible to make fewer mistakes of one or both kinds. Unbounded downward-closed constraint predicates are uninteresting, since they must allow an infinite number of mistakes of one kind or the other, and thus any such constraint is satisfiable either by the algorithm that always predicts 1 or the algorithm that always predicts 0.

We say that a constraint predicate C is *satisfiable* if there are $M_+, M_- \in \mathbf{Z}_{0+}$ for which $SAT(C, M_+, M_-)$. For allowed constraint predicates this occurs if and only if $SAT(C, 0, 0)$. Given a constraint predicate C, any sequence of instances s, any target f, and any algorithm A, we define $SAT(A, C, s, f)$ to be true if and only if $SAT(C, M_+, M_-)$, where M_+ and M_- are the numbers of false positive and false negative mistakes, respectively, that A makes on the instance sequence s with target f. For any non-empty target class F, we define $SAT(A, C, F)$ to be true if and only if, for any sequence s of instances chosen from the domain of F and any target $f \in F$, $SAT(A, C, s, f)$. If F is empty we define $SAT(A, C, F)$ to be true for any C. (In this case we do not require C to be satisfiable.)

For any target class F, we define $\mathrm{EGuar}(C, F)$ to be true if and only if there exists an

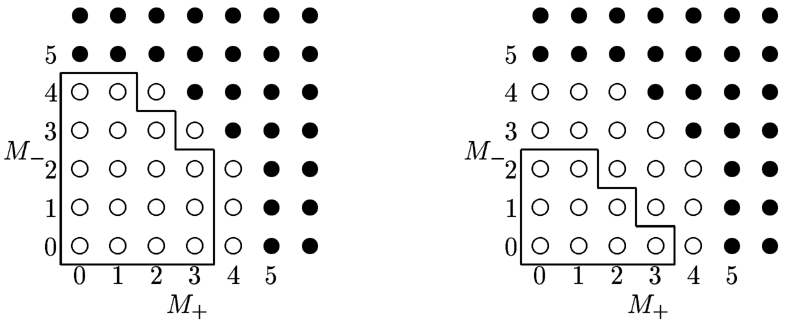

Figure 2: The above diagram illustrates a constraint C and two related constraints obtained by shifting the original constraint. On the left, the original constraint C (satisfied by the hollow circles) and the constraint $ADJ(C,1,0)$ (inside the solid line) are displayed. The diagram on the right shows the same original C and enclosed region is where the constraint $ADJ(C,1,2)$ is satisfied.

algorithm A such that $SAT(A,C,F)$. We define LFail(C) to be the least n such that there exists F of size n for which EGuar(C,F) is false. (EGuar stands for "exists guarantee" and LFail for "least fail".) It turns out to be convenient to express the recurrences that arise in terms of LFail. Note that by the above definitions, EGuar($C,\emptyset$) is true for any constraint C. For any F of size 1, EGuar(C,F) is true if and only if C is satisfiable. Thus LFail(C) $= 1$ if C is unsatisfiable, and LFail(C) ≥ 2 if C is satisfiable.

To illustrate these concepts, consider the concept class $F =$ INTERVAL$_3$ over domain $\{1,2,3\}$. The three functions in this class, f_1, f_2, f_3, map $(1,2,3)$ to $(1,0,0)$, $(1,1,0)$, and $(1,1,1)$ respectively. The halving algorithm predicts with the majority of the targets agreeing with the previous trials (and thus initially predicts as f_2). If the halving algorithm makes a mistake then actual target will be identified and no additional mistakes will be made. Thus the halving algorithm makes at most one mistake and EGuar($M_+ + M_- \leq 1, F$) is true. In fact, for every F where $|F| = 3$, the halving algorithm makes at most one mistake (as two of the three possible targets will be inconsistent whenever the halving algorithm predicts incorrectly). Now consider the class INTERVAL$_4$ on domain $\{1,2,3,4\}$. An adversary can force any algorithm to make two mistakes by first forcing the algorithm to make a mistake on instance 3 and then using either instance 2 or 4 (depending on the algorithm's first prediction) to force a second mistake. This shows that no algorithm can guarantee that $M_+ + M_- \leq 1$ is satisfied on INTERVAL$_4$. Therefore LFail($M_+ + M_- \leq 1$) ≤ 4. Since the halving algorithm satisfies the constraint on all classes of three functions, we have that LFail($M_+ + M_- \leq 1$) $= 4$.

For any constraint predicate C, and $M_+, M_- \in \mathbf{Z}_{0+}$ define $ADJ(C,M_+,M_-)$ to be the constraint predicate C' given by $SAT(C',M'_+,M'_-) \iff SAT(C,M'_+ + M_+, M'_- + M_-)$. Thus $ADJ(C,M_+,M_-)$ effectively shifts C down and to the left (see Figure 2).

When discussing a particular run of a constraint-satisfaction algorithm that is to satisfy the constraint predicate C, we define C_t to be $ADJ(C,M_+,M_-)$, where M_+ is the number of false positive mistakes made by the algorithm prior to trial t, and M_- is the corresponding number of false negative mistakes.

Given a particular target class F and a particular run of a constraint satisfaction algorithm, we let F_t denote the elements of F that are consistent with the first $t-1$ trials, i.e. if f is the target function, F_t consists of those functions $g \in F$ for which $g(x_{t'}) = f(x_{t'})$ for all $t' < t$.

2.2 Algorithms *SCS* and *CCS*

The following algorithm is the general optimal algorithm for constraint satisfaction.

Algorithm SCS. In trial t, algorithm $\mathit{SCS}(C, F)$ predicts 1 if both

$$\mathrm{EGuar}(\mathit{ADJ}(C_t, 1, 0), F_t \cap \mathit{VAL}_0(x_t))$$

and

$$|F_t \cap \mathit{VAL}_1(x_t)| > 0.$$

Otherwise, $\mathit{SCS}(C, F)$ predicts 0. □

Note that Algorithm *SCS* can be implemented whenever the required EGuar predicates can be computed. If the domain of F is finite there is a straightforward (although generally not efficient) method to evaluate the needed predicates.

The following says that this algorithm is optimal. It further establishes the fact that if $\mathrm{EGuar}(C, F)$ is false then any algorithm can be forced to make a mistake on every trial until that constraint is violated. Here we use the notation "$\mathit{violates}(A, C, s, f)$" to stand for "not $\mathit{SAT}(A, C, s, f)$".

Theorem 1 *For any satisfiable constraint predicate $C \in \mathcal{C}$, finite domain X, and non-empty target class F over X the following are equivalent:*

(a) $\mathrm{EGuar}(C, F)$

(b) The following statement is false: For every algorithm A, there exists a target $f \in F$ and a sequence of instances s such that A makes a mistake on every trial and $\mathit{violates}(A, C, s, f)$.

(c) For each $x \in X$ either $\mathrm{EGuar}(\mathit{ADJ}(C, 1, 0), F \cap \mathit{VAL}_0(x))$ *or* $\mathrm{EGuar}(\mathit{ADJ}(C, 0, 1), F \cap \mathit{VAL}_1(x))$.

(d) $\mathit{SAT}(\mathit{SCS}(C, F), C, F)$.

Proof

We prove this theorem by induction on the size of F. If $|F| = 1$, then $\mathit{SCS}(C, F)$ makes no mistakes. It is easy to see that in this case all four parts are true (since C is satisfiable) and thus equivalent. For arbitrary target classes of size $n > 1$, assume for the inductive step that the entire equivalence holds for smaller non-empty target classes. Clearly, (d) implies (a) and (a) implies (b).

To show (b) implies (c) we prove the contrapositive. We assume that there exists an x such that both $\text{EGuar}(ADJ(C,1,0), F \cap VAL_0(x))$ and $\text{EGuar}(ADJ(C,0,1), F \cap VAL_1(x))$ are false, and show that for any algorithm A, an adversary can find an $f \in F$ and a sequence of instances s such that A makes a mistake on on every instance in s and $\mathit{violates}(A, C, s, f)$. Since $\text{EGuar}(ADJ(C,1,0), F \cap VAL_0(x))$ and $\text{EGuar}(ADJ(C,0,1), F \cap VAL_1(x))$ are false, both $F \cap VAL_0(x)$ and $F \cap VAL_1(x)$ are non-empty; thus, $0 < |F \cap VAL_0(x)| < |F|$.

The adversary chooses x as the first instance in s and chooses the target so that algorithm A makes a mistake. Without loss of generality, assume that A predicts 1 on x so the target chosen by the adversary will be an $f \in F \cap VAL_0(x)$. To complete the description of the adversary we consider the algorithm A' which predicts as A does after A has seen the instance x labeled with the value 0. Thus the prediction of A' on the t^{th} instance of any sequence s' with any target $f' \in F \cap VAL_0(x)$ is the same as the $t+1^{\text{st}}$ prediction of A on the sequence (x, s') with target f'. If $ADJ(C,1,0)$ is satisfiable then we can apply the induction hypothesis to $ADJ(C,1,0)$ and $F \cap VAL_0(x)$. Since $\text{EGuar}(ADJ(C,1,0), F \cap VAL_0(x))$ is false, there exists a target $f \in F \cap VAL_0(x)$ and a sequence of instances s' such that A' makes a mistake on every trial and $\mathit{violates}(A', ADJ(C,1,0), s', f)$. If $ADJ(C,1,0)$ is not satisfiable then we set f to any function in $F \cap VAL_0(x)$ and s' to the empty sequence. Note that in both cases A' makes a mistake on every trial of s' and $\mathit{violates}(A', ADJ(C,1,0), s', f)$. The adversary chooses f as the target and creates the sequence s by appending the sequence s' to instance x. Let M'_+ and M'_- be the number of false positive and false negative mistakes made by A starting with the second trial (i.e. *not* counting the false positive mistake made on instance x). Since $\mathit{violates}(A', ADJ(C,1,0), s', f)$ holds, we have $\mathit{violates}(ADJ(C,1,0), M'_+, M'_-)$ and $\mathit{violates}(C, M'_+ + 1, M'_-)$. As A makes all the mistakes made by A' plus an additional false positive mistake on the first trial of s, we see that A makes a mistake on every trial and $\mathit{violates}(A, C, s, f)$. This completes the proof that (b) implies (c).

