Learning with Attribute Costs

Haim Kaplan[*] Eyal Kushilevitz[†] Yishay Mansour [‡]

Abstract

We study an extension of the "standard" learning models to settings where observing the value of an attribute has an associated cost (which might be different for different attributes). Our model assumes that the correct classification is given by some target function f from a class of functions $\mathcal{F}$; most of our results discuss the ability to learn a clause (an OR function of a subset of the variables) in various settings:

Offline: We are given both the function f and the distribution D that is used to generate an input x. The goal is to design a strategy to decide what attribute of x to observe next so as to minimize the expected evaluation cost of $f(x)$. (In this setting there is no "learning" to be done but only an optimization problem to be solved; this problem turns out to be NP-hard and hence approximation algorithms are presented.)

Distributional online: We study two types of "learning" problems; one where the target function f is known to the learner but the distribution D is unknown (and the goal is to minimize the expected cost including the cost that stems from "learning" D), and the other where f is unknown (except that $f \in \mathcal{F}$) but D is known (and the goal is to minimize the expected cost while limiting the prediction error involved in "learning" f).

Adversarial online: We are given f, however the inputs are selected by an adversary. The goal is to compare the learner's cost to that of the best fixed evaluation order (i.e., we analyze the learner's performance by a competitive analysis).

1 Introduction

This paper is motivated by the observation that in many cases there is an inherent cost associated with observing each attribute of a given input x. Our goal is to model such scenarios and to offer algorithms that address this concern. The main variant of our model can be viewed as an extension of the classical PAC model [19], and has been proposed recently in the Machine Learning literature

[*]School of Computer Science, Tel-Aviv university, email haimk@post.tau.ac.il. Work partially supported by Israel Science Foundation (ISF) Grant no. 548.

[†]Computer Science, Technion, email eyalk@cs.technion.ac.il

[‡]School of Computer Science, Tel-Aviv University. The work was done while the author was a fellow in the Institute of Advance studies, Hebrew University. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778, by a grant no. 1079/04 from the Israel Science Foundation and an IBM faculty award. This publication only reflects the authors' views.

[15, 14]. To motivate the various variants of our model, we use a simple (hypothetical) medical setting. Each input x represents a patient where each attribute x_i represents the outcome of a medical test for this patient.

Consider a setting in which a doctor would like to diagnose if a patient has some disease. The doctor can run various tests on the patient in order to deduce the outcome; In fact, one would expect the order of tests to be determined so as to minimize the expected cost, an important criteria for any HMO (intuitively, one would expect that the doctor would run a simple blood test before using an expensive MRI; however, the probability of passing a certain test also influences its cost-effectiveness). This setting can be viewed as an *offline* setting, where the function f that the doctor evaluates (i.e., the final diagnosis based on the test results) is known, the distribution of the test outcomes (attributes) in the general public is also known, and the goal is to minimize the expected cost of evaluating $f(x)$.

Alternatively, consider a doctor conducting a research in order to formulate a diagnosis function f. She has the set of tests that she can run on each patient; after observing some of these tests she predicts the outcome. Since f is not known, the predictions might be incorrect. The doctor later learns whether her prediction was correct or not. Clearly, we would like to minimize those mistakes (for the insurance company sake) while maintaining a low overall cost, which is composed from learning the function and the evaluation process (for the HMO sake). We call this model the distributional online model, since the patients arrive in an online way; however, they are drawn from some fixed underlying distribution D. The above describes the case where the target function f is unknown yet the distribution D might be known. An alternative case is when D is unknown but f is known; the goal is to minimize the expected cost of the tests, while collecting information regarding the distribution. (This scenario can be motivated in cases of changing the patients' population; e.g., in the case of evaluating on humans a function learned using monkeys, where the function f is the same, but the distribution might be different.)

Finally, we consider the *adversarial online* model, where the target function is known but the adversary controls the arrival process. The hope is to bound the average online cost compared to a fixed online evaluation order. (The adversary simply models Murphy Law: *If anything can go wrong, it will.*)

1.1 Our Model and Notation

We consider Boolean functions over the domain $\{0, 1\}^n$. We assume that each attribute x_i has an associated cost c_i. The evaluation of a given function f on input x is done by specifying a decision tree T of the attributes. This decision tree T is used to decide what attributes of x to observe next, given the outcome of the attributes that we observed. The value of f is written at the leaves of the tree. The cost of evaluating an input x using a tree T is the sum of the costs of the attributes along the path that x follows in T.

We mostly deal with simple functions, like clauses, where seeing an attribute which has value one determines the value of the function. For such functions we can limit the evaluation model to a *leveled decision tree*. In a leveled decision tree, all the (internal) nodes at a the same level (distance from the root) of the tree, test the same attribute of x. For such a tree we might encounter attributes which do not have an influence on the target function f. (Specifically, attribute x_i has influence on a function g if there exists two inputs x, x' that differ only on their i-th attribute and such that $g(x) \neq g(x')$.) In case we encounter such an attribute at a node v, we assume that the subtree rooted at the left child of v and the subtree rooted at the right child of v are identical, and hence we can continue without evaluating the attribute at v. For this reason, the cost of evaluating an input x using a leveled decision tree T is the sum of the costs of the influential attributes along the path that x follows in T. A leveled decision tree is in fact equivalent to a *permutation* of the attributes, that determines the order in which we inspect them. While evaluating x we inspect the attributes one by one, in the order determined by the permutation, skipping attributes that have no influence, until the value of the function is determined. We pay the sum of the costs of the attributes that we inspected.

As mentioned, we consider target functions f that belong to certain simple classes of functions over $\{0,1\}^n$. The target function f is used to give the correct classification for each input. A *Disjunctive Normal Form* (DNF) is an OR of terms, where each term is an AND of attributes. A Read-Once DNF has every attribute appearing in at most one term. A *Conjunctive Normal Form* (CNF) is an AND of clauses, where each clause is an OR of attributes.

The input generation is done as follows. In the *adversarial online model* the inputs are chosen by an adversary. In all other models, there is a fixed distribution D and the inputs are sampled i.i.d. using D. The description of a distribution (when it is known) is given as follows. A product distribution is specified by the probabilities p_i of each attribute x_i to get the value 1 (wlog, $0 < p_i < 1$). In the case of a general distribution D, we assume a *conditional distribution oracle* that, given any assignment to a subset of the attributes, returns the conditional probability that the target function f is one, e.g., $\Pr_D[f = 1 | x_3 = 1, x_7 = 0]$.

The algorithm needs to compute (or to predict, in some settings) the value of the target function f for a given input x. By "computing the value", we mean that the algorithm needs to "explain" this value; i.e., if $f(x) = 1$ then the algorithm outputs a term of the function which is true, and if $f(x) = 0$ the algorithm outputs a clause which is zero. (Namely, even if we are given a DNF which is always true, the algorithm still needs, for every input, to exhibit a term which is true!) If $f(x) = 1$ we say that a term which equals 1 is a certificate for $f(x)$, and if $f(x) = 0$ we say that a clause which equals 0 is a certificate for $f(x)$. In case $f(x) = 1$ we denote the lowest cost of a term which is 1 by $\mathrm{CER}(f, x)$, otherwise (i.e., $f(x) = 0$) we denote the lowest cost of a clause which is 0 by $\mathrm{CER}(f, x)$.

In general our algorithms would have two performance measures. The first is the cost of evaluating $f(x)$. The second, which is relevant when the target function is unknown, is the number of mistakes performed until the correct target function is identified.

1.2 Our Results

Our main results are as follows:

1. Known function and known distribution (Offline Computation): Given a function f and a distribution D, compute the optimal/approximate permutation that minimizes the average cost of computing f. (In this setting there is no learning and the main task is an optimization.)

For a general distribution D, the optimization problem is NP-Complete and we present a 4-approximation algorithm for the case that f is a clause. We generalize the results to the case where f can be represented as a read-once DNF. For monotone functions that have a small DNF and CNF (e.g., small decision trees), we give a logarithmic approximation algorithm when the cost and the distribution are both uniform.

2. Known function and unknown distribution (Distributional Online I): Given a target function $f \in \mathcal{F}$ (but no information about D), evaluate f while minimizing the average evaluation cost.

For the case where f is a clause, we show a learning algorithm for which, with probability $1-\delta$, the cost of evaluating the first T examples is bounded by $O(T \cdot E[\mathrm{OPT}_D] + \mathrm{OPT}_D(S) \cdot poly(n))$, where $E[\mathrm{OPT}_D]$ is the expected cost of the optimal permutation and $\mathrm{OPT}_D(S)$ is the cost of OPT_D on a subset of the examples S, where $|S| = poly(n, \log 1/\delta)$.

3. Unknown function and known distribution (Distributional Online II): Given a class of functions $\mathcal{F}$, learn the target function $f \in \mathcal{F}$ while minimizing the average evaluation cost as well as the number of mistakes the algorithm performs. (There is a clear tradeoff between the two parameters, our aim would be to have a bounded number of mistakes and get our average cost to converge to the optimal average cost.)

We study product distributions and target functions that are clauses. We show how the online algorithm can guarantee at most $O(n)$ mistakes, while its expected cost is at most the optimal expected cost, in the case it is given the probability that $f = 1$, and otherwise it makes at most $O(n \log n)$ mistakes and its expected cost near the optimal expected cost.

4. Adversarial online model: Given a function f, at each time step the online algorithm selects a permutation and the adversary selects an input.

For the lower bound, we consider a simple case where the target function is a clause, known to the learner, and all the costs are equal. We show that the adversary can force any deterministic algorithm to have average cost of $n/2$ while there always exists a fixed ordering of the attributes that has average cost at most 2. For the upper bound, we show that if we evaluate the attributes in order of increasing costs then the total cost is at most n times the optimal cost, for any input and any target function.

It is important to note that our results do not depend on the magnitude of the costs, or on the largest possible cost. This is a very important feature which makes both our algorithms and analysis much more delicate.

1.3 Related work

The offline version of our problem, when the distribution is known and the function is a clause containing all the variables, is in fact a generalization of a problem which has been studied recently, under different names, one of which is Min-Sum Set Cover (MSSC). The input to the MSSC problem (like the input to the classical Set Cover problem) consists of a collection of sets $\mathcal{F} = \{S_1, \ldots, S_m\}$ containing elements from some universe $\mathcal{U}$. The goal is to find a permutation $S_{\pi(1)}, S_{\pi(2)}, \ldots, S_{\pi(m)}$ that minimize the average *covering times* $f(e)$ for all $e \in \mathcal{U}$. The covering time of $e \in \mathcal{U}$ is the minimum j such that $e \in S_{\pi(j)}$. Bar-Noy et al. [3] show that the greedy algorithm for MSSC, which iteratively picks the set with the smallest ratio of cost to the number of uncovered elements which it covers, is a 4-approximation algorithm.[1] More recently Feige, Lovasz, and Tetali [9] gave a simpler proof and also showed that it is NP-hard to approximate this problem to within a factor smaller than 4.

The MSSC problem had found several exciting applications recently. Cohen, Fiat, and Kaplan [6] considered a generalization of the problem in the context of exploiting associations to speed up search in P2P networks. Babu et al. [2, 17] used it in the context of recent work on data streams, where a stream had to go through a sequence of commutative *pipelined filters* that should be ordered so that average processing time is minimized. Some of our results may be interesting in the context of these applications as well.

The papers of Cohen et al. and of Babu et al. mentioned above, as well as this paper, generalize MSSC as defined in [9] in various ways: (1) sets are generalized to be predicates with certain success probability over some probability distribution, (2) predicates have non uniform costs, and (3) we may only have *estimates* on the success probability of each predicate. Theorem 2.4 unifies all these generalizations; The proof of [9] with some changes suffices to establish it.