The final step is to show (c) implies (d). It suffices to show that $SAT(SCS(C,F), C, s, f)$ for an arbitrary instance sequence s and target $f \in F$. We will consider three cases based on the first mistake made by $SCS(C,F)$ on s. Surprisingly, we need the antecedent (c) only for the third case. The first case is when $SCS(C,F)$ makes no mistakes on s. In this case (since C is satisfiable) we have $SAT(SCS(C,F), C, s, f)$. Otherwise, let t be the first trial at which $SCS(C,F)$ makes a mistake. Note that C_t (the adjusted constraint used by $SCS(C,F)$ on trial t) equals C and the set of consistent functions, F_t, is a subset of F.

For the second case, assume that $SCS(C,F)$ predicts 1 at trial t. In this case, both $\text{EGuar}(ADJ(C,1,0), F_t \cap VAL_0(x_t))$ holds and $|F_t \cap VAL_1(x_t)| > 0$. Since SCS made a mistake, the target function maps x_t to 0 so $F_t \cap VAL_0(x_t)$ is non-empty and $0 < |F_t \cap VAL_0(x_t)| < |F_t| \leq |F|$. Also, $\text{EGuar}(ADJ(C,1,0), F_t \cap VAL_0(x_t))$ together with the non-emptyness of $F_t \cap VAL_0(x_t)$ imply that constraint $ADJ(C,1,0)$ is satisfiable. This allows us to apply the induction hypothesis to $ADJ(C,1,0)$ and $F_t \cap VAL_0(x_t)$. Using (a) $\Longrightarrow$ (d), we know for all sequences s' of elements of X and all targets $f \in F_{t+1} = F_t \cap VAL_0(x_t)$ that $SAT(SCS(ADJ(C,1,0), F_{t+1}), ADJ(C,1,0), s', f)$. Let M'_+ and M'_- be the number of false positive and false negative mistakes made by $SCS(ADJ(C,1,0), F_{t+1})$ applied to the target f on the subsequence of s beginning with its $(t+1)$-st element. Algorithm $SCS(C,F)$ makes exactly the same mistakes on s (ignoring the the extra false positive mistake on trial t) since the same constraints and sets of consistent functions are used on the trials after t.

This implies $SAT(SCS(C,F),C,s,f)$, as desired.

The final case occurs when $SCS(C,F)$ incorrectly predicts 0 at trial t. Note that $|F_t \cap VAL_1(x_t)| > 0$, since the target maps x_t to 1. From the definition of SCS, $\mathrm{EGuar}(ADJ(C,1,0), F_t \cap VAL_0(x_t))$ fails to hold, so $|F_t \cap VAL_0(x_t)| > 0$ and $0 < |F_t \cap VAL_1(x_t)| < |F_t| \leq |F|$. Since $F_t \subseteq F$ and $\mathrm{EGuar}(ADJ(C,1,0), F_t \cap VAL_0(x_t))$ fails to hold, $\mathrm{EGuar}(ADJ(C,1,0), F \cap VAL_0(x_t))$ is also false. Therefore the antecedent (c) ensures that $\mathrm{EGuar}(ADJ(C,0,1), F \cap VAL_1(x_t))$. Again using $F_t \subseteq F$, we see that $\mathrm{EGuar}(ADJ(C,0,1), F_t \cap VAL_1(x_t))$ holds. As $F_t \cap VAL_1(x_t)$ is non-empty, $ADJ(C,0,1)$ must be satisfiable. Therefore the inductive hypothesis applies to $ADJ(C,0,1)$ and $F_t \cap VAL_1(x_t)$, and we can use the argument from the previous case to complete the proof that (c) implies (d). □

We will use Theorem 1 to characterize the function LFail that maps each constraint C to the size of the smallest target class for which C cannot be guaranteed to be satisfied.

The following lemma describes a sense in which amply splittable classes are among the hardest classes of a given size.

Lemma 2 *Choose a constraint $C \in \mathcal{C}$, an amply splittable target class F, and an arbitrary target class $|F'|$ such that $|F'| = |F|$. Then* $\mathrm{EGuar}(C,F)$ *implies* $\mathrm{EGuar}(C,F')$.

Proof The proof is by induction on $|F|$. The case $F = \emptyset$ is trivial. This establishes the base case.

Assume $|F| > 0$, and that $\mathrm{EGuar}(C,F)$ is true. We wish to prove that $\mathrm{EGuar}(C,F')$ is true. By Theorem 1, it is sufficient to prove that for all x, either

$$\mathrm{EGuar}(ADJ(C,1,0), F' \cap VAL_0(x))$$

or

$$\mathrm{EGuar}(ADJ(C,0,1), F' \cap VAL_1(x)).$$

Choose x. Let $n_0 = |F' \cap VAL_0(x)|$ and $n_1 = |F' \cap VAL_1(x)|$. Since EGuar is always true for empty classes, the desired result clearly holds if n_0 or n_1 is 0. Otherwise, since $n_0 + n_1 = |F|$, we have $0 < n_0, n_1 < |F|$.

Since $\mathrm{EGuar}(C,F)$, again, by Theorem 1, for an arbitrary y in the domain of F, either

$$\mathrm{EGuar}(ADJ(C,1,0), F \cap VAL_0(y))$$

or

$$\mathrm{EGuar}(ADJ(C,0,1), F \cap VAL_1(y)).$$

Since F is amply splittable, there exists a y such that $|F \cap VAL_0(y)| = n_0$ and $|F \cap VAL_1(y)| = n_1$. Note that both $F \cap VAL_0(y)$ and $F \cap VAL_1(y)$ are amply splittable. Therefore, since $n_0, n_1 < |F|$, the induction hypothesis implies that either

$$\mathrm{EGuar}(ADJ(C,1,0), F' \cap VAL_0(x))$$

or

$$\mathrm{EGuar}(ADJ(C,0,1), F' \cap VAL_1(x))$$

as desired. □

The following lemma follows immediately as a corollary.

Lemma 3 *Choose $C \in \mathcal{C}$. Then* $\mathrm{EGuar}(C, F)$ *is false for every amply splittable target class F of size at least* $\mathrm{LFail}(C)$.

Note that from the definition of LFail, $\mathrm{EGuar}(C, F)$ is true for every target class of size less than $\mathrm{LFail}(C)$.

We apply these in the following theorem which characterizes LFail. Note that this theorem leads to an obvious dynamic programming technique for calculating $\mathrm{LFail}(C)$ for allowed constraints.

Theorem 4 LFail *is the unique function $f : \mathcal{C} \to \mathbf{Z}_+$ satisfying $f(C) = 1$ if C is unsatisfiable and $f(C) = f(ADJ(C, 1, 0)) + f(ADJ(C, 0, 1))$ otherwise.*

We present the proof of Theorem 4 as two lemmas, the first showing that LFail satisfies the recurrence and the second showing (a stronger version of) the uniqueness property.

Lemma 5 *The function* $\mathrm{LFail}(C)$ *mapping $C \in \mathcal{C}$ to $\mathbf{Z}_+$ satisfies $f(C) = 1$ if C is unsatisfiable and $f(C) = f(ADJ(C, 1, 0)) + f(ADJ(C, 0, 1))$ otherwise.*

Proof The case where C is unsatisfiable follows immediately from the definition. Otherwise, let F be any amply splittable target class of size $\mathrm{LFail}(C)$. By Lemma 3, $\mathrm{EGuar}(C, F)$ is false, so (by Theorem 1) there exists an x such that

$$\mathrm{EGuar}(ADJ(C, 1, 0), F \cap VAL_0(x))$$

and

$$\mathrm{EGuar}(ADJ(C, 0, 1), F \cap VAL_1(x))$$

are both false. Applying the definition of LFail, we have $|F \cap VAL_0(x)| \geq \mathrm{LFail}(ADJ(C, 1, 0))$ and $|F \cap VAL_1(x)| \geq \mathrm{LFail}(ADJ(C, 0, 1))$. Thus

$$\mathrm{LFail}(C) = |F| = |F \cap VAL_0(x)| + |F \cap VAL_1(x)| \geq \mathrm{LFail}(ADJ(C, 1, 0)) + \mathrm{LFail}(ADJ(C, 0, 1)).$$

It remains to show that $\mathrm{LFail}(C) \leq \mathrm{LFail}(ADJ(C, 1, 0)) + \mathrm{LFail}(ADJ(C, 0, 1))$. To do this, we now let F be any amply splittable target class of size $\mathrm{LFail}(ADJ(C, 1, 0)) + \mathrm{LFail}(ADJ(C, 0, 1))$. It is easy to see that LFail is always at least 1. Thus since F is amply splittable, there exists an x in the domain of F such that $|F \cap VAL_0(x)| = \mathrm{LFail}(ADJ(C, 1, 0))$ and $|F \cap VAL_1(x)| = \mathrm{LFail}(ADJ(C, 0, 1))$. For this x, since $F \cap VAL_0(x)$ and $F \cap VAL_1(x)$ are both amply splittable, Lemma 3 implies that

$$\mathrm{EGuar}(ADJ(C, 1, 0), F \cap VAL_0(x))$$

and

$$\mathrm{EGuar}(ADJ(C, 0, 1), F \cap VAL_1(x))$$

are both false. By Theorem 1, this implies that $\mathrm{EGuar}(C, F)$ is false, which implies that $|F| = \mathrm{LFail}(ADJ(C, 1, 0)) + \mathrm{LFail}(ADJ(C, 0, 1)) \geq \mathrm{LFail}(C)$, completing the proof. □

The following lemma completes the proof of Theorem 4 This lemma actually gives a slightly stronger uniqueness property than that required by the theorem – the stronger property will be needed later in the paper (for Lemma 12).