Babu et al. [17] also consider an online version of MSSC similar to our adversarial online model. They use a technique of Kalai and Vempala [11] together with an offline LP-based approximation algorithm, to design a randomized online algorithm that picks a permutation for the next example, based on the examples it had seen so far. This offline algorithm, in contrast with the greedy algorithm, provides per element guarantee on its cost, compared to the fractional cost of the LP. This is crucial in order for the technique of Kalai and Vempala to work. By carefully combining these ideas they obtain a randomized online algorithm whose cost is $O(\log n * \text{OPT} + \sqrt{T})$, where OPT is the cost of the best single permutation to serve the sequence, and T is the length of the sequence.

[1] Bar-Noy et al. [3] in fact study the Min Sum Coloring (MSC) problem where the input is a graph and the output is a coloring using positive integers, where the goal is to minimize the sum over the vertices of the colors. This can be viewed as a linear ordering of the independent sets of the graph so that if we color each vertex with the index of the first independent set containing it then the sum of the colors is minimized. They showed that the greedy algorithm for MSC which iteratively picks the largest independent set is a 4-approximation algorithm. Their proof however also applies to MSSC and shows that the greedy algorithm of set cover gives a 4-approximation for MSSC. Bar-Noy, Halldorson and Kortsarz [4] give examples where the performance of greedy for MSC is tight. Their examples apply also to MSSC.

Etzioni et al. [7, 8] considered a related but different offline problem. In their work, attributes model information sources. An example x (a file, say) has $x_i = 1$ if the i^{th} information source stores x. They assume a product distribution, i.e., each source answers the query with fixed probability p_i. In their setup source i, in addition to cost c_i, also has a duration d_i which is the time that it takes source i to provide the answer. The basic problem is to find a schedule, that assigns to each source i a time t_i in which we query it, assuming we have not been able to get an answer for any source before that time. The objective is to minimize the expected overall cost of the schedule, where the overall cost of the schedule is the sum of (1) the time it takes to perform the schedule, and (2) its cost (the sum of the costs of the sources we asked). Etzioni et al. [7, 8] present a constant approximation algorithm for this basic problem. They also considered several variations of the problem, for example when we do not have to schedule all sources, and we want a schedule that maximizes our reward, where the reward is a function of the time that it takes to perform the schedule, the cost of the schedule, and whether we got an answer or not.

Charikar et al. [5] also considered an offline problem similar to ours. They looked at functions defined by boolean AND/OR binary tree. Their goal was to find a decision tree with good *competitive ratio*. The *competitive ratio of a decision tree T for a particular example x* is the ratio between the cost of T on x, and $\mathrm{CER}(f, x)$, the cost of the cheapest certificate for $f(x)$. The *competitive ratio* of T is the maximum of the competitive ratio of T over all examples x. Given a function f, Charikar et al. [5] produce a decision tree with competitive ratio bounded by $\max\{k, \ell\}$, where k is the size of the largest term in a minimal DNF representation of f, and ℓ is the size of the largest clause in a minimal CNF representation of f. Furthermore, they show how to design an algorithm specific to a function f and a cost vector c, achieving the best possible competitive ratio. In contrast our results in Section 2.2 give an algorithm for monotone functions f whose DNF and CNF representations are given, and uniform costs, that is $O(\log m)$ competitive in the terminology of Charikar et al., where m is the largest among the number of terms in the DNF representation, and the number of clauses in the CNF representation. Notice that $m \geq \max\{k, \ell\}$ and $\log m \leq \min\{k, \ell\} \log(4n)$.

The most relevant work in the machine learning literature is the work of Madani et al. [15] which describes a fixed budget learning setting. In their setting, the learner needs to optimize its performance using a fixed budget. They study the coin problem – finding the coin with the highest bias; they show that in general the optimization problem is hard, and they consider specific priors and algorithms in the fixed budget model. In [14] the performance of the Naive Bayes estimator is evaluated using a Dirichlet prior.

Littlestone [13] formulated the online (mistake-bound) learning model which we use for presenting some of our algorithms. We also use the idea of "elimination" formulated by [13] (with roots in [19]) and used as a basic technique in other learning algorithms. We note that basic classes, such as clauses, form test cases for many learning models; in particular, Valiant [19] used them to demonstrate the ability to PAC-learn interesting functions, and further justified dealing with such classes in [20]. It should also be noted that very expressive classes of functions are not known to be learnable (and in fact, under certain cryptographic assumptions, learnability of such classes may be quite limited [12]).

Smorodinsky and Tennenholtz [18] discuss a game theoretic setting, where each attribute is associated with an agent, and there is a target function. The agents can either evaluate the attribute, and pay the cost, or reply with a different value. They study the conditions under which it would be a dominating strategy for the agents to reply with the correct evaluation.

Minimizing the number of attributes observed appears also in the evaluation game of an alternating AND and OR computation tree. While deterministically one might need to evaluate all attributes, using a randomized algorithm the number of attributes evaluated is sublinear (see, [16]).

Organization: Section 2 summarizes our results for the offline problem. Section 3 discusses the adversarial online model. Section 4 contains our results in the distributional online model where the function is a clause and the distribution is unknown. Section 5 contains our results in the distributional online model where the function is not known, but the distribution is known to learner.

2 Offline Computation

In this section, we discuss the offline problem; namely, both the target function f and the distribution D are known to the algorithm. For a target function which is a clause, as stated before, the optimal decision tree is a leveled decision tree that is equivalent to a permutation of the attributes. The following theorem characterizes the optimal offline permutation for a product distribution. (This result is known, e.g., in [10], and we give the proof mainly for completeness.)

Theorem 2.1. *Given a clause $f = x_1 \vee \ldots \vee x_k$ over the input $\{0,1\}^n$, and a product distribution D, the optimal permutation is sorted according to c_i/p_i (from small to large). Hence, there is a polynomial time algorithm to compute the optimal permutation.*

Proof. Since D is a product distribution, we clearly have no benefit from observing variables not in the support of f. Consider two adjacent attributes in the optimal permutation, say x_i and x_j. Let A be the event that we reach x_i (i.e., all the attribute before it evaluated to zero) and B the event that A occurred and both $x_i = 0$ and $x_j = 0$. We like to compare the original ordering to the ordering that swaps x_i and x_j. Let R be the expected cost, given that event B occurred (R includes only the cost of attributes after x_j, not including x_j). The expected cost of the optimal order, assuming event A occurs, is

$$C_1 = p_i c_i + (1 - p_i)[c_i + p_j c_j + (1 - p_j)[c_j + R]]$$

and similarly the cost after swapping is

$$C_2 = p_j c_j + (1 - p_j)[c_j + p_i c_i + (1 - p_i)[c_i + R]].$$

From the optimality consideration we have $C_1 \leq C_2$. This implies that $p_i c_i + (1-p_i)[c_i + p_j c_j + (1-p_j)[c_j + R]] \leq p_j c_j + (1-p_j)[c_j + p_i c_i + (1-p_i)[c_i + R]]$. It follows that $p_j p_i c_i + p_j(1-p_i)c_i \leq p_i p_j c_j + p_j(1-p_j)c_j$ which, in turn, implies

$$\frac{c_i}{p_i} = \frac{p_i c_i + (1-p_i)c_i}{p_i} \leq \frac{p_j c_j + (1-p_j)c_j}{p_j} = \frac{c_j}{p_j}$$

□

For an arbitrary distribution, given by a conditional distribution oracle[2] the following Theorem states that finding the best permutation is hard.

Theorem 2.2. *Deciding if the optimal average cost of a given clause f is at most K is NP-hard.*

Proof. The reduction is from Min-Sum Set Cover (MSSC). The input to the MSSC problem is a collection of m sets, S_i, over $\{1, \dots n\}$ and an integer K. The question is whether there is a permutation of the sets such that the average cost is at most K, where the cost of i is the index of the first set S_j in the permutation such that $i \in S_j$.

Given the m sets we define m new attributes $y_1, \dots, y_m$ and a target function f which is an OR of all the attributes, i.e., $f = y_1 \vee \cdots \vee y_m$. We create the following distribution D. For each original attribute x_i we define an m-bit indicator vector z_i, where $z_{i,j} = 1$ iff $x_i \in S_j$. The distribution D is uniform over the z_i's, i.e., it assigns to each z_i, $1 \leq i \leq m$, a probability of $1/m$. (Note that it is also simple to implement the conditional distribution oracle for D.)

The theorem follows since for any permutation π over $[1, m]$, the total cost of π for MSSC and the total cost of π in our problem are identical. □

Given that the problem is NP-Hard, it is natural to seek an approximation algorithm. A natural approximation algorithm is Greedy, which selects the attributes in the permutation one after the other. The next attribute selected is the one that has the lowest marginal cost, i.e., $\mu_i = c_i/p_i$, where p_i is the probability that $x_i = 1$ given that all the attributes in the prefix are zero. Using the ideas of Feige et al. [9], we show that the Greedy algorithm is an 4-approximation algorithm (The following is a corollary of Theorem 2.4, which we prove latter).

Theorem 2.3. *For any distribution D and any clause f, the expected cost of* Greedy *is at most 4 times the expected cost of OPT.*

In the learning setting we might know the distribution only approximately. For this reason we consider a variation of algorithm Greedy, called Greedy$'$. At each iteration Greedy$'$ receives estimates of the conditional probabilities (rather than the exact conditional probabilities that Greedy uses).

We define $\hat{p}$ to be an (ϵ, γ)-*estimate* of p if it satisfies the following two conditions.

[2] Recall that a conditional distribution oracle is given any assignment to a subset of the attributes and returns the conditional probability that the target function is one, e.g., $\Pr[f = 1 | x = 1, x_7 = 0]$.

1. If $\hat{p}_i^k \neq 0$ then $\hat{p}_i^k \in [(1-\epsilon)p_i^k, (1+\epsilon)p_i^k]$.

2. For any $p_i^k > \gamma/n$ we have $\hat{p}_i^k > 0$.

In a way similar to GREEDY, algorithm GREEDY′ first determines $g(1)$, the index of its first attribute and, in general, it determines $g(k)$ after it has determined already $g(1), \ldots, g(k-1)$. Let p_j^k be the probability that $x_j = 1$ given that the attributes $x_{g(1)}$ through $x_{g(k-1)}$ evaluate to 0. Algorithm GREEDY′ = GREEDY′(ϵ, γ) is parameterized by two parameters ϵ and γ. When choosing $x_{g(k)}$ it receives (ϵ, γ)-estimates for p_i^k for every i such that $i \notin \{g(1), \ldots, g(k-1)\}$. Algorithm GREEDY′ then computes $\hat{\mu}_\ell^k = c_\ell / \hat{p}_\ell^k$, and picks attribute x_ℓ with a minimal value $\hat{\mu}_\ell^k$, and places it in the k-th position in its permutation, i.e., it sets $g(k) = \ell$.

We prove the following upper bound on the cost of the permutation produced by GREEDY′.

Theorem 2.4. *For any distribution D, the expected cost of* GREEDY′ *is at most* $4\frac{(1+\epsilon)}{(1-\epsilon)(1-2\gamma)}$ *times the expected cost of* OPT_D.

Proof. We use the following notation for a permutation π. For an example e, let $I(e, \pi)$ be the minimum j such that $e_{\pi(j)} = 1$. We define: (1) $w_i^\pi = Pr[I(x, \pi) = i]$ and $w_0^\pi = 0$. (2) $\alpha_i^\pi = \sum_{j=1}^{i} w_i^\pi$, (3) $\beta_i^\pi = 1 - \alpha_i^\pi = \Pr[I(x, \pi) \geq i+1]$, (4) $C_i^\pi = \sum_{j=1}^{i} c_{\pi(j)}$ and (5) $D(S) = \Pr[x \in S]$, where (1)-(4) are defined for $0 \leq i \leq n$.