Lemma 6 *If $\mathcal{C}' \subseteq \mathcal{C}$ and if for all $C \in \mathcal{C}'$ we have $ADJ(C, 1, 0) \in \mathcal{C}'$ and $ADJ(C, 0, 1) \in \mathcal{C}'$, then there is a unique function $f : \mathcal{C}' \to \mathbf{Z}_+$ satisfying $f(C) = 1$ if C is unsatisfiable and $f(C) = f(ADJ(C, 1, 0)) + f(ADJ(C, 0, 1))$ otherwise.*

Proof Let $\mathcal{C}'$ be any downward-closed subset of $\mathcal{C}$ and

$$B(C) = \max\{M_+ + M_- : M_+, M_- \in \mathbf{Z}_{0+} \text{ and } SAT(C, M_+, M_-)\},$$

if C is satisfiable, and -1 otherwise. The quantity $B(C)$ is finite for every $C \in \mathcal{C}'$. We use induction to show that for every integer b, if functions f and g satisfy the conditions of the lemma then $f(C) = g(C)$ for all $C \in \mathcal{C}'$ such that $B(C) \leq b$. Once we have shown that this holds for all b we will have shown that it holds for all $C \in \mathcal{C}'$, yielding the desired result. The base case $(b = -1)$, is immediate. For the induction step, we assume that for some $b \geq 0$ and any f and g satisfying the conditions of the lemma, $f(C) = g(C)$ for every $C \in \mathcal{C}'$ such that $B(C) < b$. Let C' be an arbitrary constraint in $\mathcal{C}'$ where $B(C') = b$. Since $B(C') \geq 0$, constraint C' is satisfiable and from the definition of ADJ we have $B(ADJ(C', 0, 1)) < B(C')$ and $B(ADJ(C', 1, 0)) < B(C')$. Therefore $f(C') = f(ADJ(C', 1, 0)) + f(ADJ(C', 0, 1)) = g(ADJ(C', 1, 0)) + g(ADJ(C', 0, 1)) = g(C')$, as desired. □

We next describe Algorithm *CCS*, which usually makes its prediction on a given trial t by counting the number of functions f in F consistent with the previous trials for which $f(x_t) = 0$, and comparing this number with a threshold.

Algorithm CCS. In trial t, algorithm $CCS(C, F)$ predicts 1 if $|F_t \cap VAL_0(x_t)| < \mathrm{LFail}(ADJ(C_t, 1, 0))$ and $|F_t \cap VAL_1(x_t)| > 0$. Otherwise, $CCS(C, F)$ predicts 0. □

The following theorem gives a basic result about the relationship between *CCS* and *SCS*.

Theorem 7 *For any $C \in \mathcal{C}$ and any amply splittable target class F over a finite domain, $CCS(C, F)$ and $SCS(C, F)$ make the same predictions.*

Proof From the definition of LFail and Lemma 3, we have $\mathrm{EGuar}(ADJ(C_t, 1, 0), F_t \cap VAL_0(x_t))$ if and only if $|F_t \cup VAL_0(x_t)| < \mathrm{LFail}(ADJ(C_t, 1, 0))$. Thus, the two algorithms make the same predictions. □

The comparison between algorithms *CCS* and *SCS* is more complicated when target class F is defined on an infinite domain X. If F is finite then algorithm *CCS* can be implemented whenever the value of each $f \in F$ on each element is computable. However, even if F is finite, it may be impossible to implement algorithm *SCS* as it is unclear that $\mathrm{EGuar}(C, F)$ can always be computed when the domain is infinite.

CCS can perform very badly compared to *SCS*. Consider the target class F over the positive integers where each function maps all but two points to one, i.e. $F = \{f_{i,j} : i, j \in \mathbf{Z}_+\}$ where $f_{i,j}(k) = 0$ if $k = i$ or $k = j$ and $f_{i,j}(k) = 1$ otherwise. Algorithm *SCS* can satisfy the constraint $M_+ \leq 2$ and $M_- \leq 0$ for this class while *CCS* cannot satisfy any simultaneous finite bounds on M_+ and M_-. A similar example can be constructed using a large finite domain.

There are two obvious ways to create algorithms related to Algorithm *CCS*. First, the bias of Algorithm *CCS* towards 0 predictions can be neutralized or reversed. Second, it may be desirable to use a different function mapping constraints to reals other than LFail. In particular, one might want to use an approximation of LFail when it is difficult to compute LFail exactly. (Of course, the algorithms become less able to guarantee satisfaction of constraints as the approximation to LFail gets worse.) The following lemma is used in our analysis of Algorithm *CCS* and is stated in a such a way that it can be applied to these relatives of *CCS*.

Lemma 8 *Suppose that function* $h : \mathcal{C} \to \mathbf{R}$ *is such that* $h(C) \leq 1$ *if* C *is unsatisfiable and* $h(C) \leq h(ADJ(C, 1, 0)) + h(ADJ(C, 0, 1))$ *for every satisfiable* $C \in \mathcal{C}$*. Let* F *be a non-empty target class over a finite domain and* $C \in \mathcal{C}$ *be any allowed constraint such that* $|F| < h(C)$*. Pick a* T *in* $\mathbf{Z}_+ \cup \infty$*. Suppose that* A *is a constraint-satisfaction algorithm with the property that, in every trial* $1 \leq t < T$*, algorithm* A *predicts:*

- *0 if* $|F_t \cap VAL_0(x_t)| \geq h(ADJ(C_t, 1, 0))$ *and* $|F_t \cap VAL_0(x_t)| > 0$*,*
- *1 if* $|F_t \cap VAL_1(x_t)| \geq h(ADJ(C_t, 0, 1))$ *and* $|F_t \cap VAL_1(x_t)| > 0$*.*

(If neither of these conditions holds, then algorithm A *may make either prediction.) For every run of* A *on sequences of length* T *and targets from* F*,* $|F_t| < h(C_t)$ *for all* $t \leq T$ *and if* T *is infinite then* $SAT(A, C, F)$*.*

The parameter T is introduced to facilitate inductive proofs using this lemma, and the function h corresponds to the function LFail used by *CCS*.

Proof

First we prove that $|F_t| < h(C_t)$ for all $t \leq T$. Since $C_1 = C$ and $F_1 = F$, $|F_1| < h(C_1)$ by assumption. If a false positive mistake is made in trial $t < T$ then the value of the target function in that trial is 0, so $|F_t \cap VAL_0(x_t)| > 0$. Since the algorithm did not predict 0, we must have $|F_t \cap VAL_0(x_t)| < h(ADJ(C_t, 1, 0))$. Since in this case $F_{t+1} = F_t \cap VAL_0(x_t)$, and $C_{t+1} = ADJ(C_t, 1, 0)$ this yields $|F_{t+1}| < h(C_{t+1})$ in this case. Similarly, if a false negative mistake is made in trial $t < T$, we must have $|F_t \cap VAL_1(x_t)| < h(ADJ(C_t, 0, 1))$ and again we get $|F_{t+1}| < h(C_{t+1})$. If a mistake is not made in the tth trial, we have $|F_{t+1}| \leq |F_t|$ and $C_{t+1} = C_t$. Therefore a trivial inductive argument yields that $|F_t| < h(C_t)$ for all $t \leq T$.

To complete the proof of the lemma for the case where T is infinite, note that since $|F_t| \geq 1$ for all t, we have $h(C_t) > 1$ for all t, which by our assumptions about h implies that C_t is satisfiable. If the algorithm makes M_+ false positive mistakes and M_- false negative mistakes before trial t, then $C_t = ADJ(C, M_+, M_-)$. Since C_t is satisfiable, we have $SAT(C_t, 0, 0)$ which implies $SAT(C, M_+, M_-)$ as desired. □

The following theorem gives the basic result regarding CCS.

Theorem 9 *If* $|F| < \mathrm{LFail}(C)$*, then* $SAT(CCS(C, F), C, F)$*.*

We give two proofs of this theorem. The first relies on Lemma 8 and the second on results about Algorithm SCS.

Proof We apply Lemma 8 with $h = \mathrm{LFail}$. By Theorem 4, h satisfies the requirements of Lemma 8. To obtain the desired result we need only show that the predictions of CCS satisfy the appropriate requirements. It is clear that CCS predicts 0 when it is required to to satisfy the hypothesis of this Lemma. We will show by induction on T that for any $T > 0$ and any trial $t < T$, CCS predicts 1 if $|F_t \cap VAL_1(x_t)| \geq h(ADJ(C_t, 0, 1))$. It holds vacuously for $T = 1$. For any $T \geq 1$, we will assume that the hypotheses of the lemma are satisfied for this value T and show that this implies that they are for $T + 1$. The lemma implies that $|F_T| < h(C_T)$. Therefore $h(ADJ(C_T, 1, 0)) + h(ADJ(C_T, 0, 1)) \geq h(C_T) > |F_T| = |F_T \cap VAL_1(x_T)| + |F_T \cap VAL_0(x_T)|$. Thus if $|F_T \cap VAL_1(x_T)| \geq h(ADJ(C_T, 0, 1))$ then $|F_T \cap VAL_0(x_T)| < h(ADJ(C_T, 1, 0))$. Since $h = \mathrm{LFail}$ which is always at least 1, if $|F_T \cap VAL_1(x_T)| \geq h(ADJ(C_T, 0, 1))$ we also have $|F_T \cap VAL_1(x_T)| > 0$; thus in this case CCS will predict 1 in trial T. Thus the hypothesis of the lemma regarding the algorithm's prediction will be satisfied for any $t < T + 1$. This completes the induction step. □

We give an informal statement of the following alternate proof.

Alternate Proof of Theorem 9 Imagine expanding the domain of the target class F, defining the functions of F on the new points of the domain in such a way that F becomes amply splittable. This can be done for any target class. Let F' denote the target class F with some such expanded domain. By Theorem 7, $CCS(C, F')$ and $SCS(C, F')$ will make the same predictions. From the assumption that $|F| < \mathrm{LFail}(C)$ we have $\mathrm{EGuar}(C, F')$, so by Theorem 1 we have $SAT(SCS(C, F'), C, F')$, and thus $SAT(CCS(C, F'), C, F')$. It is easy to see that in any sequence of trials where the all of the instances are from the domain of F and the target is in F' (or equivalently F), algorithms $CCS(C, F)$ and $CCS(C, F')$ make the same predictions. We obtain the desired result by noting that this implies that if $SAT(CCS(C, F), C, F)$ did not hold, then neither would $SAT(CCS(C, F'), C, F')$. □

Given a constraint C and a target class F such that $|F| < \mathrm{LFail}(C)$, Theorems 1 and 9 show that for any run of $SCS(C, F)$ or $CCS(C, F)$, the numbers of false positive mistakes M_+ and false negative mistakes M_- will always be such that $SAT(C, M_+, M_-)$ holds. The following theorem gives a partial converse.

Theorem 10 *Suppose that C is a satisfiable constraint predicate, F is an amply splittable target class such that* $|F| = \mathrm{LFail}(C) - 1$*, and A is any algorithm for which* $SAT(A, C, F)$ *holds. Then for any non-negative integers M_+ and M_- for which* $SAT(C, M_+, M_-)$ *holds,* *there exists a sequence of $M_+ + M_-$ instances chosen from the domain of F and a target chosen from F such that A makes M_+ false positive mistakes and M_- false negative mistakes.*

Proof Since C is satisfiable, $|F| \geq 1$. We prove the theorem by induction on $M_+ + M_-$, describing how an adversary can choose an appropriate target function in F and sequence of instances. If $M_+ + M_- = 0$, then the empty sequence suffices, and any element of F can be chosen as the target. Otherwise, suppose that $M_+ > 0$. (The case where $M_+ = 0$ and $M_- > 0$ can be handled similarly.) By Theorem 4, we have $|F| = \mathrm{LFail}(C) - 1 = \mathrm{LFail}(ADJ(C, 1, 0) + \mathrm{LFail}(ADJ(C, 0, 1) - 1$. Since $SAT(C, M_+, M_-)$ we have $SAT(ADJ(C, 1, 0), M_+ - 1, M_-)$ and thus $\mathrm{LFail}(ADJ(C, 1, 0)) \geq 2$. Thus $|F| > \mathrm{LFail}(ADJ(C, 0, 1) \geq 1$. Since F is amply splittable, this implies that the adversary can choose an x such that $|F \cap VAL_1(x)| = \mathrm{LFail}(ADJ(C, 0, 1))$. When given this instance x on the first trial, the algorithm A must predict 1, since otherwise, by Lemma 3, there exists a sequence of trials (corresponding to some target in $F \cap VAL_1(x)$) for which the numbers of false positive and false negative mistakes fail to satisfy C, contradicting the assumption that $SAT(A, C, F)$ holds. The adversary will choose a target function that maps x to 0. Let A' denote an algorithm constructed from algorithm A in the same manner that an algorithm A' is constructed from some algorithm A in the proof of Theorem 1. To obtain algorithm A' we prime algorithm A by giving it the instance x and value $f(x) = 0$ in a simulated first trial before the real first trial starts. We have $SAT(A', ADJ(C, 1, 0), F \cap VAL_0(x))$ since otherwise, $SAT(A, C, F)$ would not hold. Also, $|F \cap VAL_0(x)| = \mathrm{LFail}(ADJ(C, 1, 0)) - 1$. Thus the induction hypothesis applies to the constraint $ADJ(C, 1, 0)$, the target class $F \cap VAL_0(x)$, the algorithm A', and numbers of false positive and false negative mistakes $M_+ - 1$ and M_-. If we prefix the sequence of instances yielded by application of the induction hypothesis with the instance x chosen above, and use the target yielded by the induction hypothesis, we obtain the desired sequence and target.

□

3 Linear Constraints

Here we consider constraint predicates that are expressible as conjunctions of linear inequalities. We will refer to these as *linear constraint predicates.*

3.1 Definitions, Notation, and Basic Results

We represent a linear constraint predicate as a pair $(G, \vec{v})$ where G is an $r \times 2$ matrix and $\vec{v}$ is a column vector with r components, for some r. The pair $(G, \vec{v})$ represents the constraint predicate that is true for those (M_+, M_-) such that $G\left[\begin{smallmatrix} M_+ \\ M_- \end{smallmatrix}\right] \leq \vec{v}$, where we use the convention that $\begin{bmatrix} v_1 \\ \vdots \\ v_r \end{bmatrix} \leq \begin{bmatrix} w_1 \\ \vdots \\ w_r \end{bmatrix}$ if $v_i \leq w_i$ for $i = 1, \ldots, r$.

When discussing a particular run of a constraint-satisfaction algorithm that is to satisfy the constraint predicate $(G, \vec{v})$, we define $\vec{v}_t$ to be $\vec{v} - G\left[\begin{smallmatrix} M_+ \\ M_- \end{smallmatrix}\right]$, where M_+ is the number of false positive mistakes made by the algorithm prior to trial t, and M_- is the corresponding number of false negative mistakes.

We say that a matrix of real numbers is an *allowed constraint matrix* if it has exactly two columns, all of its entries are non-negative, and each column and each row contains at least one non-zero entry. Let $\mathcal{C}_L$ denote the set of all pairs $(G, \vec{v})$ such that $\vec{v}$ is an r-component column vector with real entries, for some $r \geq 1$, and G is an allowed constraint matrix with r rows.

The following lemma indicates that it suffices to work with $\mathcal{C}_L$ when dealing with linear constraints, and characterizes satisfiable constraints.

Lemma 11 *If $(G, \vec{v}) \in \mathcal{C}_L$ then it represents a constraint predicate in $\mathcal{C}$; it is satisfiable if and only if each component of $\vec{v}$ is non-negative. If $(G, \vec{v})$ represents a constraint predicate in $\mathcal{C}$ then there exists a pair $(G', \vec{v}') \in \mathcal{C}_L$ that represents the same constraint predicate.*

If $(G, \vec{v}) \in \mathcal{C}_L$ then for all $M_+, M_- \geq 0$, we have $(G, \vec{v} - G\begin{bmatrix} M_+ \\ M_- \end{bmatrix}) \in \mathcal{C}_L$ and $(G, \vec{v} - G\begin{bmatrix} M_+ \\ M_- \end{bmatrix})$ represents the constraint predicate $ADJ((G, \vec{v}), M_+, M_-)$.

Proof The claims of the lemma are readily verified. Note that the assumption that each column of G contains at least one non-zero entry guarantees that the constraint predicate represented by $(G, \vec{v})$ is true on a bounded region. If we start with a pair $(G, \vec{v})$ that corresponds to an unsatisfiable constraint, it is trivial to find a suitable $(G', \vec{v}')$. For a satisfiable constraint, to obtain the pair $(G', \vec{v}')$ we delete all rows with no strictly positive entries from G and the corresponding components from $\vec{v}$ is in $\mathcal{C}_L$. Any row with two non-positive entries can be deleted, since the fact that $M_+ = 0$, $M_- = 0$ must satisfy the constraint implies that including such a row cannot eliminate any points from the region in which the constraint is satisfied. We replace any row with entries a, b, where one is negative and one is positive with the row with entries $\max(a, 0), \max(b, 0)$. Since the region where the constraint is satisfied is downward-closed, no point in the region can have $M_+ > v/a$ if $a > 0$ (consider what happens when $M_- = 0$) nor can it have $M_- > v/b$ if $b > 0$. Thus replacing the row as indicated does not make the constraint stronger. It is easy to see that it also does not weaken it.

For the final claim, we have $SAT(ADJ((G, \vec{v}), M_+, M_-), M'_+, M'_-)$ if and only if $G\begin{bmatrix} M_+ + M'_+ \\ M_- + M'_- \end{bmatrix} \leq \vec{v}$ if and only if $G\begin{bmatrix} M'_+ \\ M'_- \end{bmatrix} \leq \vec{v} - G\begin{bmatrix} M_+ \\ M_- \end{bmatrix})$ if and only if $SAT((G, \vec{v} - G\begin{bmatrix} M_+ \\ M_- \end{bmatrix}), M'_+, M'_-)$. □

For an allowed constraint matrix G with r rows, we define $LF_G : \mathbf{R}^r \to \mathbf{Z}$ by $LF_G(\vec{v}) = \mathrm{LFail}((G, \vec{v}))$.

Lemma 12 *Suppose that G is an allowed constraint matrix with r rows. Then LF_G is the unique function $f_G : \mathbf{R}^r \to \mathbf{Z}$ satisfying $f_G(\vec{v}) = 1$ if any of the components of $\vec{v}$ are negative and $f_G(\vec{v}) = f_G(\vec{v} - G\begin{bmatrix} 1 \\ 0 \end{bmatrix}) + f_G(\vec{v} - G\begin{bmatrix} 0 \\ 1 \end{bmatrix})$ otherwise.*

Proof This lemma follows immediately from Theorem 4, Lemma 6, and Lemma 11. □

The following lemma is a restatement of Lemma 8 for the case of linear constraints.

Lemma 13 *Suppose that F is a non-empty target class. Suppose that G is an allowed constraint matrix with r rows, and $h : \mathbf{R}^r \to \mathbf{R}$ satisfies, for every $\vec{v} \in \mathbf{R}^r$, $h(\vec{v}) \leq 1$ if any component of $\vec{v}$ is negative and $h(\vec{v}) \leq h(\vec{v} - G\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]) + h(\vec{v} - G\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right])$ otherwise. Let T be a positive integer or infinity. Suppose that A is a constraint-satisfaction algorithm with the property that, in any trial $t < T$ it predicts 0 if $|F_t \cap VAL_0(x_t)| \geq h(\vec{v}_t - G\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right])$ and $|F_t \cap VAL_0(x_t)| > 0$, and it predicts 1 if $|F_t \cap VAL_1(x_t)| \geq h(\vec{v}_t - G\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right])$ and $|F_t \cap VAL_1(x_t)| > 0$. (If neither of these conditions holds, then it may make either prediction.) For any run of A with target class F, if $|F| < h(\vec{v})$ and T is finite then $|F_t| < h(\vec{v}_t)$ for all $t \leq T$. If $|F| < h(\vec{v})$ and T is infinite then $|F_t| < h(\vec{v}_t)$ for all t and $SAT(A, (G, \vec{v}), F)$.*

The proof of this lemma is essentially identical to the proof of Lemma 8.