Let g denote the permutation produced by GREEDY′ and let *opt* denote the permutation of OPT_D. For OPT_D we define a step function OP, such that $OP(y) = C_k^{opt} = \sum_{j=1}^{k} c_j^{opt}$, where $y \in [\alpha_{k-1}^{opt}, \alpha_k^{opt})$. Clearly the expected cost of OPT_D is $\int_0^1 OP(y)dy$. For GREEDY′, we define a function G such that for $y \in [\alpha_{k-1}^g, \alpha_k^g)$, $G(y) = \beta_{k-1}^g \frac{c_{g(k)}}{w_k^g}$. Again, it is easy to see that the expected cost of GREEDY′ is $\int_0^1 G(y)dy$. The crux of the proof is to show that for every k, $1 \leq k \leq n$,

$$OP(1 - \frac{\beta_{k-1}^g}{2}) \geq G(1 - \beta_{k-1}^g) \frac{(1-2\gamma)(1-\epsilon)}{2(1+\epsilon)} \ .$$

This implies that if we shrink G by a factor of $\frac{2(1+\epsilon)}{(1-2\gamma)(1-\epsilon)}$ in the y dimension and by a factor of 2 in the x dimension it fits underneath OP from which the theorem follows.

Consider OPT_D after it covered[3] a $1 - \beta_{k-1}^g/2$ fraction of the space (and has cumulative cost $OP(1 - \beta_{k-1}^g/2)$). Denote by S_1 the $1 - \beta_{k-1}^g/2$ fraction of the probability space that OPT_D covers. Let S_2 be the subset of S_1 not covered by $\{x_{g(1)}, \ldots, x_{g(k-1)}\}$. It follows that $D(S_2)$ is at least a $\beta_{k-1}^g/2$.

Denote by A the set of attributes x_j whose residual probability, p_j^k after GREEDY′ picked its first $k-1$ attributes is smaller than γ/n. Let S_3 be the subset of S_2 which is covered by attributes from A. Since $|A| \leq n$, and each attribute in A covers at most γ/n of the remaining β_{k-1}^g fraction of the space that GREEDY′ does not cover with its first $k-1$ attributes (i.e. those examples for which $x_{g(1)}, x_{g(2)}, \ldots, x_{g(k-1)}$ are all zeros), we obtain that $D(S_3)$ is no larger than $\beta_{k-1}^g \cdot n(\gamma/n) = \beta_{k-1}^g \gamma$.

[3]We use terminology related to set cover here. By saying that an algorithm "covers" some fraction p of the probability space with a set of attributes Y, we mean that $p = \Pr[\vee_{i \in Y} x_i = 1]$.

Let $S_4 = S_2 - S_3$, we have that $D(S_4) \geq \beta^g_{k-1}(1/2 - \gamma)$. Obviously since $D(S_4) > 0$ there must be an attribute x_j whose residual probability $p^k_j \geq \gamma/n$. In particular $x_{g(k)}$ is such an attribute.

Let $q = \Pr[x_{g(1)} = 0, \ldots, x_{g(k-1)} = 0]$. The probability $p^k_{g(k)}$ that $x_{g(k)} = 1$ given that $x_{g(1)}, x_{g(2)}, \ldots, x_{g(k-1)}$ are all zeros is w^g_k/q by the definitions of w^g_k and q. So the residual cost of $x_{g(k)}$ after GREEDY$'$ picked $x_{g(1)}, x_{g(2)}, \ldots, x_{g(k-1)}$ is $c_{g(k)}/(w^g_k/q)$.

The optimal algorithm OPT_D must also have attributes $x_o \notin \{x_{g(1)}, x_{g(2)}, \ldots, x_{g(k-1)}\} \cup A$ to cover S_4. Let x_o be one such attribute and let p^k_o the probability that x_o is one given that $x_{g(1)}, x_{g(2)}, \ldots, x_{g(k-1)}$ are zero.

Since GREEDY$'$ picks an attribute with smallest estimate of residual cost we know that the estimate that GREEDY$'$ had for the residual cost of x_o was larger than the estimate GREEDY$'$ had for the residual cost of $x_{g(k)}$ when it picked its kth attribute. If $\hat{p}^k_o$ is the estimate that GREEDY$'$ had for the residual probability of x_o and $\hat{p}^k_{g(k)}$ is the estimate that GREEDY$'$ had for the residual probability of $x_{g(k)}$ then we have that

$$\frac{c_{g(k)}}{(1+\epsilon)w^g_k/q} \leq \frac{c_{g(k)}}{\hat{p}^k_{g(k)}} \leq \frac{c_o}{\hat{p}^k_o} \leq \frac{c_o}{(1-\epsilon)p^k_o}$$

Rearranging the rightmost and leftmost sides we get that

$$\frac{(1-\epsilon)p^i_o q c_{g(k)}}{(1+\epsilon)w^g_k} \leq c_o \ .$$

Summing up over all attribute x_o that OPT$_D$ uses to cover S_4 we obtain that

$$OP(1 - \beta^g_{k-1}/2) \geq \beta^g_{k-1}\frac{(1-2\gamma)(1-\epsilon)}{2(1+\epsilon)}\frac{c_{g(k)}}{w^g_k} = G(1-\beta^g_{k-1})\frac{(1-2\gamma)(1-\epsilon)}{2(1+\epsilon)} \ ,$$

since $\sum_o p^k_o q \geq \beta^g_{k-1}\frac{1-2\gamma}{2}$. □

Remark. Note that GREEDY is simply GREEDY$'$ with $\gamma = 0$ and $\epsilon = 0$, and therefore it has an approximation ratio of 4.

2.1 Read-Once DNF

We first show that the optimal permutation for computing a Read-Once (R/O) DNF is a permutation on the terms. A *permutation of terms* is a permutation of the attributes that can be described as first a permutation of the terms and then for each term a permutation of its variables. (We will use this definition only for R/O DNF, in which case it is well defined.) Surprisingly, the following lemma shows that for any distribution and any set of costs there is an optimal permutation which is a permutation of terms.

Lemma 2.5. *Let f be a R/O DNF and π a permutation. There exists a permutation π' which is a permutation of the terms, whose cost on any input is at most the cost of π.*

Proof. Let π be any permutation. Consider the index i which is the least index such that there exists a term T_j which is a subset of the first i variables of π. (Note that since f is a R/O DNF the term T_j is unique.) Let π_1 be the permutation that tests T_j first and then continues according to π. We claim that the cost of π_1, for *any* input, is at most the cost of π.

Let x be an input for which $f(x) = 1$. In such a case we have to reach index i before we terminate, and therefore testing T_j first cannot increase the cost. Note that the same variables in T_j are evaluated in both π and π_1. For the other variables, any variable that is relevant in π_1 is also relevant in π. Therefore the cost can only decrease.

Let y be an input for which $f(y) = 0$. In such a case we have to find for *every* term an attribute which is zero. Since the function is read-once, T_j does not share any variables with any other term. This implies that once we find a variable in T_j that is zero, we do not evaluate any of its remaining variables (here is where we use the read-once assumption). This implies that the cost of evaluating $f(y)$ equals the sum of the costs of evaluating $T_j(y)$. In this case any ordering that will maintain the order within each term will have the same cost.

We showed how to "pull-out" one term. We can recursively perform the claim on the remaining DNF and get an optimal permutation which is a permutation of terms. □

The above lemma is somewhat surprising. On one hand, we need the read-once property, otherwise the claim is not true. On the other hand, we do not assume anything about the distribution. In many learning settings, when we have an arbitrary distribution, one can transform any function to be read-once, by duplicating the appearance of the attributes. In our case, duplicating attributes implies that each time we evaluate them we are charged again, this is why the "standard" reduction fails. To illustrate that the read once assumption is crucial, consider $f = x_1x_2 \vee x_2x_3x_4$ with uniform cost. On input 0101 the optimal order $x_1x_3x_2x_4$ has cost of 2. This order is not a permutation of terms (and the cost of any such permutation is larger), however f is clearly not read once.

We can now extend Theorems 2.1 and 2.3 to R/O DNF.

Theorem 2.6. *Let f be a function which is a R/O DNF. Then, (1) For a product distribution P, there is a polynomial time algorithm to compute the optimal permutation, and (2) for any distribution D, there is a polynomial time algorithm to compute a permutation whose expected cost is 4 times the optimal permutation.* □

2.2 Small CNF and DNF

Assume that we are given a monotone function f that has at most m terms in its DNF representation and at most m clauses in its CNF representation. (We are given both the CNF and the DNF description of the function.) We assume uniform costs (i.e., $c_i = 1$ for every i), and uniform distribution U from which examples are drawn.

To formulate the upper bound, recall the definition in Section 1.1: $\mathrm{CER}(f, x)$, when $f(x) = 1$ is the cost of the cheapest term of f which is one, and when $f(x) = 0$, $\mathrm{CER}(f, x)$ is the cost of the cheapest clause of f which is zero.[4] We show how to achieve an average cost which is bounded by the average (over U) of $\mathrm{CER}(f,x)$ times $O(\ln m)$. Obviously the optimal cost for x is at least $\mathrm{CER}(f,x)$ since the partial assignment it observes must always form a certificate.

Theorem 2.7. *Let f be a monotone function that can be represented by both an m term DNF and by an m clause CNF. For uniform costs there is an algorithm that evaluates f with expected cost of $O(E_D[\mathrm{CER}(f,x)]\ln m)$.*

Proof. Since f is monotone we can assume, without loss of generality, that the DNF and CNF representations are both monotone (i.e., consisting only of non-negated variables). The idea is that we create two programs, one assuming that the function is one and the other assuming that the function is zero. We interleave the two programs, and show that the cost of the program with the correct assumption is at most a factor $O(\ln m)$ from the expected value of $\mathrm{CER}(f,x)$. At each iteration we evaluate two variables. One that appears in the most terms (in the DNF) that were not yet eliminated and the other that appears in the most clauses (in the CNF) that were not yet eliminated.

Given an input x, assume that $f(x)=1$ (the case $f(x)=0$ is similar). To simplify the analysis, we assume that the cost is 1 only when the variable evaluates to 1 and zero otherwise. We denote this new cost function by $cost_1$. (We will overcome this assumption at the end of the proof.) Since $f(x)=1$, some term of f is one according to x. Let K be the size of the smallest such term T; i.e., $\mathrm{CER}(f,x)=K$. This implies that the K variables in T intersect all the clauses of f. The argument proceeds in a way similar to the analysis of the greedy algorithm for set cover; that is, we first observe that there is a variable that appears in m/K clauses. If the variable evaluates to 1 then we have cost 1 and eliminate m/K clauses. If the variable is zero then we incur no cost (and might not eliminate any clauses). Therefore, after evaluating at most $K\ln m$ variables to 1 we eliminate all the clauses. This shows that the cost which our algorithm pays according to $cost_1$ is no larger than $\mathrm{CER}(f,x)\ln m$.

Generalizing to a uniform cost function, we would like intuitively to claim that about half of the variables that are evaluated, turn out to be ones. Consider the i-th variable that we evaluate. Given that the value of the function is 1 and any values for the other variables, we claim that the probability that $x_i=1$ is at least $1/2$. To see why this holds, consider x_1. We have two type of inputs for which $f=1$: either $f(0y)=f(1y)=1$ or $f(1y)=1$ and $f(0y)=0$ (here we use the fact that f is monotone). Since we assume that the distribution is uniform, it is clear that the probability that $x_1=1$ given that $f(x)=1$ is at least $1/2$.

To complete the proof, note that since $\Pr[x_i=1 \mid f(x)=1] \geq 1/2$ we obtain that for any strategy A to query the variables and any termination condition, $E_D[cost_1(A)|f(x)=1] \geq (1/2)E_D[cost(A)|f(x)=1]$.