In the next few subsections, we consider several applications of the above results regarding linear constraint predicates, each characterized by a particular choice, or class of choices, for G.

3.2 Bounds on the numbers of false positive and false negative mistakes

Consider a robot which must return to a recharging station before its power supply is exhausted. At each stage of the journey the robot can either rush towards the recharging station and risk a collision (predicting 1) or proceed cautiously and risk running out of power (predicting 0). Assume also that v_1 collisions are enough to seriously damage the robot and v_2 unduly cautious decisions will cause the robot to run out of power before reaching the recharging station. This situation can be modeled with the simple linear constraint $(G, [v_1, v_2])$ where matrix $G = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. We call these constraints rectangular because they are satisfied on a rectangular subset of $\mathbf{Z}_{0+} \times \mathbf{Z}_{0+}$.

From Lemma 12 we have that LF_G is the unique function satisfying $LF_G(v_1, v_2) = 1$ when either $v_1 < 0$ or $v_2 < 0$ and $LF_G(v_1, v_2) = LF_G(v_1 - 1, v_2) + LF_G(v_1, v_2 - 1)$ for $v_1, v_2 \geq 0$. This has the solution for $v_1, v_2 \geq -1$ of $LF_G(v_1, v_2) = \binom{\lfloor v_1 \rfloor + \lfloor v_2 \rfloor + 2}{\lfloor v_2 \rfloor + 1}$.

If the proper robot responses (rush or caution) can be viewed as an amply splittable target class, then we can categorize when the robot will be able to make it back to the recharging station with the following lemma.

Lemma 14 *Pick $m_+, m_- \in \mathbf{Z}_{0+}$ and an amply splittable target class F. If $|F| \geq \binom{m_+ + m_- + 2}{m_- + 1}$ then for any learning algorithm A there exists a sequence of trials and target in F such that either A makes at least m_+ false positive mistakes or A makes at least m_- false negative mistakes. If $|F| < \binom{m_+ + m_- + 2}{m_- + 1}$ then there is an algorithm which always makes fewer than m_+ false positive mistakes and fewer than m_- false negative mistakes when the target is in F.*

Proof Follows from the fact that $LF_G(v_1, v_2) = \binom{m_+ + m_- + 2}{m_- + 1}$, Lemma 3, and the definition of LFail. □

Satisfying rectangular constraints has another application related to "apple tasting" [HLL]. The apple tasting learning model is an on-line learning model where the learner tries to minimize the total number of mistakes (with false positives and false negatives weighted equally). However, the value of the target on the instance remains hidden whenever the learner predicts 0. This means that the hungry learner must bite into the apple (predict 1) in order to determine if the apple contains a worm. The apple tasting model leads to interesting exploitation/exploration tradeoffs, as the learner must decide whether to exploit a hypothesis that the correct prediction is 0 or to check the hypothesis by predicting 1.

We show in the companion paper [HLL] that on-line algorithms which obtain the value of the target on every trial can be converted into apple tasting algorithms. If the number M_+ of false positive and M_- of false negative mistakes mistakes made by the original on-line algorithm satisfy $M_+ \leq v_1$ and $M_- \leq v_2$, then the resulting (randomized) apple tasting algorithm expects to make at most $v_1 + 2\sqrt{Tv_2}$ mistakes on any sequence of T trials. Furthermore, the companion paper presents lower bounds showing that if no (standard) on-line algorithm can satisfy the rectangular constraint $M_+ < v_1$ and $M_- < v_2$ then every apple tasting algorithm can be forced to make at least $\frac{1}{2}\min\{v_1, \frac{1}{2}\sqrt{v_2 T}\}$ (expected) mistakes on sequences of length[1] T (for the same target class). Therefore determining whether or not rectangular constraints can be satisfied has a direct bearing on the apple tasting complexity of a target class.

3.3 Bounds on the number of false negative mistakes and the total number of mistakes

Another specific constraint matrix for which we can solve the LF_G recurrence indicated by Lemma 12 is $G = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. This matrix represents a constraint on the total number of mistakes together with a constraint on the number of false negatives. This kind of constraint can arise in the recharging robot example if every collision costs the same amount of time as proceeding cautiously, and a certain number of collisions will cause the robot to fail.

For these kinds of constraints, LF_G is the unique function satisfying $LF(v_1, v_2) = 1$ when either $v_1 < 0$ or $v_2 < 0$ and $LF(v_1, v_2) = LF(v_1 - 1, v_2) + LF(v_1 - 1, v_2 - 1)$ otherwise. This has the solution $LF(v_1, v_2) = \sum_{i=0}^{\lfloor v_2 \rfloor + 1} \binom{\lfloor v_1 \rfloor + 1}{i}$ for $v_1, v_2 \geq -1$.

3.4 General loss

Here we consider a single linear constraint with arbitrary factors multiplying the number of false positive and false negative mistakes. Thus the constraint matrix G has the form $\begin{bmatrix} a & b \end{bmatrix}$, where a is the loss for each false positive mistake, and b is the loss for each false negative mistake. We assume that G is an allowed constraint matrix so both a and b are positive. The goal of this section is to closely approximate the value $LF_G(v)$ as a function of a, b, and v. First we state the following lemma.

Lemma 15 *For any $a, b > 0$, the equation $\alpha^{-a} + \alpha^{-b} = 1$ has exactly one positive solution. At this solution $\alpha > 1$.*

[1] Actually, this only holds if $T \geq \min\{v_1, v_2\}$.

Proof The function $f(\alpha) = \alpha^{-a} + \alpha^{-b}$ is continuous and monotone decreasing for $\alpha > 0$. It takes the value 2 at 1 and approaches 0 as α approaches infinity. This gives the desired result. □

Now we are ready to approximate LF_G.

Theorem 16 *Let* $G = \left[\begin{array}{cc} a & b \end{array}\right]$*, where* $a, b > 0$*. Let* α *be the unique positive solution of* $\alpha^{-a} + \alpha^{-b} = 1$*. Then for any* $v \geq -\max\{a, b\}$*, we have* $\alpha^v < LF_G(v) \leq \alpha^{v+\max\{a,b\}}$*.*

Proof Fix a G satisfying the required conditions. Let $B_G(v) = \max\{M_+ + M_- : M_+, M_- \in \mathbf{Z}_{0+}$ and $G\left[\begin{smallmatrix} M_+ \\ M_- \end{smallmatrix}\right] \leq v\}$ if this set is non-empty, and -1 otherwise. Because a and b are positive, $B_G(v)$ is finite. We prove this theorem for a given G by induction on $B_G(v)$. For the base case we consider values v such that $B_G(v) = -1$. In this case the constraint (G, v) is unsatisfiable, so $-\max\{a, b\} < v < 0$ and the inequality holds.

For the induction step, we assume that there is a (integer) $k \geq 0$ such that the theorem holds for all v where $B_G(v) < k$ and show that it holds for an arbitrary v where $B_G(v) = k \geq 0$ Since $B_G(v) \geq 0$, constraint (G, v) is satisfiable and $v \geq 0$. Applying Lemma 12, we see that

$$LF_G(v) = LF_G(v - a) + LF_G(v - b). \tag{1}$$

In order to apply in the inductive hypothesis, we must establish that $B_G(v - a) \leq B_G(v) - 1$ and $B_G(v - b) \leq B_G(v) - 1$. If $v - a < 0$ then $B_G(v - a) = -1 \leq B_G(v) - 1$. If $v - a \geq 0$ then there is a pair of integers, M_+ and M_-, such that $M_+ + M_- = B_G(v - a)$ and $G\left[\begin{smallmatrix} M_+ \\ M_- \end{smallmatrix}\right] \leq v - a$. Therefore, $G\left[\begin{smallmatrix} M_+ + 1 \\ M_- \end{smallmatrix}\right] \leq v$ and $B_G(v) \geq M_+ + M_- + 1 = B_G(v - a) + 1$. The analogous argument shows $B_G(v - b) \leq B_G(v) - 1$. From the inductive hypothesis we have the following.

$$\alpha^{v-a} < LF_G(v - a) \leq \alpha^{v-a+\max\{a,b\}}$$

$$\alpha^{v-b} < LF_G(v - b) \leq \alpha^{v-b+\max\{a,b\}}$$

Substituting these inequalities into Equation (1) gives

$$\alpha^v(\alpha^{-a} + \alpha^{-b}) < LF_G(v) \leq \alpha^{v+\max\{a,b\}}(\alpha^{-a} + \alpha^{-b}),$$

showing the desired result. □

Theorem 16 implies that there exists a constraint-satisfaction algorithm that satisfies the constraint (G, v) for any target class of size α^v or less. One such algorithm is an appropriate instantiation of *CCS*, which in fact satisfies the constraint for any class of size less than $LF_G(v)$. Running the algorithm *CCS* requires calculating values of *LF*. From Lemma 13 it follows that it if we are willing to approximate $LF_G(v)$ with α^v in our bound, then we can also make that approximation in the algorithm. (There is a region where this is not a good approximation, but, as remarked after Theorem 18, a simple alternate algorithm can be used in that region.) If one modifies *CCS* so that $LF_G(v)$ is approximated by α^v and applies the resulting algorithm to some target class F with v initialized to $\log_\alpha |F|$, then one obtains an on-line learning algorithm that incurs loss at most $\log_\alpha |F|$ for any target in F. This result also follows from the work of Vovk [Vov90]. His algorithms depend on a parameter λ that he restricts to $(0,1)$. If one lets λ go to 0 then one obtains from Vovk's results an algorithm that we can show satisfies the hypotheses of Lemma 13 with $h(v) = \alpha^v$. Hence we obtain the same loss bound for his algorithm with $\lambda = 0$ as for our approximation to *CCS*. The same bound also follows for Vovk's algorithm from a very different derivation given by Vovk. Vovk's algorithm (for $\lambda = 0$) predicts 1 exactly when $\left(\frac{n_0+n_1}{n_0}\right)^b > \left(\frac{n_0+n_1}{n_1}\right)^a$, where $n_i = |F_t \cap VAL_i(x_t)|$ for $i \in \{0,1\}$. (Either prediction is OK in the case of equality.) We show that this algorithm satisfies the hypotheses of Lemma 13 (in Section A.1) by first showing that this prediction rule is equivalent to the rule of predicting 1 when $\alpha^b n_1 > \alpha^a n_0$.

We now examine the loss bounds for this case more closely. We wish to study the best loss that can be guaranteed to be achieved for a given $[a\ b]$ as a function of the size n of the concept class. This is given by $\mathrm{LC}_{a,b}$, defined as follows:

$$\mathrm{LC}_{a,b}(n) = \min\{v : LF_{[a\ b]}(v) > n\}.$$

The existence of this minimum follows from the fact that $\mathrm{LC}_{a,b}(n) = \min\{v : LF_{[a\ b]}(v) \geq n+1\}$ and $LF_{[a\ b]}$ is continuous from the right, which can be shown by induction using Lemma 12.

The following trivial lemma establishes the fact that it suffices to study the case where $b \geq a = 1$.

Lemma 17 *Choose a nonnegative integer n, and $a, b, c > 0$. Then*

$$LC_{ca,cb}(n) = cLC_{a,b}(n).$$

Also,

$$\begin{aligned} LC_{a,b}(n) &= LC_{b,a}(n) \\ &= aLC_{1,b/a}(n) \\ &= bLC_{1,a/b}(n). \end{aligned}$$

Theorem 16 yields the following bound on $\mathrm{LC}_{a,b}(n)$.

Theorem 18 *Suppose $b \geq a > 0$, $n \in \mathbf{Z}_+$ and $\alpha > 0$ is the solution to $\alpha^{-a} + \alpha^{-b} = 1$. Then*

$$LC_{a,b}(n) \leq \min\{a(n-1), \log_\alpha n\}.$$

Proof First, an algorithm can obtain a loss bound of $a(n-1)$ by simply predicting 1 whenever any consistent hypothesis evaluates to 1. Each mistake eliminates at least one hypothesis, thus the number of false positive mistakes is at most $(n-1)$. This algorithm obviously never makes a false negative mistake. Thus

$$\mathrm{LC}_{a,b}(n) \leq a(n-1). \tag{2}$$

By Theorem 16, for any $v \geq 0$ we have $LF_{[a,b]}(v) > \alpha^v$. This implies that the v for which $\alpha^v = n$ provides an upper bound on $\mathrm{LC}_{a,b}(n)$. Solving for v proves that $\mathrm{LC}_{a,b}(n) \leq \log_\alpha n$. Combining this with (2) completes the proof. □

Notice that the upper bound of Theorem 18 is stronger than the $\log_\alpha |F|$ bound discussed above. In some ranges of parameters, the alternate linear bound can be significantly better. The linear bound can be obtained directly using the non-approximated version of *CCS*, or, as is done in the proof of Theorem 18, it can be obtained when desired by running an alternate algorithm that predicts 1 whenever any consistent hypothesis takes the value 1. In Appendix A.2 we show that though Vovk's algorithm achieves the $\log_\alpha |F|$ bound, it can fail to achieve the alternate linear bound.

We next prove a lower bound on $\mathrm{LC}_{a,b}(n)$ that matches the above upper bound to within a constant factor.

Theorem 19 *Suppose $b \geq a > 0$, $n \in \mathbf{Z}_+$ and $\alpha > 0$ is the solution to $\alpha^{-a} + \alpha^{-b} = 1$. Then*

$$LC_{a,b}(n) \geq \min\{a(n-1), \max\{b, \log_\alpha n - b\}\} \geq \min\left\{a(n-1), \frac{\log_\alpha n}{2}\right\}.$$

Proof If $n = 1$, then $\mathrm{LC}_{a,b}(n) = 0$, and the theorem holds. Next we consider the case where $2 \leq n < b/a + 1$. In that case, for $0 \leq v < n - 1$, from Lemma 12, we have $LF_{[1 \ \ b/a]}(v) = 2 + \lfloor v \rfloor \leq n$. Thus $\mathrm{LC}_{1,b/a}(n) \geq n - 1$. By Lemma 17 we have $\mathrm{LC}_{a,b}(n) \geq a(n-1)$, so the theorem holds for $n < b/a + 1$.

By Theorem 16, for $G = \left[\begin{array}{cc} a & b \end{array}\right]$ and for any $v \geq 0$,

$$LF_G(v) \leq \alpha^{v+\max\{a,b\}} = \alpha^{v+b}.$$

Thus if $v \leq \log_\alpha n - b$ then $LF_G(v) \leq n$, so

$$\mathrm{LC}_{a,b}(n) > \log_\alpha n - b. \tag{3}$$

We apply this bound most fruitfully when $n \geq b/a + 1$. In this case $n \geq \lceil b/a \rceil + 1$. For any $v < b/a$ we have $\lfloor v \rfloor < \lceil b/a \rceil$, so $2 + \lfloor v \rfloor \leq \lceil b/a \rceil + 1$. Thus, for $0 \leq v < b/a$ from Lemma 12 we have $LF_{[1 \ \ b/a]}(v) = 2 + \lfloor v \rfloor \leq n$. Therefore, from the definition of $\mathrm{LC}_{1,b/a}(n)$ we have $\mathrm{LC}_{1,b/a}(n) \geq b/a$. By Lemma 17 we have $\mathrm{LC}_{a,b}(n) \geq b$. Also, by (3), $\mathrm{LC}_{a,b}(n) > \log_\alpha n - b$. Therefore in this case, $\mathrm{LC}_{a,b}(n) \geq \max\{b, \log_\alpha n - b\} \geq (b + (\log_\alpha n - b))/2 = (\log_\alpha n)/2$, completing the proof. □

We may learn something more of the flavor of how $\mathrm{LC}_{a,b}(n)$ varies with a and b by obtaining looser, closed-form bounds based on the above results.

By Theorem 18 and Theorem 19, $\mathrm{LC}_{a,b}(n)$ is always within a constant factor of $\min\{a(n-1), \log_\alpha n\}$, where α is the solution to

$$\alpha^{-a} + \alpha^{-b} = 1.$$

Therefore, we may determine the rate of growth of $\mathrm{LC}_{a,b}(n)$ to within a constant factor by treating the rate of growth of $\log_\alpha n$. Since $\log_\alpha n = (\ln n)/(\ln \alpha)$ we may achieve this by determining how $1/(\ln \alpha)$ varies with a and b. If $z = 1/(\ln \alpha)$, then z is the solution to

$$e^{-a/z} + e^{-b/z} = 1,$$

so we might just as well work with the above equation, bounding the rate of growth of z as a function of a and b.

Recall that by Lemma 17, we may assume without loss of generality that $b \geq a = 1$. Our proof of an upper bound on the solution z of

$$e^{-1/z} + e^{-b/z} = 1 \tag{4}$$

when $b \geq 1$ proceeds in a series of lemmas. First, notice that it is trivial to solve for b in terms of z.

Lemma 20 *Real numbers b and z satisfy*

$$e^{-1/z} + e^{-b/z} = 1,$$

if and only if

$$b = z \ln \frac{1}{1 - e^{-1/z}}.$$

But we'd like to have bounds on z in terms of b. Thus, we'd like to (approximately) invert the right-hand side of the above lemma. The following standard approximation will prove useful.

Lemma 21 *For all x, $1 + x \leq e^x$.*

The next lemma shows that as the z in Equation 4 increases, so does the solution b in terms of z.

Lemma 22 *If g is defined on the positive reals by*

$$g(x) = x \ln \frac{1}{1 - e^{-1/x}},$$

then g is increasing on its domain, and

$$g^{-1}(1) = \frac{1}{\ln 2}.$$

Proof Since $1/x$ is a decreasing function, $-1/x$ and therefore $e^{-1/x}$ are increasing functions. Continuing in this vein, it is easy to see that $1/(1 - e^{-1/x})$ is increasing, and thence that $g(x)$ is increasing. One can readily verify that $g(1/\ln 2) = 1$.

□

We will also use the following.

Lemma 23 *For all $x \geq 1$, $\ln(1 + x) \leq 1 + \ln x$.*

Proof The fact that $\ln(1 + x) - \ln x \leq 1$ follows from the mean value theorem and the fact that the derivative of $\ln x$ is at most 1 for $x \geq 1$.

□

Now, we are ready to upper bound z.

Theorem 24 *If $b \geq 1$ and b and z satisfy*

$$e^{-1/z} + e^{-b/z} = 1,$$

then

$$z \leq \frac{2b}{\ln(1+b)}.$$

Proof First, notice that Lemma 22 establishes that $z \geq 1/\ln 2$.

We start by rewriting z.

$$\begin{aligned}
z &= \frac{1}{1-(1-1/z)} \\
z &\leq \frac{1}{1-e^{-1/z}} \quad \text{(Lemma 21)} \\
\ln z &\leq \ln\frac{1}{1-e^{-1/z}} \\
\ln z + \ln\frac{1}{1-e^{-1/z}} &\leq 2\ln\frac{1}{1-e^{-1/z}} \\
\ln z + (1 + \ln\ln\frac{1}{1-e^{-1/z}}) &\leq 2\ln\frac{1}{1-e^{-1/z}} \quad \text{(Lemma 21)} \\
1 &\leq \frac{2\ln\frac{1}{1-e^{-1/z}}}{1+\ln z+\ln\ln\frac{1}{1-e^{-1/z}}} \quad \text{(Since } z \geq 1/\ln 2\text{)} \\
z &\leq \frac{2z\ln\frac{1}{1-e^{-1/z}}}{1+\ln(z\ln\frac{1}{1-e^{-1/z}})} \\
z &\leq \frac{2b}{1+\ln b} \quad \text{(Lemma 20)} \\
z &\leq \frac{2b}{\ln(1+b)} \quad \text{(Lemma 23)}
\end{aligned}$$

This completes the proof. □

We will also make use of the following inequality.