The case that the function is zero is similar (eliminating the terms of the DNF, rather than the clauses of the CNF, when an attribute is zero). □

[4] For uniform costs, the cheapest term (or clause) is the one containing the smallest number of attributes.

3 Adversarial Online

In this section we consider learning in the adversarial online setting. We start by presenting a lower bound, which applies for the case where the target function f is a clause (specifically, the OR of all n variables). We then, show an almost matching upper bound that applies to *all* target functions.

Theorem 3.1. *Let the target function be $f = x_1 \vee \cdots \vee x_n$ and a uniform cost ($c_i = 1$, for all i). For any deterministic online algorithm A for f there is an input sequence such that the cost of A is at least $n/4$ times the cost of the best static permutation.*

Proof. At time t, the online algorithm outputs a permutation π_t (that will be used to observe the attributes of the t-th input assignment). Let F_{π_t} be the set of $n/2$ attributes that appear first in the permutation π_t. The adversary sets the value of attributes in F_{π_t} to 0 and the other attributes to 1. The cost of the online at time t (using π_t) is $n/2$ while the cost of OPT is at most 2 (since a random permutation has expected cost of 2). □

Next, we show that sorting the variables according to their cost (and using this permutation for all inputs), almost achieves the lower bound.

Theorem 3.2. *Let f be a target function and π be a permutation of the attributes according to the cost (from small to large). Then, for any input x, the cost of π satisfies $cost(\pi, x) \leq n \cdot \mathrm{CER}(f, x)$.*

Proof. Consider the application of the evaluation order π to assignment x and fix a lowest cost certificate for x. If there exists a variable x_i in the certificate that was not observed in π, or if x_i was the last variable in π, then the cost of each variable observed in π is at most c_i and so $cost(\pi, x) \leq n \cdot c_i \leq n \cdot \mathrm{CER}(f, x)$. To complete the proof, note that if there is no such variable x_i then this means that our algorithm observed all the variables in the certificate and kept going – in contradiction to the algorithm definition (which says that once $f(x)$ is fixed no more variables are observed). □

4 Online Distribution Model I: Unknown distribution

In this section, we address the case where the distribution D is unknown and arbitrary while the function f, which is a clause, is given. We bound the cost of our algorithm in terms of the expected cost of OPT_D. Therefore we may assume that $f(x) = 1$ for every $x \in D$. For D such that $f(x) = 0$ for some inputs x, our performance relative to the performance of OPT_D only improves since if $f(x) = 0$ every algorithm has to observe all attributes, due to our general requirement to exhibit a clause which is zero in the case that $f(x) = 0$. Our goal is therefore to find an attribute which is one with low cost, assuming that there is such an attribute. The basic approach is to sample from D, estimate the residual costs of the attributes, and run Greedy$'$ using the estimates. However, since some of the attributes may be much more expensive than others, we have to be careful not to pay too much for learning parts of D which are insignificant.

Let Y be a set of attributes and let D_Y be the distribution D conditioned on the attributed in Y being zero and let A be the event (over D_Y) that $x_i = 1$ for some $x_i \notin Y$. By sampling a set S of examples from D_Y we can calculate an estimate $\hat{p}_{D_Y}(S, A, \theta)$ of $p_{D_Y} = \Pr_{D_Y}[A]$ as follows. Let $k = |S \cap A|$ and $m = |S|$. If $k \geq \theta$ then $\hat{p}_{D_Y}(S, A, \theta) = k/m$ and otherwise $\hat{p}_{D_Y}(S, A, \theta) = 0$. More specifically, let $\gamma, \epsilon, \delta < 1$ be parameters (to be fixed latter). Set $\theta = \theta(\epsilon, \delta) = \frac{6 \ln(4/\delta)}{(\epsilon)^2}$ and $L = L(\epsilon, \delta, \gamma) = 2n\theta/\gamma$. By Lemma A.2 if we sample a set S of size L then with probability $1 - \delta$, $\hat{p}_{D_Y}(S, A, \theta)$ is an (ϵ, γ)-estimate of p_{D_Y}.[5] (For the definition of (ϵ, γ)-estimate see Section 2.)

Recall the offline algorithm $\text{Greedy}'(\epsilon, \gamma)$ from Section 2. After fixing the first $k-1$ attributes $x_{g(1)}, x_{g(2)}, \ldots, x_{g(k-1)}$ in its permutation, Greedy' uses (ϵ, γ)-estimates $\hat{p}_i^k$ on the probabilities p_i^k that $x_i \notin \{x_{g(1)}, x_{g(2)}, \ldots, x_{g(k-1)}\}$ is one, given that $x_{g(1)}, x_{g(2)}, \ldots, x_{g(k-1)}$ are all zeros, to choose $x_{g(k)}$.

We now describe a straightforward algorithm that processes examples drawn from D while computing estimates $\hat{p}_i^k$, as required by Greedy', and converging to its permutation g. We call this algorithm the Naive Learning algorithm (NL).

Algorithm NL maintains a sequence of attributes which is a *prefix* of the permutation that it gradually constructs. This prefix is constructed such that it is also a prefix that $\text{Greedy}'(\epsilon, \gamma)$ would have constructed, for some (ϵ, γ)-estimates of the probabilities p_i^k.

The prefix is initially empty. In general assume that the prefix of NL consists of $k-1$ attributes after processing some sequence of examples. For the next example, say e, NL first observes the attributes in the prefix. If NL hits an attribute in the prefix which is one then NL stops. Otherwise, if all attributes in the prefix are zero then NL uses e to estimate p_i^k for every attribute x_i not in the prefix. To do that (and find an attribute which is one at the same time) NL observes the value of each of the remaining attributes in e. When NL had already processed a set S of L examples for which it observed all attributes not in its prefix, NL calculates estimates $\hat{p}_i^k$ for p_i^k, each is an (ϵ, γ)-estimate for p_i^k with probability $1 - \delta$. NL then picks the attribute with smallest residual cost according to its estimates and adds it to its prefix.

Running NL with $\theta(\epsilon, \delta')$, and $L(\epsilon, \delta', \gamma)$ where $\delta' = \delta/(n^3 + n^2 + 1)2^n$ we obtain by the union bound that with probability $1 - \delta$, NL's estimates for all p_i^k's are an (ϵ, γ) estimate, which implies that it is a valid input for $\text{Greedy}'(\epsilon, \delta)$.

The following theorem summarizes the properties of NL, its proof is similar to the proof of Theorem 4.2 and hence omitted.

Theorem 4.1. *Let $s = e_1, \ldots, e_T$ be a sequence of examples to which we apply the algorithm NL. Let s' be the subset of s for which NL observes the attributes that are not in the prefix (notice that, by the definition of NL, we have $|s'| \leq nL$). Suppose that we run NL on S with $\delta' = \delta/(n^3 + n^2 + 1)2^n$. Then, with probability at least $1 - \delta$, the following hold.*

1. *Let $c^{pre}(e_j)$ be the cost paid by NL for observing attributes in the prefix at the time in which it processed e_j. Then, $\sum_{j=1}^{T} c^{pre}(e_j) \leq T \cdot \frac{4(1+\epsilon)}{(1-\epsilon)(1-2\gamma)} \cdot (1 + \epsilon) E_D(\text{OPT}_D)$, where $E_D(\text{OPT}_D)$ is the expected cost of OPT_D on an example drawn from D.*

[5] Using Lemma A.2 and setting $q = \frac{\gamma}{2n}$ and $F = \theta$. The number of samples m is then equal to L.

2. *The cost NL pays for observing additional attributes, for examples in s', is at most $\sum_{i=1}^{n} c_i = O(nc_{max})$ for each example in s', where c_{max} is the cost of the most expensive attribute.*

The weakness of this straightforward approach is that the cost that NL pays on s' (and therefore on the entire sequence) may be very large compared to the cost of OPT_D on the particular sample or even compared to the expected cost of OPT_D on the distribution D. This is because we may be observing an expensive attribute for about nL times while OPT_D may not observe it at all. To be able to relate the learning cost to the cost of OPT_D on the sample, we describe an algorithm which we call the *Greedy Learning (GL)* algorithm. The GL algorithm also uses a permutation which starts with a prefix, that is also a prefix of $\textsc{Greedy}'$. However, in contrast with NL, when GL cannot resolve an example with the attributes in its prefix, it does not observe all other attributes. Instead, GL carefully observes attributes in its suffix one by one, in a particular order, to avoid large cost. Even before defining GL precisely, to contrast its performance with the performance of NL, we state our main theorem.

Theorem 4.2. *Let $s = e_1, \ldots, e_T$ be a sequence of examples when a round of GL ends, and let s' be the subset of s for which GL observed attributes that are not in the prefix (as we shall see, from the definition of GL would follow that $|s'| \leq n^2 L$). Suppose we run GL on s with $\delta' = \delta/(n^3+n^2+1)2^n$. Then, with probability at least $1-\delta$, the followings hold:*

1. *Let $c^{pre}(e_j)$ be the cost paid by GL for observing attributes in its prefix while processing e_j. Then, $\sum_{j=1}^{T} c^{pre}(e_j) \leq T \cdot (1+\epsilon) E_D(\textsc{Greedy}'(\epsilon,\gamma)) \leq T \cdot \frac{4(1+\epsilon)}{(1-\epsilon)(1-2\gamma)} \cdot (1+\epsilon) E_D(\mathrm{OPT}_D)$.*

2. *The cost GL pays for observing additional attributes, for examples in s', is at most $O(n^2)$ times the cost paid by OPT_D on the examples in s'.*

It is worthwhile to compare Theorem 4.1 and Theorem 4.2. The main and most important difference is in the second guarantee. While NL has a guarantee of $O(n^2 c_{max} L)$ on its cost for processing the set s', algorithm GL has a guarantee of $O(n^2)$ times the cost of OPT_D to process s'. This is a crucial difference, showing that GL is as good as OPT_D, up to a factor which is independent of the costs. (A minor issue is that the set s' is slightly larger in GL.)

4.1 The Greedy Learning algorithm: GL

The *Greedy Learning (GL)* algorithm maintains a permutation of the attributes partitioned into two subsequences called *prefix* and *suffix* and one additional attribute in between the prefix and the suffix which is the *currently explored attribute*. The order of attributes in the prefix is determined by the algorithm. The attributes in the suffix are sorted by their costs. Each attribute in the suffix is marked either as "explored" or "unexplored".

The GL algorithm progresses in *phases*, where each phase is divided to *rounds*. In phase k, let $\mathcal{P}^k$ be the attributes in the prefix, and let $\mathcal{S}^k$ denote the set of the remaining attributes. At round j of this phase, the explored attributes are $\mathcal{E}^{k,j}$, and the currently explored attribute is $a^{k,j}$, where $a^{k,j} \notin \mathcal{E}^{k,j}$ (the currently explored attribute does not change during a round). The permutation

$\pi^{k,j}$ in round j of phase k has first the attributes in the prefix $\mathcal{P}^k$, in the order that they were added to the prefix, followed by $a^{k,j}$ followed but the attributes in $\mathcal{S}^k \setminus \{a^{k,j}\}$, ordered by increasing costs.

In phase 0, we have an empty prefix, i.e., $\mathcal{P}^0 = \emptyset$, and all the attributes are marked "unexplored". In the first round (of phase 0) the currently explored attribute, $a^{0,1}$, is the cheapest attribute. The permutation $\pi^{0,1}$ starts with $a^{0,1}$ followed by the remaining attributes sorted by their cost. In general, in round j of phase k, we use permutation $\pi^{k,j}$ until we observe the currently explored attribute $a^{k,j}$ in $L(\epsilon, \delta', \gamma)$ examples (where ϵ, δ' and γ are parameters that will determine the performance as in Theorem 4.2). After GL observed $a^{k,j}$ in $L(\epsilon, \delta', \gamma)$ examples, round j terminates, $a^{k,j}$ is marked "explored", and $\mathcal{E}^{k,j+1} = \mathcal{E}^{k,j} \cup \{a^{k,j}\}$.