Lemma 25 *For all $x \geq 1$,*

$$\ln(1+x) \geq x\ln(1+1/x).$$

Proof Define $f : [1,\infty) \to \mathbf{R}$ by,

$$f(x) = \ln(1+x) - x\ln(1+1/x).$$

For any $x \geq 1$,

$$f'(x) = \frac{2}{1+x} - \ln(1+\frac{1}{x})$$

$$\begin{aligned}
&\geq \frac{2}{1+x} - \frac{1}{x} \quad \text{(Lemma 21)} \\
&= \frac{2x-(1+x)}{x(1+x)} \\
&= \frac{x-1}{x(1+x)} \\
&\geq 0,
\end{aligned}$$

since, again, $x \geq 1$.

Thus, f is increasing on its domain. Since f is 0 at 1, this gives the desired result. □

Now we are ready for the following.

Theorem 26 *If $b \geq 1$ and b and z satisfy*

$$e^{-1/z} + e^{-b/z} = 1,$$

then

$$z \geq \frac{b}{\ln(1+b)}.$$

Proof Choose $b \geq 1$. Define $f_b : (0, \infty) \to \mathbf{R}$ by

$$f_b(z) = e^{-1/z} + e^{-b/z}.$$

Clearly, f_b is an increasing function. By Lemma 25,

$$\begin{aligned}
\ln(1+b) &\geq b \ln(1+1/b) \\
\frac{1}{b}\ln(1+b) &\geq \ln(1+1/b) \\
(1+b)^{1/b} &\geq 1+1/b \\
\left(\frac{1}{1+b}\right)^{1/b} &\leq \frac{b}{1+b} \\
\left(\frac{1}{1+b}\right)^{1/b} &\leq 1 - \frac{1}{1+b} \\
\left(\frac{1}{1+b}\right)^{1/b} + \frac{1}{1+b} &\leq 1 \\
\exp\left(\frac{-\ln(1+b)}{b}\right) + \exp(-\ln(1+b)) &\leq 1 \\
f_b\left(\frac{b}{\ln(1+b)}\right) &\leq 1.
\end{aligned}$$

Since f_b is increasing, we have that

$$f_b(z) = 1 \quad \Rightarrow \quad z \geq \frac{b}{\ln(1+b)}.$$

This completes the proof. □

Finally, we apply Theorems 18, 19, 24 and 26 to obtain closed-form bounds on $\mathrm{LC}_{a,b}(n)$.

Theorem 27 *Choose $b \geq a > 0$, and an integer $n \geq 1$. Then*

$$\min\left\{a(n-1), \frac{b\ln n}{2\ln(1+b/a)}\right\} \leq LC_{a,b}(n) \leq \min\left\{a(n-1), \frac{2b\ln n}{\ln(1+b/a)}\right\}.$$

Symmetric bounds hold if $a \geq b$.

Proof We prove the theorem under the assumption that $b \geq 1$ and $a = 1$. The remaining cases can be handled through trivial application of Lemma 17.

By Theorems 18 and 19, if α is the solution to

$$\alpha^{-1} + \alpha^{-b} = 1,$$

then

$$\min\left\{(n-1), \frac{\ln n}{2\ln\alpha}\right\} \leq \mathrm{LC}_{1,b}(n) \leq \min\left\{(n-1), \frac{\ln n}{\ln\alpha}\right\}. \tag{5}$$

By Theorems 24 and 26,

$$\frac{b}{\ln(1+b)} \leq \frac{1}{\ln\alpha} \leq \frac{2b}{\ln(1+b)}.$$

Combining this with (5) completes the proof. □

3.5 Fibonacci numbers

By considering the special case in which mistakes of one sort are twice as expensive as mistakes of the other, we obtain a cute example for the tools of this section and of Section 2. Here, without loss of generality, we set $G = \left[\begin{array}{cc} 1 & 2 \end{array}\right]$. In this case, by Lemma 12, $LF_G(v)$ is the unique function from $\mathbf{R}$ to $\mathbf{Z}_+$ satisfying

- $LF_G(v) = 1$ for $v < 0$
- $LF_G(v) = 2$ for $0 \leq v < 1$, and
- $LF_G(v) = LF_G(v-1) + LF_G(v-2)$ for $v \geq 1$.

One may readily recognize this as the recurrence defining the fibonacci numbers. Thus, for all $v \geq 0$, we have that $LF_{[1,2]}(v) = a_{\lfloor v \rfloor + 2}$, where $a_0 = 1, a_1 = 1, a_2 = 2, a_3 = 3, a_4 = 5, \ldots$ are the fibonacci numbers. Note also that the positive solution to $\alpha^{-1} + \alpha^{-2} = 1$ is the golden ratio, $(1+\sqrt{5})/2$.

3.6 Relationships between various constraint satisfying algorithms

The following results describe how optimal algorithms for certain constraints can be used to obtain nearly optimal algorithms for other constraints. Let $LIN_{(a,b)}(v)$ denote the constraint $(\left[\begin{array}{cc} a & b \end{array}\right], v)$ and let $RECT(v_1, v_2)$ denote the constraint $(\left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right], \left[\begin{array}{c} v_1 \\ v_2 \end{array}\right])$. We look at algorithms $SCS(LIN_{(a,b)}(v), F)$ and $SCS(RECT(v_1, v_2), F)$ for various choices of the parameters. The following theorem shows that either of these types of algorithms can be used to substitute for the other, or for *SCS* applied to any other satisfiable linear constraint predicate, if one chooses parameters appropriately and is willing to relax the constraint by doubling the constraint limits given in the vector $\vec{v}$. (The case where the components of $\vec{v}$ are not all strictly positive, which is excluded from the following theorem, is dealt with in Corollary 30.)

Theorem 28 *Suppose* $(G, \vec{v}) \in \mathcal{C}_L$*, and* r *is the number of rows in* G*. Let* $\vec{a}$ *be the first column of* G*, and* $\vec{b}$ *be the second column of* G*. Suppose further that* $v_i > 0$ *for* $i = 1, \ldots, r$*. Let* $A = \max\{\frac{a_1}{v_1}, \ldots, \frac{a_r}{v_r}\}$ *and let* $B = \max\{\frac{b_1}{v_1}, \ldots, \frac{b_r}{v_r}\}$*. For any target class* F*,*

(a) $\mathrm{EGuar}((G, \vec{v}), F) \Longrightarrow \mathit{SAT}(\mathit{SCS}(\mathit{RECT}(\frac{1}{A}, \frac{1}{B}), F), (G, 2\vec{v}), F)$.

(b) $\mathrm{EGuar}((G, \vec{v}), F) \Longrightarrow \mathit{SAT}(\mathit{SCS}(\mathit{LIN}_{(A,B)}(2), F), (G, 2\vec{v}), F)$.

Before proving this theorem, we prove a lemma we will use in the proof of the theorem.

Lemma 29 *Under the assumptions of Theorem 28,* $\mathrm{EGuar}((G, \vec{v}), F)$ *implies both* $\mathrm{EGuar}(\mathit{RECT}(\frac{1}{A}, \frac{1}{B}), F)$ *and* $\mathrm{EGuar}(\mathit{LIN}_{(A,B)}(2), F)$.

Proof Since $(G, \vec{v}) \in \mathcal{C}_L$, $A > 0$ and $B > 0$. For any $M_+, M_- \geq 0$ we have

$$\begin{aligned} &\mathit{SAT}((G, \vec{v}), M_+, M_-) \\ &\quad\Longrightarrow G\left[\begin{smallmatrix} M_+ \\ M_- \end{smallmatrix}\right] \leq \vec{v} \\ &\quad\Longrightarrow a_i M_+ + b_i M_- \leq v_i \text{ for } i = 1, \ldots, r \\ &\quad\Longrightarrow M_+ \leq \tfrac{1}{A} \text{ and } M_- \leq \tfrac{1}{B}. \end{aligned}$$

This implies both $\mathit{SAT}(\mathit{RECT}(\frac{1}{A}, \frac{1}{B}), M_+, M_-)$ and $\mathit{SAT}(\mathit{LIN}_{(A,B)}(2), M_+, M_-)$. Thus

$$\begin{aligned} &\mathrm{EGuar}((G, \vec{v}), F) \\ &\quad\Longrightarrow \mathit{SAT}(\mathit{SCS}((G, \vec{v}), F), (G, \vec{v}), F) \\ &\quad\Longrightarrow \mathit{SAT}(\mathit{SCS}((G, \vec{v}), F), \mathit{RECT}(\tfrac{1}{A}, \tfrac{1}{B}), F) \\ &\quad\Longrightarrow \mathrm{EGuar}(\mathit{RECT}(\tfrac{1}{A}, \tfrac{1}{B}), F), \end{aligned}$$

and similarly $\mathrm{EGuar}((G, \vec{v}), F) \Longrightarrow \mathrm{EGuar}(\mathit{LIN}_{(A,B)}(2), F)$. □

Proof of Theorem 28

Note that for $i = 1, \ldots, r$,

$$\mathit{SAT}(\mathit{RECT}(\frac{1}{A}, \frac{1}{B}), M_+, M_-) \Longrightarrow a_i M_+ + b_i M_- \leq \frac{a_i}{A} + \frac{b_i}{B} \leq 2v_i.$$

Thus

$$\mathit{SAT}(\mathit{RECT}(\frac{1}{A}, \frac{1}{B}), M_+, M_-) \Longrightarrow G\left[\begin{smallmatrix} M_+ \\ M_- \end{smallmatrix}\right] \leq 2\vec{v}.$$

Thus

$$\begin{aligned} &\mathrm{EGuar}(\mathit{RECT}(\tfrac{1}{A}, \tfrac{1}{B}), F) \\ &\quad\Longrightarrow \mathit{SAT}(\mathit{SCS}(\mathit{RECT}(\tfrac{1}{A}, \tfrac{1}{B}), F), \mathit{RECT}(\tfrac{1}{A}, \tfrac{1}{B}), F) \\ &\quad\Longrightarrow \mathit{SAT}(\mathit{SCS}(\mathit{RECT}(\tfrac{1}{A}, \tfrac{1}{B}), F), (G, 2\vec{v}), F). \end{aligned}$$

which, combined with Lemma 29, gives part (a).