When round j terminates, we calculate an estimate to the probability of the event that all attributes in $\mathcal{P}^k$ are zero and $a^{k,j} = 1$ as suggested at the beginning of Section 4. we first check the number of times b, that $a^{k,j}$ was true (out of the L times it was observed). If $b < \theta$, we estimate $\hat{p}_i^k = 0$; otherwise, we estimate $\hat{p}_i^k = b/L$. We also set our estimate of the residual cost of attribute $a^{k,j}$ to be $\hat{\mu}_i^k = c_i/\hat{p}_i^k$, if $\hat{p}_i^k \leq 1$, and $\hat{\mu}_i^k = \infty$ otherwise. This completes a round.

When a round is completed we test if the phase terminates, or we should continue to the next round of the same phase. We calculate the minimum among our estimates of the residual costs of the explored attributes in the phase, i.e., attributes in $\mathcal{E}^{k,j+1}$, denote it by $\hat{w}^{k,j} = \min_{i \in \mathcal{E}^{k,j+1}} \hat{\mu}_i^k$. We also calculate the minimum cost among the costs of all unexplored attributes in the suffix, denote it by $w^{k,j} = \min_{i \in \mathcal{S}^k - \mathcal{E}^{k,j+1}} c_i$. (If $\mathcal{S}^k - \mathcal{E}^{k,j+1} = \emptyset$ then we set $w^{k,j} = \infty$.

If both $w^{k,j} = \infty$ and $\hat{w}^{k,j} = \infty$ then we terminate with in a `failure` state. (We show that this happens with very low probability.)

If $\hat{w}^{k,j} > w^{k,j}$ then we set $a^{k,j+1}$ to be the unexplored attribute in the suffix with the cheapest cost, i.e., let $\ell = \arg\min_{i \in \mathcal{S}^k - \mathcal{E}^{k,j+1}} c_i$, and $a^{k,j+1} = x_\ell$. We continue to round $j+1$ of phase k.

If $\hat{w}^{k,j} \leq w^{k,j}$ we add to the prefix the explored attribute with the smallest residual cost in the suffix. I.e., $\mathcal{P}^{k+1} = \mathcal{P}^k \cup \{x_a\}$, where $a = \arg\min_{i \in \mathcal{E}^{k,j+1}} \hat{\mu}_i^k$, and $\mathcal{S}^{k+1} = \mathcal{S}^k \setminus \{x_a\}$. This completes phase k, and we move to phase $k+1$. At the beginning of phase $k+1$, we mark all elements that are not in $\mathcal{P}^{k+1}$ as "unexplored" and make the cheapest one the currently explored attribute.

Algorithm GL, with probability $1-\delta$, simulates the algorithm Greedy$'$, described in Section 2, where at each iteration we try to avoid estimating the p_i's of expensive attributes, when knowing they cannot change the decision of Greedy$'$. Specifically, when phase k terminates we have estimates for for p_i^k's of the explored attributes in the suffix. If there are unexplored attributes in the suffix when we add the next attribute to the prefix, then it is guaranteed that, whatever the p_i^k of those attributes are, Greedy$'$ could have taken the same decision.

4.2 The analysis of GL

We assume that we run GL on a sequence $\Xi = e_1, \ldots, e_T$ and after e_T a round of GL ends. We first show that all our estimates are (ϵ, γ)-estimates, with high probability. To establish the first part of Theorem 4.2, we show that, the prefix of GL is in fact a prefix that could have obtained by

GREEDY$'(\epsilon,\gamma)$, and we also show that the cost of such prefix while processing $e_1,\ldots,e_T$ is close to T times its expected cost on an example from D. Then, to establish the second part of Theorem 4.2, we show that GL does not observe attributes which are too expensive even for those examples for which the attributes in the prefix are all zero.

For a set of attributes Y, we recall that D_Y is the distribution D conditioned on attributes in Y being zero. Let S be a sample of examples from D_Y such that $m = |S|$. Recall that for an event A we define the estimate $\hat{p}_{D_Y}(S,A,\theta)$ of $\Pr_{D_Y}[A]$ as follows. Let $k = |S \cap A|$. If $k \geq \theta$ then $\hat{p}_{D_Y}(S,A,\theta) = k/m$ and otherwise $\hat{p}_{D_Y}(S,A,\theta) = 0$. The following lemma follows directly from Lemma A.2.

Lemma 4.3. *Let $\theta = \theta(\epsilon,\delta')$ and $L = L(\epsilon,\delta',\gamma)$. Let Y be a set of attributes, and let S be a random sample from D_Y such that $|S| \geq L$. Let $x_i \notin Y$, and let A be some arbitrary over D_Y. Then with probability $1-\delta'$, $\hat{p}_{D_Y}(S,A,\theta)$ is an (ϵ,γ)-estimate of $\Pr_{D_Y}[A]$.*

Combining this lemma with the union bound we obtain the following lemma and its corollary.

Lemma 4.4. *Let $\theta = \theta(\epsilon,\delta')$ and $L = L(\epsilon,\delta',\gamma)$. Let $S^{j,k}$ be the L examples drawn during round j of phase k. Let $Y = \mathcal{P}^k$, let $x_j \notin Y$ and let A be the event, over D_Y, that $x_j = 1$. Then with probability $1 - n2^n\delta'$, $\hat{p}^k_j = \hat{p}_{D_Y}(S^{j,k},A,\theta)$ is an (ϵ,γ)-estimate of $p^k_j = \Pr_{D_Y}[A]$.*

Proof. By the union bound and Lemma 4.3, with probability $1-n2^n\delta'$, for every set Y, and any attribute $x_j \notin Y$, if we sample a set $S^{j,k}$ of L examples from D_Y then $\hat{p}_{D_Y}(S,A,\theta)$ is an (ϵ,γ)-estimate of $\Pr_{D_Y}[A]$, where $A = \{x \mid x_j = 1\}$.

Therefore regardless oft the specific set $Y = \mathcal{P}^k$, which is determined by the prefix of $\Xi = e_1,\ldots,e_T$ up to the beginning of phase k, for every attribute $x_j \notin Y$, with probability $1-n2^n\delta'$, $\hat{p}^k_j$ is an (ϵ,γ)-estimate of p^k_j. □

Given that we always have (ϵ,γ)-estimates, we can deduce that the run of GL is equivalent to that of GREEDY$'(\epsilon,\gamma)$.

Corollary 4.5. *Let $\delta' = \frac{\delta}{(n^3+n^2+1)2^n}$, $\theta = \theta(\epsilon,\delta')$ and $L = L(\epsilon,\delta',\gamma)$. Then with probability $1-n^3 2^n\delta'$, at any phase k, the prefix $\mathcal{P}^k$ of the permutation maintained by algorithm GL is a prefix of* GREEDY$'(\epsilon,\gamma)$.

Proof. Lemma 4.4 and the union bound (over the at most n phases, and the at most n rounds of each phase) imply that the estimate, $\hat{p}^k_j$, on the probability, p^k_j, that $a_{j,k} = 1$ given that attributes in $\mathcal{P}^k$ are zero , is an (ϵ,γ)-estimate for every round j and phase k, with probability $1-n^3 2^n\delta'$.

Furthermore, residual costs of unexplored attributes in phase k are guaranteed to be larger than the ones for the explored attributes. Therefore the attribute that GL adds to the prefix at the end of phase k is the same as the attribute that GREEDY$'$ would have added for the same (ϵ,γ)-estimates.

Note that if we always have (ϵ,γ)-estimates then we never have $w^{k,j} = \infty$, and therefore the algorithm does not reach a `failure` state. □

Let $c(e_j)$, $1 \le j \le T$, be the cost of GL for processing example e_j. We partition $c(e_j)$ into three components $c(e_j) = c^{pre}(e_j) + c^{cur}(e_j) + c^{suf}(e_j)$. The first part $c^{pre}(e_j)$ is the cost paid by GL for observing attributes in the prefix while processing e_j. The second part $c^{cur}(e_j)$ is the cost paid by GL if it observed the currently explored attribute while processing example e_j, otherwise $c^{cur}(e_j) = 0$. The third part $c^{suf}(e_j)$ is the cost incurred by GL for observing attributes in the suffix while processing e_j if GL observes attributes in the suffix while processing e_j, otherwise $c^{suf}(e_j) = 0$. Let $E^{pre} = \sum_j c^{pre}(e_j)$, $E^{cur} = \sum_j c^{cur}(e_j)$, and $E^{suf} = \sum_j c^{suf}(e_j)$. We bound each of E^{pre}, E^{cur}, and E^{suf} in the remainder of the section. The next Lemma bounds E^{pre}, and by combining it with Theorem 2.4, we establish the first part of Theorem 4.2.

Lemma 4.6. *If we use GL with $\delta' = \delta/(n^3 + n^2 + 1)2^n$ then, with probability $1 - 2^n\delta'$ over the choice of S, we have $E^{pre} \le T(1+\epsilon)E_D(\text{GREEDY}')$.*

Proof. Let $P = x_{i_1}, \ldots, x_{i_k}$ be the prefix of GL after processing the last example e_T. Let $\overline{c}^{pre}(e_i)$ be the cost we pay if we serve e_i using P. Let $\overline{E}^{pre} = \sum_{i=1}^{T} \overline{c}^{pre}(e_i)$. Since the prefix while processing any example e_j is a prefix of P, $E^{pre} \le \overline{E}^{pre}$. So it suffices to show that $\overline{E}^{pre} \le T(1+\epsilon)E_D(\text{GREEDY}')$.

Let $P^\ell = x_{i_1}, \ldots, x_{i_\ell}$, the prefix of P of length ℓ, and let $\hat{q}_\ell$, $0 \le \ell \le k-1$, be the fraction of the examples in $\Xi = e_1, \ldots, e_T$ for which the attributes in P^ℓ are all zero. Clearly $\overline{E}^{pre} = T\sum_{\ell=1}^{k-1} \hat{q}_\ell c_{i_{\ell+1}}$. Let q_ℓ, $0 \le \ell \le k-1$, be the probability over D that $x_{i_1}, \ldots, x_{i_\ell}$ are zero, and let $E_D(P)$ be the expected cost of applying P to an example from D, that is $E_D(P) = \sum_{\ell=1}^{k-1} q_\ell c_{i_{\ell+1}}$.

Let Z be an arbitrary set of attributes and let A the event that all attributes in Z are zero. Using Lemma 4.3, and the union bound, we obtain that with probability $1 - 2^n\delta'$ for every such set Z, $\hat{p}_D(\Xi, A, \theta)$ is (ϵ, γ)-estimate of $\Pr_D[A]$. In particular we obtain that with probability $1 - 2^n\delta'$, we have $\hat{q}_\ell \le (1+\epsilon)q_\ell$ for every ℓ. This shows that $\overline{E}^{pre}$ is bounded by $(1+\epsilon)TE_D(P)$. Corollary 4.5 implies that P is also a prefix of GREEDY$'$ so the lemma follows. □

Our last two lemmas establish the second part of Theorem 4.2. Let $E^{cur} = \sum_k E_k^{cur}$ where E_k^{cur} is the contribution to E^{cur} of the cost of observing the currently explored attribute for examples during phase k. Let $S(k)$ be the set of examples for which we observed the currently explored attribute during phase k. Assume we explore $\zeta(k) > 1$ attributes during phase k. Then, since we observe each such attribute exactly L times, the size of $S(k) = \cup_{j\in[1,\zeta(k)]} S^{k,j}$ is $\eta(k) = |S(k)| = \zeta(k)L$. Let $x_{\ell e(k)}$ be the last ($\zeta(k)$-th) and therefore the most expensive attribute that GL explores at phase k. Then $E_k^{cur} \le \eta(k)c_{\ell e(k)}$. Let $\text{OPT}_D(S(k))$ be the cost of OPT_D when it processes the examples in $S(k)$. The following lemma shows that with probability at least $1 - n^2 2^n\delta'$, for every phase k, we have $\eta(k)c_{\ell e(k)} \le \frac{n^2\text{OPT}_D(S(k))}{(1-\epsilon)^2}$.