For part (b) we note that

$$\begin{aligned} &SAT(LIN_{(A,B)}(2), M_+, M_-) \\ &\quad \Longrightarrow AM_+ + BM_- \leq 2 \\ &\quad \Longrightarrow a_i M_+ + b_i M_- \leq v_i AM_+ + v_i BM_- \leq 2v_i \text{ for } i = 1, \ldots, r. \end{aligned}$$

Thus

$$SAT(LIN_{(A,B)}(2), M_+, M_-) \Longrightarrow G\left[\begin{smallmatrix} M_+ \\ M_- \end{smallmatrix}\right] \leq 2\vec{v}.$$

Thus

$$\text{EGuar}(LIN_{(A,B)}(2), F) \Longrightarrow SAT(SCS(LIN_{(A,B)}(2), F), (G, 2\vec{v}), F)$$

which, combined with Lemma 29, gives part (b). □

The following corollary generalizes Theorem 28 to the case where some components of $\vec{v}$ may be 0.

Corollary 30 *Suppose $(G, \vec{v}) \in \mathcal{C}_L$ represents a satisfiable constraint. Let r be the number of rows in G, let $\vec{a}$ be the first column of G, and let $\vec{b}$ be the second column of G. Construct a vector $\vec{v}' = \begin{bmatrix} v_1' \\ \vdots \\ v_r' \end{bmatrix}$ as follows: For each i, if $v_i > 0$ set $v_i' = v_i$. If $v_i = 0$ and $a_i = 0$ set $v_i' = b_i/4$. If $v_i = 0$ and $b_i = 0$, set $v_i' = a_i/4$. Otherwise, set $v_i' = \min(a_i, b_i)/4$. Let $A = \max\{\frac{a_1}{v_1'}, \ldots, \frac{a_r}{v_r'}\}$ and let $B = \max\{\frac{b_1}{v_1'}, \ldots, \frac{b_r}{v_r'}\}$. For any target class F,*

(a) $\text{EGuar}((G, \vec{v}), F) \Longrightarrow SAT(SCS(RECT(\frac{1}{A}, \frac{1}{B}), F), (G, 2\vec{v}), F)$.

(b) $\text{EGuar}((G, \vec{v}), F) \Longrightarrow SAT(SCS(LIN_{(A,B)}(2), F), (G, 2\vec{v}), F)$.

Proof In each case, $v_i' > 0$ and $a_i M_+ + b_i M_- \leq v_i'$ if and only if $a_i M_+ + b_i M_- \leq v_i$. Furthermore, $a_i M_+ + b_i M_- \leq 2v_i'$ if and only if $a_i M_+ + b_i M_- \leq 2v_i$. Thus $(G, \vec{v}')$ represents that same constraint as $(G, \vec{v})$ and $(G, 2\vec{v}')$ represents that same constraint as $(G, 2\vec{v})$. Thus we obtain the corollary by applying Theorem 28 to $(G, \vec{v}')$. □

Corollary 31 *For $a, b > 0$,*

$$\text{EGuar}(LIN_{(a,b)}(v), F) \Longrightarrow SAT(SCS(RECT(\frac{v}{a}, \frac{v}{b}), F), LIN_{(a,b)}(2v), F).$$

Corollary 32 *If $v_1 > 0$ and $v_2 > 0$ then*

$$\text{EGuar}(RECT(v_1, v_2), F) \Longrightarrow SAT(SCS(LIN_{(\frac{1}{v_1}, \frac{1}{v_2})}(2), F), RECT(2v_1, 2v_2), F).$$

References

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[BF72] J. M. Barzdin and R. V. Freivald. On the prediction of general recursive functions. *Soviet Mathematics-Doklady*, 13:1224–1228, 1972.

[GRS89] S. A. Goldman, R. L. Rivest, and R. E. Schapire. Learning binary relations and total orders. *Proceedings of the 30th Annual Symposium on the Foundations of Computer Science*, 1989.

[HLL] D. P. Helmbold, N. Littlestone, and P. M. Long. Apple tasting. Submitted.

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[Lit89] N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms.* PhD thesis, UC Santa Cruz, 1989.

[MT89] W. Maass and G. Turán. On the complexity of learning from counterexamples. *Proceedings of the 30th Annual Symposium on the Foundations of Computer Science*, pages 262–267, 1989.

[MT90] W. Maass and G. Turán. On the complexity of learning from counterexamples and membership queries. *Proceedings of the 31st Annual Symposium on the Foundations of Computer Science*, 1990.

[Vov90] V. Vovk. Aggregating strategies. In *Proceedings of the 3nd Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann, 1990.

A Appendix

A.1 Verification that Vovk's algorithm satisfies hypotheses of Lemma 13

To see this, first note that since $\alpha^{-a}+\alpha^{-b}=1$ we have $\alpha^{b-a}+1=\alpha^b$ and $\alpha^{a-b}+1=\alpha^a$. Thus $\alpha^b n_1 > \alpha^a n_0$ iff $\frac{n_1}{n_0} > \alpha^{a-b}$ iff $\frac{n_0+n_1}{n_0} > 1+\alpha^{a-b} = \alpha^a$ iff $\left(\frac{n_0+n_1}{n_0}\right)^b > \alpha^{ab}$. Also, $\alpha^b n_1 > \alpha^a n_0$ iff $\alpha^{b-a} > \frac{n_0}{n_1}$ iff $\alpha^b > \frac{n_0+n_1}{n_1}$ iff $\alpha^{ab} > \left(\frac{n_0+n_1}{n_1}\right)^a$. This implies $\left(\frac{n_0+n_1}{n_0}\right)^b > \left(\frac{n_0+n_1}{n_1}\right)^a$ iff $\left(\frac{n_0+n_1}{n_0}\right)^b > \alpha^{ab}$ and thus $\alpha^b n_1 > \alpha^a n_0$ iff $\left(\frac{n_0+n_1}{n_0}\right)^b > \left(\frac{n_0+n_1}{n_1}\right)^a$, so Vovk's prediction rule is equivalent to the rule of predicting 1 if and only if $\alpha^b n_1 > \alpha^a n_0$.

Next we show that using this prediction rule yields an algorithm that satisfies the hypotheses of Lemma 13 with $h(v) = \alpha^v$. Note that $h(v - G\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]) = \alpha^{v-a}$ and $h(v - G\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]) = \alpha^{v-b}$. Since $\alpha^{-a}+\alpha^{-b}=1$ it is easy to see that h satisfies the requirements of the theorem.

We show that the remaining requirements on the predictions are satisfied using induction on t. Assume $|F| < h(\vec{v})$. We take for an induction hypothesis that if $n_{t,i} = |F_t \cap VAL_i(x_t)|$ for $i \in \{0,1\}, t \in \mathbf{Z}_+$, then

1. $|F_t| < h(v_t)$.

2. if $n_{t,0}, n_{t,1} > 0$, then on trial t Vovk's algorithm predicts 0 if $n_{t,0} \geq h(v_t - G\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right])$ and 1 if $n_{t,1} \geq h(v_t - G\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right])$.

For the base case, in which $t = 1$, by assumption $|F_1| = |F| < h(v) = h(v_1)$. Further if $n_{1,0} \geq h(v - G\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]) = \alpha^{v-a}$ then $n_{1,1} = |F| - n_{1,0} < \alpha^v - \alpha^{v-a} = \alpha^{v-b}$. Thus $\alpha^b n_{1,1} < \alpha^a n_{1,0}$ so the algorithm predicts 0 as required. Similarly, if $n_{1,1} \geq h(v - G\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]) = \alpha^{v-b}$, then $n_{1,0} < \alpha^{v-a}$ and the algorithm predicts 1 as required.

For the induction step, choose $t > 1$, and assume that the induction hypothesis holds for all $s < t$. By Lemma 13, $|F_t| < h(\vec{v}_t)$. If $n_{t,0} \geq h(v_t - G\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]) = \alpha^{v_t - a}$ then $n_{t,1} = |F_t| - n_{t,0} < \alpha^{v_t} - \alpha^{v_t-a} = \alpha^{v_t-b}$. Thus $\alpha^b n_{t,1} < \alpha^a n_{t,0}$ so the algorithm predicts 0 as required. Similarly, if $n_{t,1} \geq h(v_t - G\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]) = \alpha^{v_t-b}$, then $n_{t,0} < \alpha^{v_t-a}$ and the algorithm predicts 1 as required.

This shows that Vovk's algorithm satisfies the requirements of Lemma 13.

A.2 Vovk's algorithm is not within a constant factor of optimal

Here we show that the dependence of the loss bound of Vovk's algorithm on b and n is not within a constant factor of the best possible, by constructing a case in which the two grow together.

For a given trial t, if $n_1 = |F_t \cap VAL_1(x_t)|$, and $n_0 = |F_t \cap VAL_0(x_t)|$, then Vovk's algorithm predicts 1 exactly when

$$\left(\frac{n_0 + n_1}{n_0}\right)^b > \left(\frac{n_0 + n_1}{n_1}\right)^a.$$

Suppose $a = 1$ and $b = \lfloor (n-1) \ln n \rfloor$. In this case, by Theorem 18,

$$\mathrm{LC}_{a,b}(n) = O(n).$$

We show that Vovk's algorithm's loss behaves for $SVAR_n$ in this case, like $\Omega(n \ln n)$. Consider the first trial. Suppose $n_0 = n - 1, n_1 = 1$ (possible for $SVAR_n$). We have

$$\begin{aligned}
b &= \lfloor (n-1) \ln n \rfloor \\
b &\leq (n-1) \ln n \\
b &\leq n_0 \ln(1 + n_0) \\
b &\leq \frac{\ln(1+n_0)}{1/n_0} \\
b &\leq \frac{\ln(1+n_0)}{\ln(1+1/n_0)} \\
b \ln(1 + 1/n_0) &\leq \ln(1 + n_0)
\end{aligned}$$

$$\begin{aligned}(1+1/n_0)^b &\leq 1+n_0 \\ \left(\frac{n_0+n_1}{n_0}\right)^b &\leq \left(\frac{n_0+n_1}{n_1}\right)^a\end{aligned}$$

and Vovk's algorithm predicts 0, incurring a loss of $\lfloor(n-1)\ln n\rfloor$.