Lemma 4.7. *Let $S(k)$ denote the set of examples for which GL observes the currently explored attribute during phase k, and let $\eta(k) = |S(k)|$. Let $x_{\ell e(k)}$ be the most expensive attribute which GL explores during phase k. Then, with probability at least $1 - n^2 2^n\delta'$, for every phase k,*

$$\eta(k)c_{\ell e(k)} \le \frac{n^2\text{OPT}_D(S(k))}{(1-\epsilon)^2} \ ,$$

where $\mathrm{OPT}_D(S(k))$ *is the cost of* OPT_D *when it processes the examples in* $S(k)$.

Proof. Recall that $\mathcal{P}^k$ denotes the prefix of the permutation of GL in phase k. Let $D_{\mathcal{P}^k}$ be the distribution D conditioned on examples for which all attributes in $\mathcal{P}^k$ are zero. Consider using OPT_D on examples drawn from $D_{\mathcal{P}^k}$. Let $x_o \notin \mathcal{P}^k$ be the first attribute among $x_{opt_D(1)}$, $x_{opt_D(2)}$, $\ldots$ $x_{opt_D(n)}$, that is one with probability at least $\frac{1}{n}$ over $D_{\mathcal{P}^k}$. (There must be such attribute since, in order to maximize the difference between optimal algorithm and the learning algorithm, we assumed that $f(x) = 1$ for every x that we can draw from D.) That is $\Pr_{D_{\mathcal{P}^k}}[x_o = 1] \geq 1/n$

By Lemma 4.4, with probability $1-n2^n\delta$, we have that $\hat{p}^k_o(S(k), \{x_o = 1\}, \theta)$ is an (ϵ, γ)-estimate of $\Pr_{D_{\mathcal{P}^k}}[x_o = 1]$. It follows that with probability $1-n^2 2^n\delta$, for every k, the fraction of the examples in $S(k)$ that OPT_D processes by observing x_o is at least $(1-\epsilon)\frac{1}{n}$. So the cost of OPT_D, just on the examples in $S(k)$, is at least $\eta(k)(1-\epsilon)c_o/n$.

If $c_{\ell e(k)} \leq c_o$ the lemma follows. Otherwise we claim that $c_{\ell e(k)} \leq nc_o/(1-\epsilon)$. By the definition of the algorithm GL if $c_{\ell e(k)} > c_o$ then c_o is explored before $c_{\ell e(k)}$. Since x_o is true for a fraction $\geq 1/n$ of $D_{\mathcal{P}^k}$ then, by Lemma 4.4, when GL explores x_o in phase k, GL estimates its residual probability to be at least $(1-\epsilon)/n$. Therefore the residual cost of x_o after exploring it is at most $nc_o/(1-\epsilon)$. It follows from the definition of our algorithm that $c_{\ell e(k)} \leq nc_o/(1-\epsilon)$ as otherwise we would not explore it. □

Last we bound E^{suf}. Notice that for each example j, $1 \leq j \leq T$, such that $c^{suf}(e_j) > 0$, OPT_D must also observe an attribute in the suffix. This is since, by the definition of GL, all attributes in the prefix and the currently explored attribute at time j are zero. By Theorem 3.2, for each j such that $c^{suf}(e_j) > 0$, we have $c^{suf}(e_j) \leq n\mathrm{OPT}_D(e_j)$, where $\mathrm{OPT}_D(e_j)$ is the cost incurred by OPT_D when processing e_j. This establishes the following lemma.

Lemma 4.8. *Let* Ξ^{suf} *be the set of examples for which GL observes attributes in the suffix. The cost* E^{suf} *is at most* n *times the cost of* OPT_D *on* Ξ^{suf}. □

Combining Lemmas 4.6, 4.7, and 4.8 derives Theorem 4.2.

5 Online Distribution Model II: Unknown Target Function

In this section, we assume that the target function f is a monotone clause which is not known to the algorithm, and that the distribution from which examples are drawn is a product distribution D, which is given to the algorithm. At each time step t, our online algorithm specifies a hypothesis h^t and a way to evaluate h^t on the next input, z^t (which is drawn from D).[6] At the end of each step, the algorithm observes the correct value of the target function, i.e., $f(z^t)$. When evaluating the performance of an online algorithm, we consider two different measures: the first measure is the number of mistakes, i.e., $|\{t \mid h^t(z^t) \neq f(z^t)\}|$, and the second measure is the average cost per example.

[6]In our final algorithm, ON_APPROX, the hypothesis h^t will also be a monotone clause; this however is not a requirement of the model.

There is a clear tradeoff between the two measures. An extreme trivial solution, is not to observe *any* attribute and output a random guess! The cost is zero but the error rate is $1/2$ which is undesirable.[7] In the other extreme, one can use the best learning algorithm for the class in question (monotone clauses, in our case) regardless of the cost of observing the examples. In this section, we aim at getting a tradeoff between the two extremes. Moreover, we would like our number of mistakes to be independent of the number of time steps (specifically, the number of mistakes will be either $O(n)$ or $O(n \log n)$, where n is the number of variables, which is optimal) while keeping the cost near optimal.

Before describing our algorithm, we describe a standard procedure which is widely used in online learning of monotone clauses. The procedure $\textsc{Elim}(z^t, h^t)$ is performed when $f(z^t) = 0$ and $h^t(z^t) = 1$. In such a case $\textsc{Elim}(z^t, h^t)$ returns the set of variables in the clause h^t which are TRUE, i.e., $\textsc{Elim}(z^t, h^t) = \{x_i \in h^t \mid z_i^t = 1\}$ (and we say that $\textsc{Elim}$ is *applicable*). Note that we are guaranteed that the variables in $\textsc{Elim}(z^t, h^t)$ are not in the target function f. This implies that we can "eliminate" any such variable. Clearly, the maximum number of times we can apply $\textsc{Elim}$ is n.

We assume first that our online algorithm $\textsc{Online}(q, D)$ receives as an input, in addition to the product distribution $D = (p_1, \ldots, p_n)$, also the probability q such that $q \stackrel{\text{def}}{=} \Pr[f = 1]$; later we will show how to eliminate this assumption. Let X^t be the set of variables that have not been eliminated before time t. (Initially X^1 includes all n variables.) To specify h^t, let $x_{i_1}, \ldots, x_{i_\ell}$ be the variables in X^t sorted by increasing value of $\mu_{i_j} = c_{i_j}/p_{i_j}$ (breaking ties arbitrarily)[8]. Let $H_{k,r}^t \stackrel{\text{def}}{=} \vee_{j=1}^{k-1} x_{i_j} \vee (x_{i_k} \wedge Br(r))$, where $Br(r)$ is a Bernoulli random variable whose probability of 1 is r (and the probability of 0 is $1 - r$). The evaluation of $H_{k,r}^t$ is done by observing the variables x_{i_1} to $x_{i_{k-1}}$ in order. If any of these variables is 1 then $H_{k,r}^t$ terminates with output 1. If they are all zero then it flips a coin with bias r. If the coin comes up 1 it observes x_{i_k} and returns its value and otherwise it returns zero. The hypothesis h^t is the unique $H_{k,r}^t$ such that $\Pr[H_{k,r}^t = 1] = q$.

Lemma 5.1. *For any t, there is a unique pair (k, r) such that* $\Pr[H_{k,r}^t = 1] = q$.

Proof. Clearly, as we vary k from 1 to ℓ and r from 0 to 1, then $\Pr[H_{k,r} = 1]$ gets any value in the range $[0, q']$, where $q' = \Pr[\vee_{j=1}^{\ell} x_{i_j}]$. Since X^t contains all the variables in f, q is in this range, so there are some k and r which satisfy the lemma. The uniqueness follows from the observation that $\Pr[H_{k,r} = 1]$ is monotone in both parameters. $\square$

An online algorithm is *balanced* if, at every time step t, the hypothesis h^t satisfies $\Pr[h^t = 1] = \Pr[f = 1]$. (Algorithm $\textsc{Online}(q, D)$ is balanced by definition.) For a balanced online algorithm, given that at time t a mistake occurred, i.e., $f(z^t) \neq h^t(z^t)$, the probability that $\textsc{Elim}(z^t, h^t)$ is applicable is exactly $1/2$. This implies that the expected number of mistakes is $O(n)$.

[7] i.e., at each step t the expected number of mistakes is $t/2$; in general, the goal of online algorithms is obtaining a bound on the number of mistakes which is *independent* of t; such a bound is not given by this trivial algorithm.

[8] It is assumed, without loss of generality, that $p_{i_j} > 0$; otherwise, the variable x_{i_j} can be ignored. The motivation for using this order comes from Theorem 2.1.

Claim 5.2. *The expected number of mistakes of a balanced online algorithm is at most $O(n)$.*

Proof. For a balanced online algorithm, we have that $\Pr[f = 1] = \Pr[h^t = 1]$, for any time t. This implies that $\Pr[f = 0, h^t = 1]$=$\Pr[f = 1, h^t = 0]$. To see this, note that $\Pr[f = 1] = \Pr[f = 1, h^t = 1] + \Pr[f = 1, h^t = 0]$ and $\Pr[h^t = 1] = \Pr[f = 1, h^t = 1] + \Pr[f = 0, h^t = 1]$.

This implies that, conditioned on the event that an error occurred, the probability that $\textsc{Elim}(z^t, h^t)$ is applicable is half. Since, as argued above, $\textsc{Elim}$ cannot be applied more than n times, the lemma follows. □

Let $\mathrm{OPT}(f, D)$ be the optimal evaluation of f, with respect to a distribution D. (Recall that, by Theorem 2.1, when D is a product distribution then the optimal evaluation sorts the variables in f according to $\mu_i = c_i/p_i$.) The next lemma bounds the expected cost of $\textsc{Online}(q, D)$ at any time t.

Lemma 5.3. *Let D be a product distribution, and let $h^t = H^t_{k,r}$ be such that $\Pr[h^t = 1] = \Pr[f = 1] = q$. Then, $E[cost(\mathrm{OPT}(f, D))] \geq E[cost(h^t)]$.*

Proof. To simplify notation, assume that the variables $x_{i_1}, \dots, x_{i_\ell}$ in X^t when sorted by increasing μ_{i_j}, are in fact $x_1, \dots, x_\ell$. That is $h^t = H^t_{k,r} = \vee_{j=1}^{k-1} x_j \vee (x_k \wedge Br(r))$. We can write the expected cost of h^t as,

$$\begin{aligned}
E[cost(h^t)] &= \sum_{i=1}^{k-1} c_i \prod_{j=1}^{i-1}(1 - p_j) + rc_k \prod_{j=1}^{k-1}(1 - p_j) \\
&= \sum_{i=1}^{k-1} \mu_i p_i \prod_{j=1}^{i-1}(1 - p_j) + r\mu_k p_k \prod_{j=1}^{k-1}(1 - p_j) \ .
\end{aligned}$$

We can visualize $E[cost(h^t)]$ as an integral of a step function $\eta(x)$, whose values are μ_i. More precisely, let $q_0 = 0$, and let $q_m = \sum_{i=1}^{m} p_i \prod_{j=1}^{i-1}(1 - p_j)$, for $1 \leq m \leq k$. Note that q_m is the probability that one of the variables $x_1, \dots, x_m$ is one. For $x \in (q_{i-1}, q_i]$, where $1 \leq i \leq k-1$, we have $\eta(x) = \mu_i$; for $x \in (q_{k-1}, q_{k-1} + r(q_k - q_{k-1})]$, we have $\eta(x) = \mu_k$; and for $x \in (q_{k-1} + r(q_k - q_{k-1}), 1]$, we have $\eta(x) = 0$. This implies that

$$\int_0^1 \eta(x)dx = E[cost(h^t)].$$

Similarly, we can write the expected value of the optimal evaluation of f as an integral of $\eta_f(x)$, which is a step function defined analogously to $\eta(x)$ but with respect to $\mathrm{OPT}(f, D)$. Since $\Pr[h^t = 1] = \Pr[f = 1]$, we have $\eta_f(x) \neq 0$ for any x such that $\eta(x) \neq 0$. Furthermore, since X^t contains all the variables in f, and both h^t and the optimal evaluation of f order their variables by increasing value of μ_i, it follows that for every x such that $\eta(x) \neq 0$ we have $\eta(x) \leq \eta_f(x)$. Thus, $\eta(x) \leq \eta_f(x)$ for every x and $E[cost(h^t)] \leq E[cost(\mathrm{OPT}(f, D))]$. □

The following theorem now follows from Claim 5.2 and Lemma 5.3.

Theorem 5.4. *Let D be a product distribution. The expected number of mistakes of Algorithm* ONLINE(q, D) *is $O(n)$ and at any time t we have $E(cost(h^t)) \leq E[cost(\mathrm{OPT}(f, D))]$, where h^t is the hypothesis of* ONLINE(q, D) *at time t (and its evaluation order). (The expectation is over the choice of an example drawn from the product distribution D.)* □

Next, we extend the result to the case where $q = \Pr[f = 1]$, the probability that the target function is one, is unknown to the algorithm. We modify the algorithm (and its analysis) and show that the same goal can be achieved, with a slight increase in the cost and $O(n \log n)$ mistakes.

We can naively try and estimate the probability that $f = 1$, and determine it to some accuracy λ. However, if we overestimate the probability, we might need to use a very expensive attribute (which OPT never uses) and our expected cost might be far from OPT. If we consistently underestimate the probability, we might continuously make mistakes. At the hart of our algorithm is the delicate balancing between avoiding excessive mistakes and keeping the expected cost near optimal. This is achieved by, generally, trying to underestimate the probability, while when adding an attribute making sure that "most" of its cost could be justified (even if not all of it).

The algorithm ON_APPROX$(\bar{\epsilon}, \delta, D)$ receives as a parameter $\bar{\epsilon} \in (0, 1/4]$, a confidence parameter $0 < \delta < 1$, and a product distribution D. The algorithm will try to learn a monotone clause f from examples drawn from D. The parameter $\bar{\epsilon}$ controls the tradeoff between the number of mistakes that ON_APPROX$(\bar{\epsilon}, \delta, D)$ makes to its expected cost. For larger $\bar{\epsilon}$ we will make less mistakes but the cost would be larger relative to the optimal cost. The parameter δ is the probability that ON_APPROX$(\bar{\epsilon}, \delta, D)$ would not behave as we expect ("fail"). Specifically, with probability $1 - \delta$ ON_APPROX$(\bar{\epsilon}, \delta, D)$ will indeed converge to the function f if given enough examples from D, while making at most $O(\frac{n}{\bar{\epsilon}^2} \log(\frac{n}{\delta}))$ mistakes and having expected cost which is larger than the optimal expected cost by at most a factor of $(1 + \bar{\epsilon})$.

The algorithm ON_APPROX$(\bar{\epsilon}, \delta, D)$ runs in phases and maintains a set X of variables that have not been eliminated. At phase k, we have a hypothesis h^k which is used throughout this phase, and the evaluation of h^k is done by sorting the variables that it contains according to $\mu_i = c_i/p_i$. We start the first phase with $h^1 = \emptyset$. Phase k terminates in one of two ways:

1. There is a mistake on an example x where $f(x) = 0$ and $h^k(x) = 1$. We call such a mistake an *eliminating mistake* since ELIM(x, h^k) will eliminate at least one variable from h^k. We call a phase that terminates due to an eliminating mistake an *eliminating phase*.

   When an eliminating phase k terminates with an eliminating mistake x then we set $X = X \setminus \mathrm{ELIM}(x, h^k)$ and $h^{k+1} = h^k \cap X$.

2. During phase k, there has been $\alpha = (\gamma/\epsilon^2) \log 4n/\delta$ mistakes with $f(x) = 1$ and $h^k(x) = 0$ (where $\epsilon = \bar{\epsilon}/16$ and $\gamma \geq (1 + \epsilon)$ is a fixed constant). We call such a phase a *growing phase*.

   We then proceed to phase $k + 1$. Consider a growing phase k that terminates. Let L_k be the total number of steps in phase k and $\hat{q}_k = \alpha/L_k$. Let $Z_k = \{x_i \in X \mid p_i \leq (1 + \epsilon)\hat{q}_k/\Pr[h = 0]\} \setminus h^k$. If Z_k is empty then the algorithm continues with $h^{k+1} = h^k$ (we will show that this occurs with negligible probability). If Z_k is not empty then let $x_{i_{k+1}} = \arg\min_{x_i \in Z_k} \mu_i = c_i/p_i$. We set $h^{k+1} = h^k \vee x_{i_{k+1}}$ and proceed to phase $k+1$.

As before, there may be at most n eliminating phases since, after each such phase, we eliminate at least one variable that cannot get back into h. We say that a growing phase k *succeeds* if $Z_k \neq \emptyset$ and thereby $h^k \subsetneq h^{k+1}$. Clearly, there are at most n growing phases that succeed. We will show that with probability $1-\delta$ all growing phases succeed and therefore there are at most $2n$ phases altogether (Theorem 5.10 below). In fact, it suffices to show that the first n growing phases succeed with probability $1-\delta$, since after n successful growing phases each variable either has been eliminated or is in h, so we cannot get any more mistakes where $f(x) = 1$ and $h(x) = 0$.

Once we establish that with probability $1-\delta$ there are only $2n$ phases then, by the definition of the algorithm, it makes at most $O(n\alpha) = O(\frac{n}{\epsilon^2}\log(\frac{n}{\delta}))$ mistakes.

We focus now on a particular phase k. Denote by M^k the event that we obtain a mistake; that is, the event that for an example x drawn from D, $f(x) \neq h^k(x)$. Denote by M_e^k the event that $f(x) = 0$ and $h^k(x) = 1$, in which case we say that x is an eliminating example (recall that if this event happens then the phase ends with an elimination). Denote by M_{ne}^k the event that $f(x) = 1$ and $h^k(x) = 0$. When we focus on a particular phase k, we sometimes drop the superscript k when it is clear from the context and use M, M_e, and M_{ne} to denote the three events.

The next lemma shows that if there is a "significant" probability of getting an eliminating mistake in a phase, then indeed the phase is "very likely" to end with an elimination. Specifically,

Lemma 5.5. *If* $\Pr[M_e^k \mid M^k] \geq \epsilon^2$ *then with probability at least* $1-\delta/4n$ *phase* k *ends with an elimination.*

Proof. Since $\Pr[M_e^k \mid M^k] \geq \epsilon^2$, then the probability of not having an M_e^k event while having α mistakes is at most $(1-\epsilon^2)^\alpha \leq e^{-\epsilon^2\alpha} \leq \delta/4n$. ☐

We call a phase k where $\Pr[M_e^k \mid M^k] \geq \epsilon^2$ *heavy*; otherwise, we call phase k *light*. Lemma 5.5 asserts that a heavy phase is likely to be eliminating. Specifically,

Lemma 5.6. *With probability* $1-\delta/2$*, all heavy phases, among the first* $2n$ *phases, are eliminating phases.*

Proof. By Lemma 5.5, a heavy phase does not end with an elimination with probability $\leq \delta/4n$. So by the union bound the probability that at least one of the first $2n$ phases which is heavy is not eliminating is $\leq \delta/2$. ☐

Next, we focus on the case of a light growing phase and show that such a phase is likely to succeed (in the sense defined above). Since in a light phase $\Pr[M_e \mid M] < \epsilon^2$, it follows that the probability of a mistake (i.e., the event M) in such a phase is similar to the probability of the event M_{ne}. Specifically, let k be a light phase and let $p = \Pr[M^k]$, $p_{ne} = \Pr[M_{ne}^k]$, and $p_e = \Pr[M_e^k]$. Since the phase is light, we have $p_e \leq \epsilon^2 p \leq \epsilon^2$ and similarly $(1-\epsilon^2)p \leq p_{ne} \leq p$. Let $\beta = \Pr[M_{ne} \mid \overline{M_e}]$, where $\overline{M_e}$ is the event that the example is not eliminating (i.e., the example is either not a mistake or a mistake which is not an eliminating mistake). Since $M_{ne} \subseteq \overline{M_e}$, we get $\beta = \Pr[M_{ne}]/\Pr[\overline{M_e}] = p_{ne}/(1-p_e)$. Combining the last inequalities,

$$(1-\epsilon^2)p \leq p_{ne} \leq \beta = \frac{p_{ne}}{1-p_e} \leq \frac{p_{ne}}{(1-\epsilon^2)} \leq \frac{p}{(1-\epsilon^2)} \ , \tag{1}$$

and so

$$(1-\epsilon^2)\beta \leq p \leq \frac{\beta}{1-\epsilon^2} \leq (1+2\epsilon^2)\beta, \tag{2}$$

for $\epsilon \leq \sqrt{2}/2$.

Recall that $\hat{q}_k = \alpha/L_k$, where L_k is the length of the phase. The following lemma shows that β (and therefore also p and p_{ne}) are close to $\hat{q}_k$ with high probability.

Lemma 5.7. *Let k be a growing phase then with probability $1-\delta/2n$, $|\beta - \hat{q}_k| \leq \epsilon\hat{q}_k$.*

Proof. Let Y_i, $i = 1, \ldots, \alpha$, be the number of steps between mistake $i-1$ to mistake i in phase k. Each random variable Y_i is distributed Geometrically with parameter β. Therefore, $E[Y_i] = 1/\beta$ and $E[Y] = \alpha/\beta$, where $Y = \sum_{i=1}^{\alpha} Y_i$. Notice that $Y = L_k$. We first bound the probability that $Y \geq (1+\epsilon)\alpha/\beta$, and then bound the probability that $Y \leq (1-\epsilon)\alpha/\beta$.

For the first bound, consider $(1+\epsilon)\alpha/\beta$ Bernoulli random variables each with probability β to succeed; the probability of less than α successes is exactly the probability that $Y \geq (1+\epsilon)\alpha/\beta$. Using Theorem A.1(1) (note that $pm = (1+\epsilon)\alpha$ and substitute $\beta = \frac{\epsilon}{1+\epsilon}$), the probability of less than α successes is smaller than $e^{-\gamma\log(4n/\delta)/(1+\epsilon)} \leq \frac{\delta}{4n}$ for $\gamma \geq 1+\epsilon$.

Similarly, when we consider $(1-\epsilon)\alpha/\beta$ Bernoulli random variables each with success probability β, the probability of more than α successes is exactly the probability that $Y \leq (1-\epsilon)\alpha/\beta$. By Theorem A.1(2), this event happens with probability at most $\frac{\delta}{4n}$.

By combining the previous two bounds, with probability $1-\delta/2n$, we have $|Y - \alpha/\beta| \leq \epsilon\alpha/\beta$. Since $Y = L_k$ and $\hat{q}_k = \alpha/L_k$ it follows that with probability $1-\delta/2n$

$$|\beta - \hat{q}_k| = \left|\beta - \frac{\alpha}{L_k}\right| = \frac{|\beta L_k - \alpha|}{L_k} \leq \frac{\epsilon\alpha}{L_k} = \epsilon\hat{q}_k \ .$$

□

Equations (1) and (2), and Lemma 5.7 imply the following corollary

Corollary 5.8. *Let k be a light growing phase. Then, with probability $1-\delta/2n$, for $\epsilon \leq 1/4$,*

$$(1-2\epsilon)\hat{q}_k \leq (1-\epsilon^2)(1-\epsilon)\hat{q}_k \leq p \leq (1+2\epsilon^2)(1+\epsilon)\hat{q}_k \leq (1+2\epsilon)\hat{q}_k \ ,$$

and

$$(1-2\epsilon)\hat{q}_k \leq (1-\epsilon^2)(1-\epsilon)\hat{q}_k \leq p_{ne} \leq (1+\epsilon)\hat{q}_k \ .$$

The next lemma guarantees that, with high probability, each light growing phase indeed succeeds.

Lemma 5.9. *With probability at least $1 - \frac{\delta}{2n}$, in every light growing phase k we have $V \subset Z_k$, where V is the set of variables in f and not in h^k.*

Proof. Let g be the clause of the variables in V and $x_i \in V$. We have that

$$\begin{aligned} p_{ne} &= \Pr[f(x) = 1, h^k(x) = 0] \\ &= \Pr[g(x) = 1, h^k(x) = 0] \\ &= \Pr[g(x) = 1] \Pr[h^k(x) = 0] \\ &\geq \Pr[x_i = 1] \Pr[h^k(x) = 0] = p_i \Pr[h^k(x) = 0] \ . \end{aligned}$$

By Corollary 5.8, $p_{ne} \leq (1 + \epsilon)\hat{q}_k$, with probability $1 - \frac{\delta}{2n}$. (Note that this holds for a phase, independently of any particular $x_i \in V$.) So, for every $x_i \in V$, we have

$$p_i \leq (1 + \epsilon)\hat{q}_k / \Pr[h^k(x) = 0] \ ,$$

with probability $1 - \frac{\delta}{2n}$, and therefore $x_i \in Z_k$, with probability $1 - \frac{\delta}{2n}$. □

The following theorem states that indeed, with probability $1 - \delta$, our algorithm has at most $2n$ phases.

Theorem 5.10. *For any product distribution D, algorithm* ON_APPROX$(\overline{\epsilon}, \delta, D)$*, with probability at least $1 - \delta$, terminates after at most $2n$ phases.*

Proof. We say that the algorithm fails if among the first $2n$ phases, either (1) there is a heavy growing phase, or (2) there is a light growing phase that does not succeed. Clearly if the algorithm does not fail then it has at most $2n$ phases since every growing phase among the first $2n$ phases must be light and succeed.

We claim that the probability that the algorithm fails is at most δ. To establish the claim, notice that by Lemma 5.6 the probability that there is a heavy growing phase among the first $2n$ phases is smaller than $\delta/2$. Furthermore, by Lemma 5.9 and the union bound, the probability that a light growing phase, among the first $2n$ phases, fails is at most $\delta/2$. The claim then follows since failure is the union of the event that there is a heavy growing phase among the first $2n$ phases, and the event that there is a light growing phase among the first $2n$ phases that fails. □

Last we prove a bound on the cost of the algorithm.

Lemma 5.11. *With probability $1 - \delta$, for any phase k, the expected cost of h^k is at most $(1 + 16\epsilon)\mathrm{OPT}(f, D)$, for $\epsilon \leq 1/4$.*

Proof. The statement clearly holds for h^1. We prove that, at the end of a light growing phase k, the statement holds for h^{k+1} with probability $1 - \delta/2n$.[9] Since after a terminating phase the cost only decreases, and since with probability $1 - \delta/2$ we have only terminating phases and light growing phases, then the lemma follows.

[9] The dependency is on the same event that guarantees that the phase succeeds.

Note that $p_e = \Pr[M_e] = \Pr[M_e|M]\Pr[M]$. Since the phase is light then, by Corollary 5.8, with probability $1-\delta/2n$, we have $p = \Pr[M] \leq (1+2\epsilon)\hat{q}_k$, and $p_{ne} \geq (1-2\epsilon)\hat{q}_k$. So $p_e = \Pr[M_e|M]\Pr[M] \leq \epsilon^2(1+2\epsilon)\hat{q}_k \leq \epsilon\hat{q}_k$, for $\epsilon \leq 1/4$. Therefore,

$$\begin{aligned}\Pr[f(x)=1] - \Pr[h^k(x)=1] &= \Pr[f(x)=1, h^k(x)=0] - \Pr[f(x)=0, h^k(x)=1] \\ &= p_{ne} - p_e \\ &\geq ((1-2\epsilon)-\epsilon)\hat{q}_k \\ &\geq (1-3\epsilon)\hat{q}_k.\end{aligned}$$

Since we select $x_{i_{k+1}}$ such that $\Pr[x_{i_{k+1}}=1] \leq \frac{(1+\epsilon)\hat{q}_k}{\Pr[h^k(x)=0]}$, we have

$$\begin{aligned}\Pr[h^{k+1}(x)=1] &= \Pr[h^k(x)=1] + \Pr[h^k(x)=0\ ,\ x_{i_{k+1}}=1] \\ &\leq \Pr[h^k(x)=1] + (1+\epsilon)\hat{q}_k\ .\end{aligned}$$

Consider the following randomized hypothesis h' which selects between h^{k+1} and h^k,

$$h' = \frac{1-3\epsilon}{1+\epsilon}h^{k+1} + \frac{4\epsilon}{1+\epsilon}h^k.$$

For h' we have that $\Pr[h'(x)=1] \leq \Pr[f(x)=1]$. Consider the set of variables $h^k \cup Z_k$; then, by Lemma 5.9, both h' and f are subsets of this set, and h' is in fact $H_{k+1,\frac{1-3\epsilon}{1+\epsilon}}$ on that set. Therefore, by an argument as in the proof of Lemma 5.3, we have that $E[cost(h')] \leq E[cost(f)]$. On the other hand, $E[cost(h')] \geq \frac{1-3\epsilon}{1+\epsilon}E[cost(h^{k+1})]$. Therefore, $E[cost(h^{k+1})] \leq (1+16\epsilon)E[cost(f)]$, for $\epsilon \in (0, 1/4]$. □

The following theorem summarizes the the main result of this section.

Theorem 5.12. *For any product distribution D and any clause f, with probability $1-\delta$, the number of mistakes of* ON_APPROX$(\bar{\epsilon}, \delta, D)$ *is at most $O(n\alpha) = O(\frac{n}{\epsilon^2}\log(n/\delta))$ and, at any time t, we have $E[cost(h^t)] \leq (1+\bar{\epsilon})E[\mathrm{OPT}(f, D)]$.*

We remark that one can obtain a similar result for a general distribution D where we have a conditional distribution oracle. The cost of the online algorithm then is at most four times the cost of OPT.

References

[1] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *JCSS*, 18(2):155–193, April 1979.

[2] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *SIGMOD*, 2004.

[3] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.

[4] A. Bar-Noy, M. M. Halldórsson, and G. Kortsarz. A matched approximation bound for the sum of a greedy coloring. *IPL*, 71(3-4):135–140, 1999.

[5] M. Charikar, R. Fagin, V. Guruswami, J. Kleinberg, P. Raghavan, and A. Sahai. Query strategies for priced information. In *32nd STOC*, pages 582–591, 2000.

[6] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Proceedings IEEE INFOCOM* , 2003.

[7] O. Etzioni, S. Hanks, T. Jiang, R. M. Karp, O. Madani, and O. Waarts. Efficient information gathering on the internet. In *FOCS*, 234–243, 1996.

[8] O. Etzioni, S. Hanks, T. Jiang, and O. Madani. Optimal information gathering on the internet with time and cost constraints. *SIAM J. Comput.*, 29(5):1596–1620, 2000.

[9] U. Feige, L. Lovász, and P. Tetali. Approximating min-sum set cover. In *APPROX*, 94–107, 2002.

[10] J. M. Hellerstein and M. Stonebraker. Predicate migration: optimizing queries with expensive predicates. In *SIGMOD*, 267–276. ACM Press, 1993.

[11] A. T. Kalai and S. Vempala. Efficient algorithms for online decision problems. In *COLT*, 26–40, 2003.

[12] M. Kharitonov. Cryptographic hardness of distribution-specific learning. CLOT, 372–381, 1993.

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

[14] D. J. Lizotte, O. Madani, and R. Greiner. Budgeted learning of naive-bayes classifiers. In *UAI*, 378–385, 2003.

[15] O. Madani, D. J. Lizotte, and R. Greiner. Active model selection. ICML, 2004.

[16] R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, 1995.

[17] K. Munagala, S. Babu, R. Motwani, and J. Widom. The pipelined set cover problem. ICDT, 2005.

[18] R. Smorodinsky and M. Tennenholtz. Overcoming free riding in multi-party computations - the anonymous case. Unpublished, 2004.

[19] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[20] L. G. Valiant. Learning disjunctions of conjunctions. *IJCAI*, 560–566, 1985

A Estimating the residual probability of an attribute

For our analysis in Section 4 and 5 we need the following version of the Chernoff bound [1].

Theorem A.1 ([1])**.** *Let $X_1, \ldots, X_m$ be i.i.d. $\{0,1\}$ random variables, where $E[X_i] = p$. Let $b = \sum_{i=1}^m X_i$. Then,*

1. *For all $\beta \in (0,1]$ we have* $\Pr[b \leq (1-\beta)pm] \leq e^{-\beta^2 pm/2}$
2. *For all $\beta \in (0,1]$ we have* $\Pr[b \geq (1+\beta)pm] \leq e^{-\beta^2 pm/3}$
3. *For all $\beta \geq 1$ we have* $\Pr[b \geq (1+\beta)pm] \leq e^{-\beta^2 pm/(2+\beta)}$
4. *(combining the above 2 and 3) For all $\beta > 0$ we have* $\Pr[b \geq (1+\beta)pm] \leq e^{-\beta pm/3}$

An immediate consequence we obtain the following lemma.

Lemma A.2. *For $q \in [0,1]$ and $F \geq 0$ let $m = F/q$. Let $X_1, \ldots, X_m$ be i.i.d. $\{0,1\}$ random variables, where $E[X_i] = p$. Let $b = \sum_{i=1}^m X_i$ and $\hat{p} = b/m$. Then, for any $\epsilon \in (0,1)$, with probability $1 - 4e^{-\epsilon^2 F/6}$,*

1. *If $p \geq 2q$ then $b \geq F$.*
2. *If $p \leq q/2$ then $b < F$.*
3. *If $p \geq q/2$ then $\hat{p} \in [(1-\epsilon)p, (1+\epsilon)p]$.*

Proof. For case 1, note that $E[b] = pm \geq 2F$. This implies by case 1 of Theorem A.1 that

$$\Pr[b < F] \leq \Pr[b < (1/2)E[b]] \leq e^{-(1/2)^2 pm/2} \leq e^{-qm/4}$$

For case 2, note that $E[b] = pm \leq F/2$. This implies by case 4 of Theorem A.1 that

$$\Pr[b > F] = \Pr[b > qm] = \Pr[b/m > (1+\lambda)p] \leq e^{-\lambda pm/3}$$

where $\lambda = (q-p)/p$. Since $p \leq q/2$, we have that $q - p \geq q/2$, which implies that $\lambda p \geq q/2$ Therefore the failure probability is at most $e^{-qm/6}$.

The proof of case 3, uses cases 2 and 3 of Theorem A.1. This will imply that the failure probability is at most $e^{-\epsilon^2 qm/4} + e^{-\epsilon^2 qm/6}$.

The theorem follows since by our definitions $F = qm$. ∎